PHILIPS

P|800

P855/P860
Reference Manual

Preliminary

data systems

P|800

P855/P860
Reference Manual

Preliminary

data systems

The information in this handbook is for information purposes and is subject to change without notice.

A publication of
NV Philips-Electrologica
Industry Group Small Computers
Apeldoorn - The Netherlands

Pub. No. 5122 991 18351

December 1971

The P855/P860 Reference Manual consists of five parts:

Part 1: Hardware
Part 2: Operation
Part 3: Instruction Set
Part 4: System Software
Part 5: Interface

Part 1 contains general information on the hardware of the P855 and P860 computers. Differences between the two machines are clearly indicated.
Readers have only to be familiar with computer principles to understand the sections contained in this part.

Part 2 provides the user with the information he needs to operate the P855 and P860 computers and peripheral devices. Some information on both hardware and software is included, though only to the extent required for proper operation of the system by a user who may, or may not, be a trained computer operator.

Part 3 includes all instructions of the P855 and P860 computers, it also contains the assembly format (syntax) per instruction to be used by the programmer when writing his program in assembly language. It is assumed that the reader has a basic knowledge of computer programming.

Part 4 provides the user with the information needed for programming his system; it includes details on monitors, assembly language, linkage editor, update programs, etc. It is assumed that the reader has basic knowledge of computer programming.

Part 5 provides the user with the information to install the P855 or P860 into an already existing system or to connect them to device control units. Readers need to have a basic computer experience.

Great care has been taken to ensure that the information contained in this handbook is accurate and complete. However, should any errors or omissions be discovered, or should any user wish to make a suggestion for improving this handbook, he is invited to write his comments on the sheet provided at the end of the book and send it to:

Manual Writing Small Computers

at the address on the opposite page.

PART 1 HARDWARE

PART 2 OPERATION

PART 3 INSTRUCTION SET

PART 1 HARDWARE

Introduction

The P855 and P860 are general-purpose digital mini-computers ideally suited for industrial and scientific applications, such as process control, test and measurement, data acquisition, numerical control and scientific problem-solving. For this reason, the central processor is available in two different versions: stand-alone model and 19-inch rack version.
The basic central processor of both machines contains at least 4k of memory; the **P855** can be expanded up to 16k, the **P860** up to 32k words.
A number of I/O facilities are available: programmed channel is standard and, depending on the type of peripheral equipment connected, Multiplex Option, Direct Memory Access (DMA) Channel or Memory Increment Data Break (MIDB) can be added. For analog or digital processes MIOS or DIOS can be used to connect external devices to the P855/P860.
The basis of the interrupt system consists of eight interrupt levels one of which is able to accept up to 16 signals (to be inhibited or enabled by means of a mask-register). The others accept one signal each. This standard system can be expanded with groups of additional interrupt lines. The maximum interrupt system has 63 (48 + 15) interrupt lines.
For added flexibility and adaptation, a number of extra hardware options are available, such as Real Time Clock, Memory Protection, Power Failure/Automatic Restart.
The instruction set used for programming in Autocode is based on the 16-register structure of the central processor. This makes it a very powerful and versatile set, giving the programmer a wide range of possibilities and ensuring fast execution of his programs.
Different monitors are available for various applications. For non-disc configurations a Basic and a Basic Real Time Monitor can be supplied. For disc systems, there are also two different monitors: Disc Real Time Monitor and a Monitor for keyboard-oriented disc systems.

Both machines accept the same Autocode language. FORTRAN is also available, in different versions, starting with one which can be used in a 4k configuration. Updating can be done on the statement-level or, by means of the Text Editor, on the character-level. Finally, a Program Debugging Package is also available.

All these features are treated in more detail in the following sections of this book.

Hardware

The following figure shows the standard and optional hardware features of the P855 and P860.

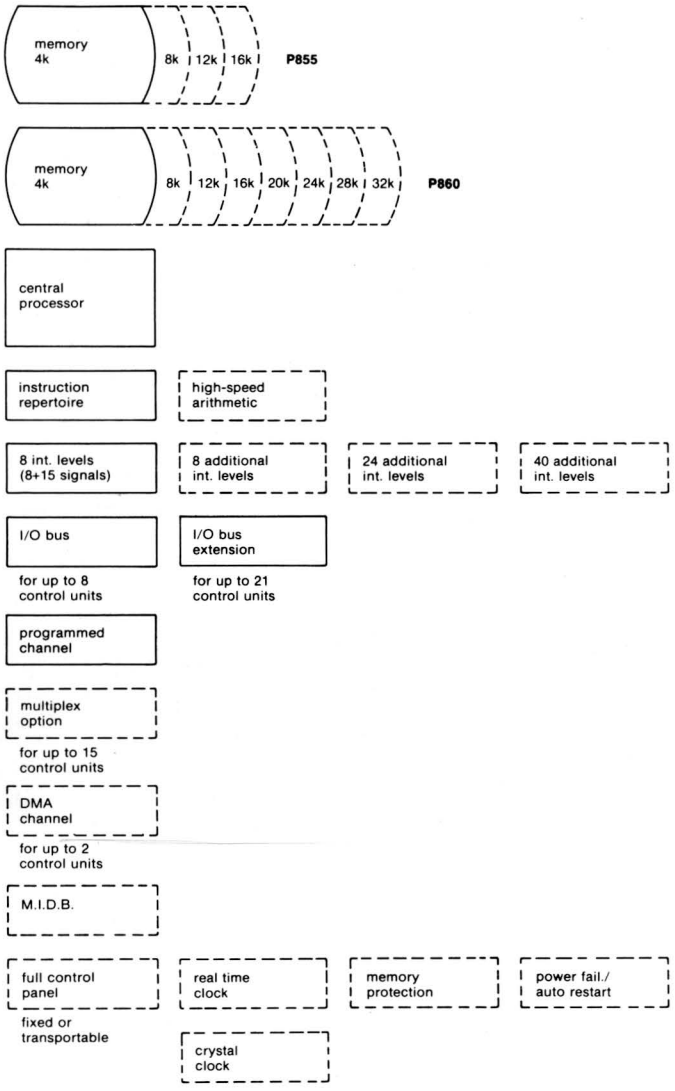


Figure 1 P855 and P860 Hardware Configuration Chart

MEMORY UNIT

The data path is 16 bits wide and data are trasferred in parallel. The internal temperature, humidity, etc. of the memory unit are constantly monitored and if they exceed acceptable limits, the unit is made inoperable without disturbing existing data. Switching on or off the power supply to the memory unit has been sequenced, so that stored data are not affected.

On option, a Read-Only Memory is also available.

Capacity

P855: minimum 4k words (k = 1024).
Can be extended up to 16k in 4k modules.
P860: minimum 4k words (k = 1024).
Can be extended up to 32k in 4k modules.

Cycle Time

The time taken for one complete read/write cycle is:
P855: 1.2 microseconds.
P860: 0.84 microseconds.

REGISTER STRUCTURE

Figure 2 shows a simplified block diagram of the main hardware units and registers of the central processor.
The following registers are accessible by software:

P-register:
A 16-bit register (register 0) used as an instruction counter to contain the address of the next program instruction to be executed. The contents of this register are incremented by 2 each time a one-word instruction is executed, by 4 each time a two-word instruction is executed.

Registers 1 to 14:
Fourteen 16-bit general-purpose registers which can be used as accumulators (to contain the intermediate results of computation), or as address or index-registers.

Register 15:
A 16-bit register used as a stack pointer for the interrupt system.

CR-register:
A 2-bit register used to indicate a condition after the execution of some instructions.

GF-register:
A general indicator (4 to 8 bits long) used to indicate control or interrupt states. The bits can be individually set or reset by program.

IM-register:
An optional 16-bit register which can be used as an interrupt mask to inhibit or enable interrupt signal lines.

MK-register:
One or two optional 16-bit registers which can be used as memory protect keys if the memory protect option is included in the system.

PL-register:
A 6-bit register used as a priority level register.

ARITHMETIC UNIT

The arithmetic unit data path is 16 bits wide. All bits are processed in parallel. This unit performs all arithmetic and logical functions in true binary using two's complement notation.

DATA FLOW

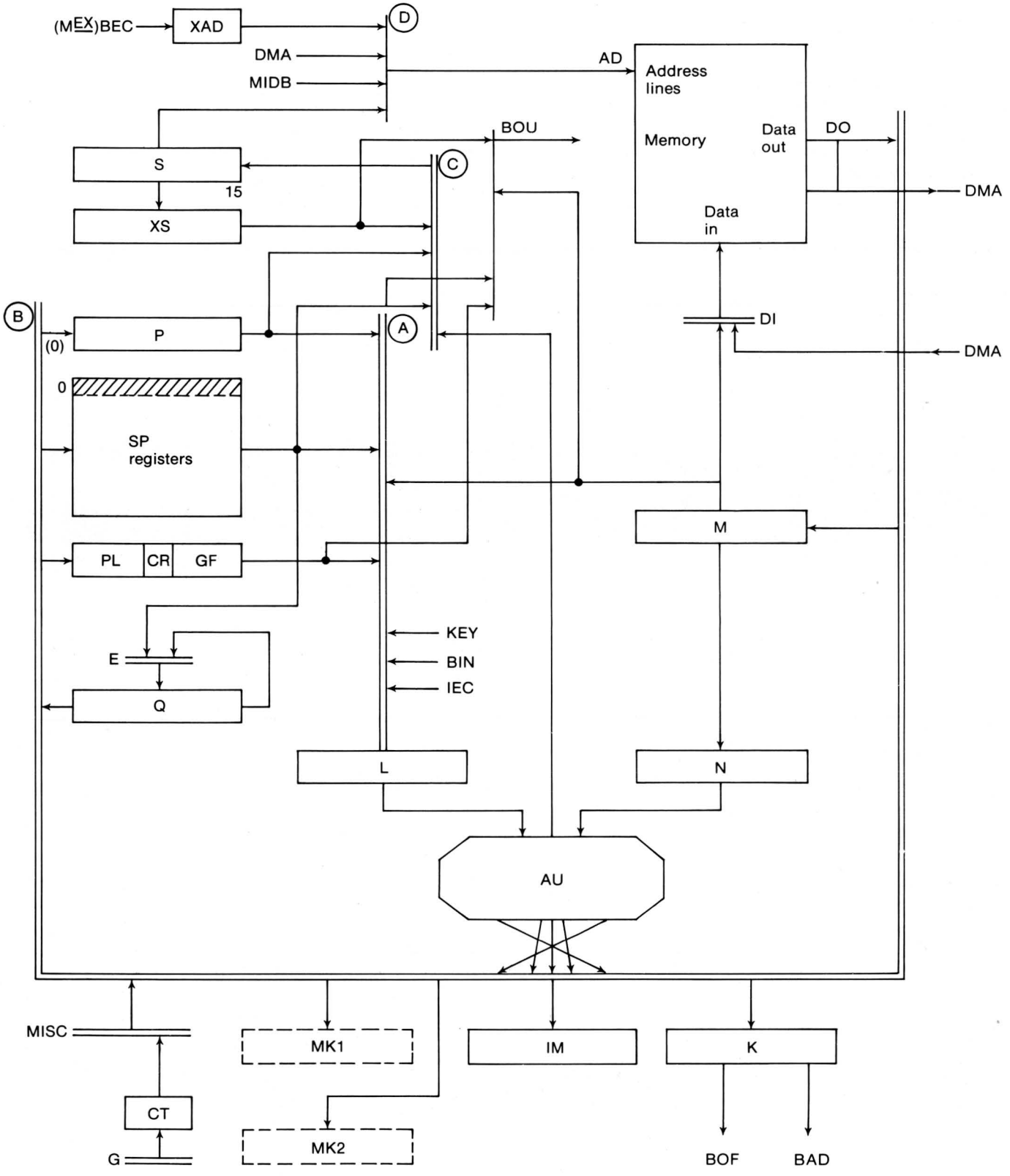


Figure 2 Data Flow

Simplified block diagram of registers and hardware units
The functions of these registers are as follows:

S-register:
This is the register which holds the memory address.

XS-register:
This is the auxiliary address register which holds the memory address relevant to the running instruction during a multiplex data exchange.

P-register:
This is the program counter which holds the address of the next program instruction to be executed. Only the 15 most significant bits refer to the 16-bit words memory address.
The contents of this register are incremented by 2 each time an instruction is executed.

SP-register:
Physically the P-register is the first register of the scratch-pad (register 0). This scratch-pad is made up of four random access memory modules each 16×4-bit words long. They are hard wired to form 16×16-bit registers. Two of the registers have unique functions, register 0 and register 15.
Register 15 is used as the stack pointer for the interrupt system. The remaining 14 registers can be used as accumulators, address of index registers.

PL/CR/GC-registers:
This is the 16-bit register which holds the program status word. The PL part of 6 bits gives the priority level of the running program (priority 0-64). CR is the 2-bit condition register. GC are the (max 8) general control bits.
bit nr.
8 run
9 interrupt enable
10 control panel interrupt
11 power failure (opt)
12 real time clock (opt)
13 program interrupt (LKM, stack overflow, not available optional instructions)
14 memory protect error
15 user mode indicator

Q-register:
This is a 16-bit register used during double length shift operations.

L-register:
Input register for the arithmetic unit.

N-register:
Input register for the arithmetic unit.

MK-register:
Memory Key Register (MK1 and MK2). These are the memory protect registers. MK1 indicates the protected 1k areas from 1-16k words. MK2 the same for 17-32k.

IM-register:
This is the interrupt mask register. It holds the mask bits for the 16 programmable interrupt lines connected to one priority level.

K-register:
This register holds the instruction read from memory. If the particular input/output instruction is used, this register is connected to the Bus Function and Bus Address lines.

M-register:
This is the memory input/output buffer register.

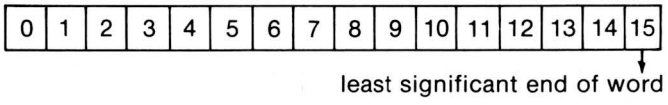
XAD-register:
The multiplex break lines are connected to this register which generates the multiplex control words.

CT-register:
This is the register which counts the number of shift positions of a shift instruction.

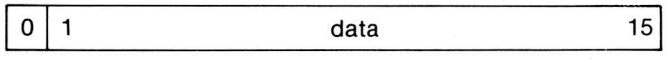
Remarks
In case of a multiplex channel operation the scratch-pad is short circuited for outgoing and incoming data.

DATA FORMAT

The basic format used to transfer data into or out of the central processor is the 16-bit word. Bit positions are numbered 0 to 15 as follows:

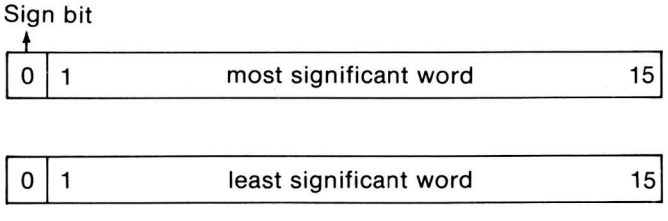


Data may be represented as single precision signed integer contained in one word as follows:



where:
bit 0 is used as a sign bit which is set to zero for positive data, and bits 1 to 15 contain the data.

If the optional double length instructions are included in the instruction set, data may also be represented as a double precision signed integer which will be contained in two words as follows:



The sign bit of the second word (least significant word) is not used and is usually set to zero, this means that thirty bits are available for data representation.

Logical Data
Logical data can also be represented in a single data word. This type of data is generally not treated arithmetically, but logically by boolean operations such as 'AND', 'OR' or 'eXclusive OR'. In this case bit 0 of the word does not represent the sign bit, but the first of sixteen conditions.

INSTRUCTION FORMATS

There are two basic instruction formats which are indicated by the value of bit 0 in the instruction word.

Format 0:
These instructions are contained in one word and bit 0 is set to 0 to indicate this type of instruction. The types of instruction which can use this format are: Constant instructions (short format), Shift instructions, Branch instructions, Control instructions, I/O instructions and Miscellaneous instructions.

Format 1:
These instructions are contained in one or two words and bit 0 is set to 1 to indicate this type of instruction. The types of instruction which can use this format are: Register to Register instructions, Memory Reference instructions, Constant instructions (long format), Branch instructions and Miscellaneous instructions.

INSTRUCTION SET

The instruction set is very powerful and versatile and includes the usual load, store, branch, arithmetic, shift, logical and I/O instructions. Optional instructions for double length arithmetic, multiply and divide may be included to extend the range of the instruction set. The instruction set can be divided into six main groups, as follows:

Memory Reference Instructions
These instructions are all written in format 1, with the exception of the relative branch instructions (format 0), and the result can be stored either in memory or in a register.

Register to Register Instructions
These instructions are all written in format 1 and by optimum use of these instructions, in conjunction with the general purpose registers to store the intermediate results, the overall execution time of the program can be held to a minimum.

Constant Instructions
These instructions can be written in either format 0 or format 1. In format 0 instructions the constant is contained in the least significant 8 bits of the instruction word, in format 1 instructions all sixteen bits of the second word of the instruction can be used to contain the constant.

Shift Instructions
These instructions allow the contents of the registers to be shifted a number of positions to the right or left. For logical data all 16 bits are shifted. Shifting can be circular or not, i.e. it is possible to reintroduce bits shifted out at one end, into the other end of the register. When arithmetic data are shifted, the sign bit is left unchanged and the instruction is the same as multiply or divide by 2 times the number of positions shifted.

Input/Output Instructions
These instructions provide for the control by the central processor of the operations on the external devices.

Miscellaneous Instructions
These instructions include possibilities for handling the interrupts, executing a halt, setting the operating mode and giving control to the monitor.

ADDRESSING

In addition to direct addressing, for memory reference instructions, the effective address may be specified by using indexed addressing, indirect addressing or a combination of both.

EXECUTION TIMES

The execution time of an instruction depends on a number of variables. For instance, format 1 instructions can be contained in either one or two words, the result of the instruction can be stored either in a register or in memory and the mode of addressing all affect the execution time of an instruction. Individual instructions all need a certain amount of time in which to perform their function and when calculating the execution time of a program, the extra time needed by the variables must be taken into account.

The execution time of all the instructions is given in full in Part 3 of this book, but below is given a table showing typical execution times, in microseconds, for some of the main types of instruction. It will be seen from this table that although indexing does not increase the execution time, the use of indirect addressing increases the execution time by 1.2 microseconds for the P855 and by 0.84 microseconds for the P860.

Instruction Type	Example	P855		P860	
		Result in Register	Result in Memory	Result in Register	Result in Memory
8-bit constant (short format)	LDK – Load Constant	1.92		1.56	
16-bit constant (long format)	ADK – Add Constant	2.4		1.68	
Register to Register (short format)	CIR – One's Complement	1.92		1.56	
Address in Register (short format)	ABR – Absolute Conditional Branch	1.92		1.56	
Address in next word (long format)	AD – Add Memory Operand	3.6	4.8	2.52	3.36
Address indexed (long format)	SU – Subtract Memory Operand	3.6	4.8	2.52	3.36
Address Indirect (long format)	OR – Logical OR with Memory Operand	4.8	6.0	3.36	4.20
Address Indexed and Indirect (long format)	AN – Logical AND with Memory Operand	4.8	6.0	3.36	4.20

Note: Short Format means either a format 0 instruction or a format 1 instruction contained in one word.
Long Format means a two-word format 1 instruction.

INPUT/OUTPUT

Data are transferred into or out of the central processor by a standard I/O Bus which consists of a sixteen data lines and the signal and address lines necessary to effect data transfers. Data transfers along the I/O Bus are controlled by the Programmed Channel.
For further details about all the I/O facilities see the Interface section.

PROGRAMMED CHANNEL

The programmed channel is a sequence of instructions which controls transfers of data along the I/O bus. It can service up to 48 device control units and all transfers are effected one word at a time. These transfers take place either within or during program interrupts.

Transfer rate: **P855:** 14Kc - 28Kc.
P860: 20Kc - 40Kc.

MULTIPLEX OPTION

This option allows data to be transferred in blocks of words instead of one word at a time. Up to 15 device controllers can be connected to the basic Multiplex option.
All control signals and data are transferred between the central processor and the peripheral devices via the standard I/O bus under control of the programmed channel.

The transfers take place by either stealing memory cycles between instructions or, if the instruction exceeds 5 memory cycles, within an actual instruction. The programmer has only to state the block length, location of data in memory and whether the transfer is to be in words or characters and the multiplex option will then effect the transfer.

Transfer rate: **P855:** 92Kc - 165Kc.
P860: 150Kc - 240Kc.

DMA-CHANNEL

This channel performs the full control of direct memory access data transfer. The Direct Memory Access is a logical switch between the central processor and the memory, which is locked during data transfers to establish a direct connection between the DMA channel and the memory.
The channel has its own registers and it can perform the necessary arithmetic functions for address control. Two control units can be connected.

Transfer rate: **P855:** 415Kc - 830Kc.
P860: 600Kc - 1.2Mc.

CONTROL UNITS

All peripherals and external devices are connected to the I/O Bus via device control units. Each control unit has a unique address which it can decode from the address lines of the I/O Bus. It can also decode the function it is to perform from the Function lines of the I/O Bus.

Once the address has been recognised and the function decoded, the control unit provides the start/stop and other control signals to the device as well as the synchronisation for the transfer of data.

The following table shows the number of devices which can be attached to a control unit; the last three columns indicate with an X the channels to which the control units can be connected.

Control Unit	Number of Devices per Channel	Programmed Channel	Multiplex Channel	DMA Channel
Operator's Typewriter	1	X	X	
Punched Tape Reader	1	X	X	
Tape Punch	1	X	X	
Line Printer	1	X	X	
Magnetic Tape (9 track)	4		X	X
Magnetic Tape (cassette)	2	X	X	
Moving Head Disc	2		X	X
Fixed Head Disc	4		X	X
Plotter	1	X	X	
CRT Display	1	X	X	

INTERRUPTS

There are two types of interrupt, internal interrupts which are generated within the central processor and external interrupts which are generated by the device control units.

Internal Interrupts
The first three interrupts listed below are all associated with the hardware options which are available for the central processor, the other three are standard. The interrupts are:

- *Power Failure:* occurs when the power fails and should be connected to the highest priority interrupt level so as to take priority over any other operation.
- *Real Time Clock:* occurs when a timing signal is received from this clock.
- *Memory Protect:* occurs when a user program tries to address the protected area.
- *Control Panel:* occurs when the operator pushes the INT button.
- *Operation Code:* occurs when the user tries to execute an optional instruction which is not included in his package (Multiply, Divide, Double Add, Double Subtract, Floating-Point Instructions) or when a Link to Monitor (LKM) instruction is executed.
- *Stack Overflow:* occurs when the stack reaches location 128 (i.e. the stack is full). See paragraph on Stack.

INTERRUPT SYSTEM

This feature is part hardware and part software and allows a program to be interrupted by either internal or external interrupt signals, controlling the servicing of the interrupt itself. The system is a multi-priority level system.

As standard, 8 interrupt lines are available. These can be expanded with 8, 24 or 40 additional interrupt lines, so the maximum number of lines is 48.
Of the 8 standard interrupt lines, one line is able to accept up to 16 signals. Each one of these signals, connected to the common interrupt line, can be inhibited or enabled by means of a 16-bit mask register in the central processor. This register is loaded by program and in this way the programmer can control which of these 16 interrupts his program will answer. The programmer can load the state of the signal lines into one of the first seven general purpose registers at any time during the execution of his program. Each bit of the loaded word represents the state of the corresponding line: a 0 indicates a line which is not active or masked, a 1 indicates an active or unmasked line. The other seven standard and 40 optional lines are each connected to individual levels. When levels become active they are coded in binary, by hardware, to give a number from 0 to 47. Thus 48 + 15 = 63 interrupt signals can be serviced.

Another way to use the system is by assigning a priority level to each program, a 6-bit hardware register holding the priority level. Thus, a maximum of 64 priority levels can be had. Level 0 has the highest priority, 63 the lowest priority and switching from one program to another is controlled by software.

During the execution of each instruction, the state of all the interrupt lines is sampled. If one or more lines are active, the level with the highest priority is compared to the priority level of the running program. If it has a priority level equal to or less than the running program, nothing happens and the running program proceeds normally. However, if the priority level is higher than the running program an interrupt occurs and the program with the higher priority is either started or allowed to continue. This interrupt is also serviced by the stack handling routine.

STACK HANDLING

The interrupt system makes use of automatic stack handling techniques to process both internal and external interrupts. This means that, for each interrupt, the contents of the P-register, Program Status Word (PSW) and the mask register, are always saved. The contents of the other registers and the needed parameters are selected by the programmer. Access to the stack, where all this is stored, is made using register A15 as a stack pointer, the contents of this register always pointing to the next available memory address in the stack. Each time new data are loaded into the stack the contents of the stack pointer are automatically decremented so as to point to the next available stack address and, conversely, when data are retrieved from the stack, the contents of the pointer are incremented. The stack is always filled towards the dedicated area of the memory (locations 0 to 128) and before a program is started, the starting address of the stack area must be loaded into register A15. In the Programming section it is explained how this address can be calculated.
When location 128 is addressed by the stack handling routines, a bit is set in the GF-register to indicate that the stack is full. If the routines try to address beyond this location, a 'stack overflow' interrupt is generated to prevent the dedicated memory locations being overwritten.
Hardware and software can access the stack and more than one interrupt can be in the stack at any one time.

OPTIONS

The following optional hardware features are available for the central processor. These options can be included at system generation time or they can be individually added later, when the need for them arises, without major rewiring. The options are:

Power Failure/Automatic Restart
This option provides the means to detect a power failure in time and automatically restart a program, without loss of information.

If the power fails completely or drops below the minimum level for error free operation, an interrupt is generated 5 milliseconds before the d.c. operating voltages fall below normal. During this time all information relevant to the current instruction is stored in memory.
Provided, that the control panel key is in the 'LOCK' position, the program is restarted automatically when the power is restored, i.e. all the hardware is reset, the stored information is retrieved and the interrupted instruction is processed again.

Control Panel
The control panel is fitted as standard to the stand alone model, but for the rack mounting version it is an optional extra.

The control panel contains switches and indicators which allow the operator or programmer to manually load or display the contents of the registers or memory locations. It is possible to interrupt the running program, to make any necessary alterations to it, and to restart it again at a selected program instruction without reloading the program.

A keylock is fitted which in one position permits all the switches on the panel to function, and in the other positions only the INT button.

Real Time Clock
This feature provides an interrupt each time a timing signal is received. The timing signal can be from either the 20 millisecond internal timer (which is tied to the 50c/s mains supply) or from an external timer.

On option, a crystal timer with an interval time of 1, 2, 5, or 10 milliseconds is available.

The clock can be started or stopped by software and is programmable in the same way as a peripheral device.

Additional Interrupt Levels
The standard 8 interrupt levels of the basic processor can be increased up to maximum of 48 by adding 8, 24 or 40 extra levels. These extra levels can be used for either internal or external interrupts.

DMA Channel
The basic features of this option and the extension to it have been described in a previous paragraph. Full details can be found in the Interface section.

Multiplex Option
The basic features of this option and the extension to it have been described in a previous paragraph. Full details can be found in the Interface section.

Memory Increment Data Break (MIDB)
This option provides a simple way for external equipment, such as Liquid Scintilation analysers and X-ray Spectrometers etc., to generate an interrupt in the central processor. It is used in conjunction with an external adapter which in turn uses the standard I/O Bus, under control of the programmed channel, for all control signals.

The adapter allocates a particular channel to dedicated memory locations (two per channel) and sends the increment memory signals, using normal I/O instructions, to the central processor. The number of channels is only limited by the amount of memory space available.

Signals from the adapter are controlled by program for each signal, the memory location associated with the channel is incremented automatically using the data break feature. If overflow occurs during an increment, all other increments are

inhibited for the extra time required to process the overflow; in this way no increments are lost.

Incremental counts may be taken for a pre-determined time set by the programmer. When this predetermined time is reached, the resulting counts in the memory locations may be processed and output on a selected peripheral device.
Full details can be found in the Interface section.

Memory Protection

If the memory protection option is present, the P855/P860 can work in system or user mode.
In system mode, all instructions may be executed, and all main memory is accessible.
In user mode, certain instructions are not permissible (system mode instructions) and their occurrence will cause an interrupt.
In addition, addressing of protected memory blocks (1k words each) indicated by the MK register, will also cause an interrupt.

DIOS (Digital Input/Output System)
DIOS is a general purpose system which acts as a TTL (Transistor to Transistor Logic) interface between a P855 or P860 and any external equipment. The principal function of the DIOS is to control the exchange of 16-bit data words (in both directions) via the programmed channel. Three versions of the basic DIOS are available, consisting of logic circuitry, each handling four sixteen-bit words by means of standard I/O instructions. These basic systems are:

- DIC: attached to the I/O bus, controls four sixteen-bit gated input words;
- DOC: attached to the I/O bus, handles four sixteen-bit buffered output words;
- DIOC: attached to the I/O bus, controls two gated input and two buffered output words.

The basic system is intended for equipment whose voltage levels and currents are similar to those used by the processor interface.

The basic system may be extended by some optional features:

- For input lines, a sixteen-bit buffer register and change-of-state detection;
- For both input and output lines, level adaptation and galvanic isolation circuitry.

The input buffer consists of two sixteen-bit registers plus one flip-flop per word for the external CALL line, which are loaded by external equipment. The output of the sixteen-bit buffer is connected to the corresponding input lines of the Basic DIOS. The change of state detection provides for an external exchange state (0 to 1).

Level adaptation allows the system to exchange TTL signals at the logical interface level of the particular external devices. Electrical isolation is provided where the ground voltage of the external device differs from that of the CPU, or where the power supply or transmission lines are noised at high level.

Two methods of operation are possible:

- *Software control:* data transfer is controlled by the program, using a single I/O instruction per word exchange;
- *External control:* a seventeenth line from the external system to each word permits a data exchange request on the corresponding data path. When one or more of these lines is activated, the DIOS generates an interrupt signal to the CPU.

Modular Input/Output System (MIOS)
MIOS is an interface system for application in process control to act as an interface between the computer and any external equipment. It performs such tasks as:

- signal acquisition and conditioning (including filtering and rationalisation of input levels;
- sequential interrogation of the various inputs;
- analog-to-digital conversion of inputs representing physical quantities, for acceptance by the digital computer as data words;
- provision of outputs for display, control or data acquisition purposes.

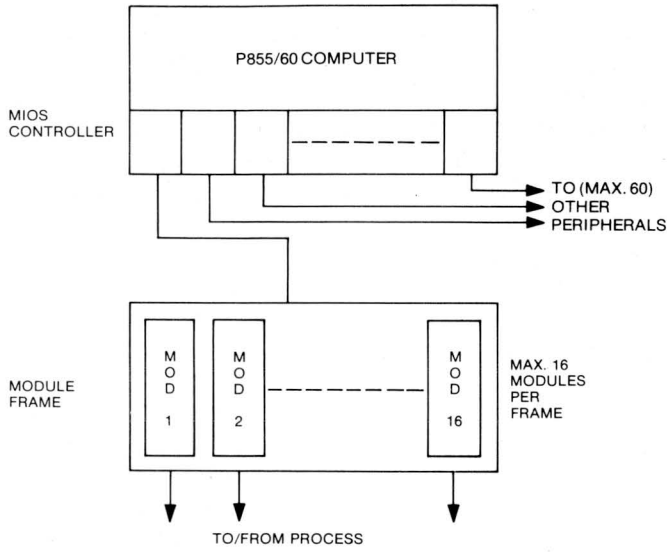
The MIOS system consists basically of controller cards and module frames containing various MIOS modules. The controller cards are inserted in the standard C.U. sockets of the I/O Bus and each card will drive up to 16 MIOS modules. These modules are connected to the external equipment.
The following modules are available:

- Digital Input Isolation Module
- Digital Input Solid-state Module
- Digital Input Counter Module
- Digital Input Priority-interrupt Module
- Digital Output Solid-state Module
- Pulse Output Control Module
- Analog Output Fast Module
- Analog Output Control Module
- Analog Input Low-level Module
- Analog Input High-level Module
- Analog Input Solid-state Module

Other MIOS items available are:

- Data Amplifiers
- Analog Input Data Concentrator
- Analog-to-Digital Converter

The following drawing shows the basic set-up of the system:



Per Module:

16 Solid-state digital inputs
16 Galvanically isolated digital inputs
16 Program interrupt digital inputs
2 Seven-bit counter inputs
16 Medium-level solid-state digital outputs
1 Pulse output
2 High-speed analog outputs
1 Analog output for control purposes.

More detailed information can be found in the Interface Section.

Data Communication

In order to be able to send data from one terminal to another Philips has designed a number of data communication devices for the P850, P855, P860 and P880 computers.

The application of those devices together with one or more computers provides a variety of communication systems such as:

- data concentrators
- store and forward switching terminals controllers
- intelligent terminal station
- remote batch installation
- data collection
- time sharing

The possibilities of those devices are further increased by coupling a minicomputer to a P880 system via an adaptor which gives the following applications:

- a stand-alone time sharing system
- a real-time distribution system
- an invoicing and inventory control system

Any system accepts several transmission codes e.g. USASCII, ISO, EBCDIC etc. The data may be sent via the switched networks or the telex networks, depending upon the installation, in half-duplex or full duplex mode.

Control Units

Three control units for low or medium speed transfer of data via the networks have been developed to couple the minicomputer to the peripherals and to the networks or to a larger computer.

All three control units are based on the MOS-TTL technology and are built on cards measuring 20.8 × 31.0 centimeters.

MELCU (Multiple Low Speed Line Control Unit)

The MELCU provides the interface and control between the central processor and up to 8 full duplex asynchronous low speed lines. This control unit is always used together with a Line Terminating Unit which provides the actual line interface. Several types of LTU's have been designed in order to be capable of interfacing a variety of data communication lines.
The interface exchange with the central processor is performed through the programmed channel.
MELCU can be used for inplant connections as well as outplant connections to the telex networks in which case it can control an automatic answering unit or to the telephone networks via modems.
A clockcard provides the timing signals for line speeds of 50, 75, 100, 110, 150, 200 and 300 bits per second.
Other low speeds are available as an option.
As the MELCU is connected to one speed of the clockcard and the interface is only suited to connect to one LTU, the 8 lines of the LTU must be of the same type.
The transmission code may consists of 5, 6, 7 or 8 bits characters with odd, even or no parity check.
The required voltages are provided by the power supply P849-014.

ASYLCU Asynchronous Single Line Control Unit

The ASYLCU provides the interface and the control between the central processor and an asynchronous low or medium speed line up to 2400 bits per second.
The line interface is contained in the control unit and is suited for connection to modems operating in accordance to the CCITT-V23 recommendation (with automatic answering feature). The mode of transmission is either half-duplex or full duplex when two ASYLCU's are used.
In asynchronous mode the 5, 6, 7 or 8 bits characters are accompanied by start and stop bits which are generated by ASYLCU when data are transferred or stripped from the data stream when data are received. The parity check may be odd, even or no parity check.
The connection to the central processor is via the Programmed Channel or the Multiplex Channel.
The transmission speed and sampling rate for received data are generated by the same clockcards as described under MELCU.
The voltages for the ASYLCU and modem are supplied by the power supply P849-014.

SYLCU Synchronous Single Line Control Unit

The SYLCU provides the interface and control between the central processor and a synchronous medium or high speed (inplant) line. The permissible maximum line speeds are 9600 bits per second for dedicated or leased outplant lines and up to 50,000 bits per second for inplant connections.

The transmission code may consist of 6, 7 or 8 bit characters with odd, even or no vertical parity check. Longitudinal or Cyclic Redundancy Checks on the transmitted data are available as an option.

The transmission of data occurs according to the ECMA or BSC procedures in point-to-point or multipoint networks.

The transmitted characters are accompanied by control characters which will cause certain actions to be performed. Those control characters are detected by SYLCU. Other characters, which control the synchronization of the sending and receiving terminal are deleted from the incoming data stream. The synchronization characters are generated by the modems.

SYLCU is connected to either the Programmed Channel or to the Multiplex channel. Connections to the switched networks are according to CCITT-V23 recommendations.

Power supply P849-014 provides the required voltages.

PART 2 OPERATION

C.P.U. Control panel

This chapter contains details of the control panel and the mini-panel and their use during program loading and running. Operating instructions for peripheral units will be found in Chapter 3.

CONTROL PANEL LAYOUT

The control panel for the P855 and P860 are shown in Figure 1; the use of the mini-panel, shown in Figure 9, is given in a later section. The control panel contains switches and indicators to allow the operator or programmer to load, display, run and supervise a program.
The switches and indicators are:

CLEAR
A pushbutton to clear the control flip-flops of the P.S.W. (Program Status Word), the other bits are set to one (i.e. program level and condition register). The control unit flip-flops are reset to the inactive state.

RUN
A pushbutton to select normal operating mode.

INST
A pushbutton to select instruction by instruction mode. In this mode a single instruction is executed each time the START button is pushed.

READ
A pushbutton to allow the contents of either the registers or the memory locations to be displayed.

LOAD
A pushbutton to allow the data selected on the data switches to be loaded into either the registers or the memory locations.

INT
A pushbutton to generate an interrupt and allow the operator or programmer access to the running program.

START
A combined pushbutton and indicator; it is pushed to start the program or any instructions given from the panel: the button lights when the program is running.

M, ST, R
Three pushbuttons that select memory register, CPU status word or register respectively.

8, 4, 2, 1
Four pushbuttons to select the register to be displayed or loaded.

DATA SWITCHES
Sixteen toggle switches used to load data or to select a memory address.

DATA INDICATORS
Sixteen indicator lights numbered 0 to 15, used to display the contents of the registers, memory locations or CPU status word.

PRESET
When pushed this switch will stop the program at the address set up on the DATA SWITCHES.

KEY LOCK
A three-position switch: OFF, ON and LOCK; used to switch power on or off and to enable or disable the control panel switches.

OFF
In this position, all power to the central processor is switched off.

ON
Switches on power to the central processor and allows all the switches to function normally, except that when RUN mode is selected the CLEAR button does not function. No automatic restart after power failure.

LOCK
In this position all the pushbuttons are disabled except the INT button which functions normally; automatic restart will be performed, if necessary.

LOADING DATA

Data can be loaded into either registers or memory locations. The routines follow the flow chart in Figure 2 and 3.

Loading Register Routine
This routine is used to load both the P-register (register 0) and the general-purpose registers. When loading the P-register, it must be remembered that this register uses only 15 bits and DATA SWITCH 15 MUST NOT be used as part of the address but must be set to 0.

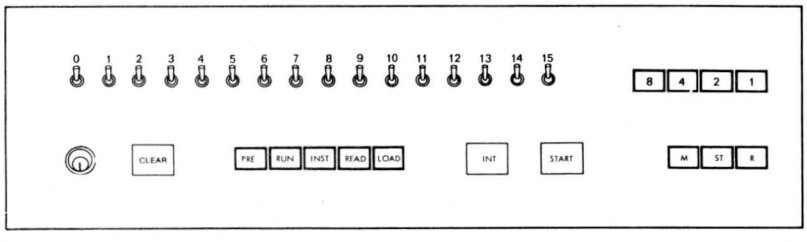


Figure 1 Control Panel

Step 1 Ensure that power is connected to the system and that it is switched on at the mains.

Step 2 Turn the key lock to the ON position and push the CLEAR button.

NOTE: If the loading is to be done during operation of the system, steps 1 and 2 should be omitted.

Step 3 Push the LOAD and R buttons on the panel.

Step 4 Push the 8, 4, 2, 1 buttons so that they indicate the address of the required register. When loading the P-register, this step should be omitted.

Step 5 Set the DATA SWITCHES to indicate the data to be loaded into the register.

Step 6 Push the START button and the data will be written into the register.

If more than one register is to be loaded, repeat steps 4, 5 and 6 as necessary.

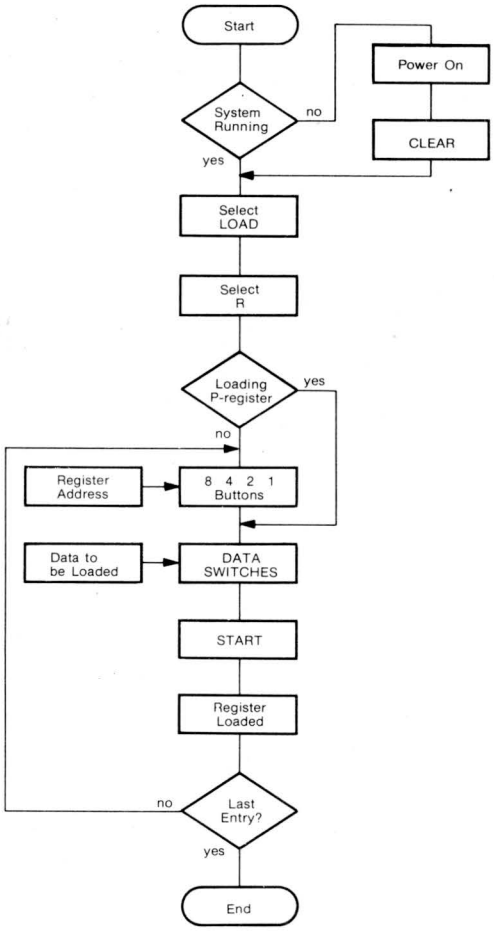


Figure 2 Loading data into registers

Loading Memory Routine

Step 1 Load the required memory address into the P-register using the loading register routine described above.

Step 2 Push the M button on the control panel.

Step 3 Set the DATA SWITCHES to indicate the data to be loaded into the memory.

Step 4 Push the START button; the data will be written into the memory and the P-register will be incremented automatically.

If successive memory locations are to be loaded, repeat steps 3 and 4 as necessary. To load memory locations which are not successive, repeat the whole memory loading sequence.

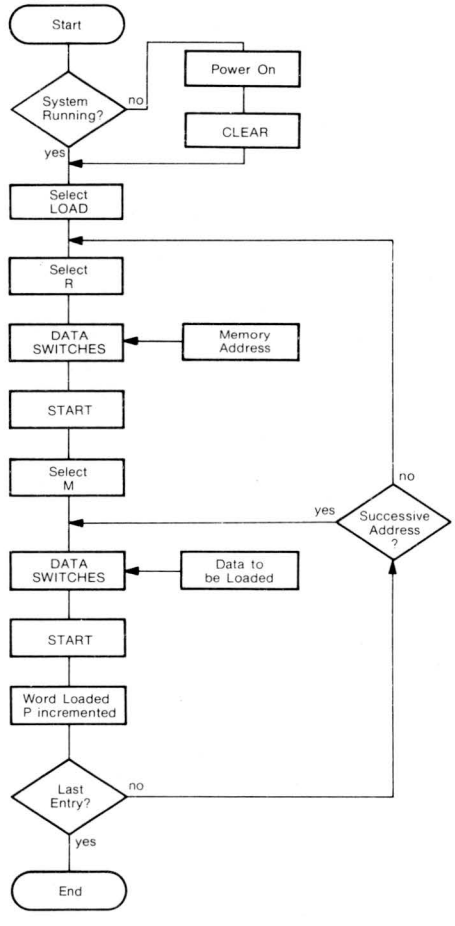


Figure 3 Loading data into memory

READING DATA

To read the contents of either the registers or the memory locations, the following routine should be used. This routine follows the sequence shown in the flowchart in Figures 4 and 5. If the program is already running, push the READ button to halt the program after executing the current instruction.

Reading Registers Routine

Step 1 Push the READ and R buttons.

Step 2 Push the 8, 4, 2, 1 buttons so that they indicate the address of the required register. When reading the P-register omit this step.

Step 3 Push the START button; the contents of the register will now be read out and will be displayed on the sixteen DATA INDICATORS.

If more than one register is to be read out, repeat steps 2 and 3 as necessary. When reading the contents of the P-register, it must be remembered that DATA INDICATOR 15 is NOT part of the address and should be ignored.

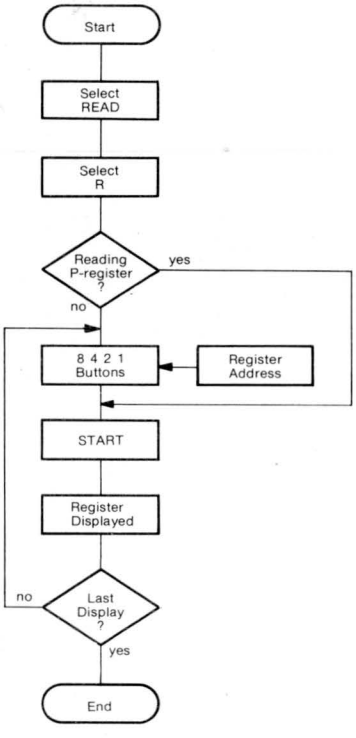


Figure 4 Reading data from registers

Reading Memory Routine

Step 1 Load the required memory address into the P-register using the loading register routine.

Step 2 Push the READ and M buttons.

Step 3 Push the START button; the contents of the memory location will be read out and will be displayed on the DATA INDICATORS; the P-register will be incremented automatically.

If successive memory locations are to be read out, repeat step 3 as necessary. When reading memory locations which are not successive, repeat the whole sequence.

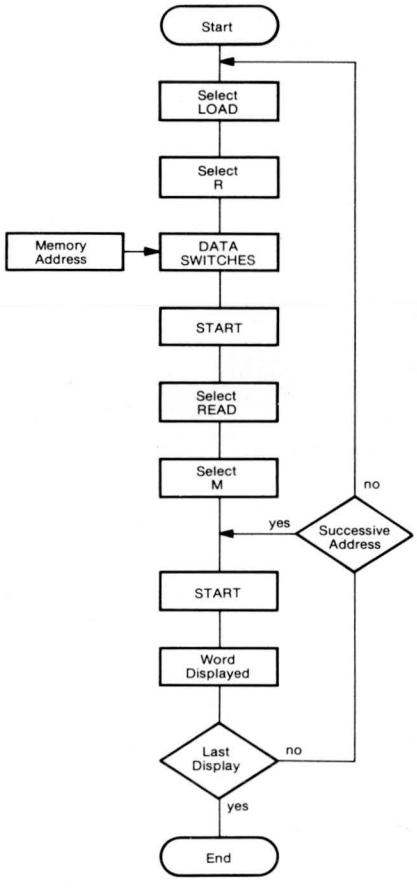


Figure 5 Reading data from memory

LOADING SYSTEM PROGRAMS

Before the system can be used, one of the system programs must be loaded into the memory. Which program is used to initiate the system will depend upon the type of system used but, in all cases, the system program will be loaded by the bootstrap.

The bootstrap loads only programs coded in binary format and before it can be used it must first be loaded into the memory (instruction by instruction) using the loading memory routine. The actual instructions used by the bootstrap will depend upon the peripheral device used to read in the program. However, in general it will follow the sequence shown in Figure 6.

Step 1 Ensure that power is connected to the system and that it is switched on at the mains.

Step 2 Load the first address of the bootstrap into the P-register, using the loading register routine.

Step 3 Load each instruction of the bootstrap into memory, using the loading memory routine.

Step 4 Check that the bootstrap has been loaded correctly by using the reading memory routines.

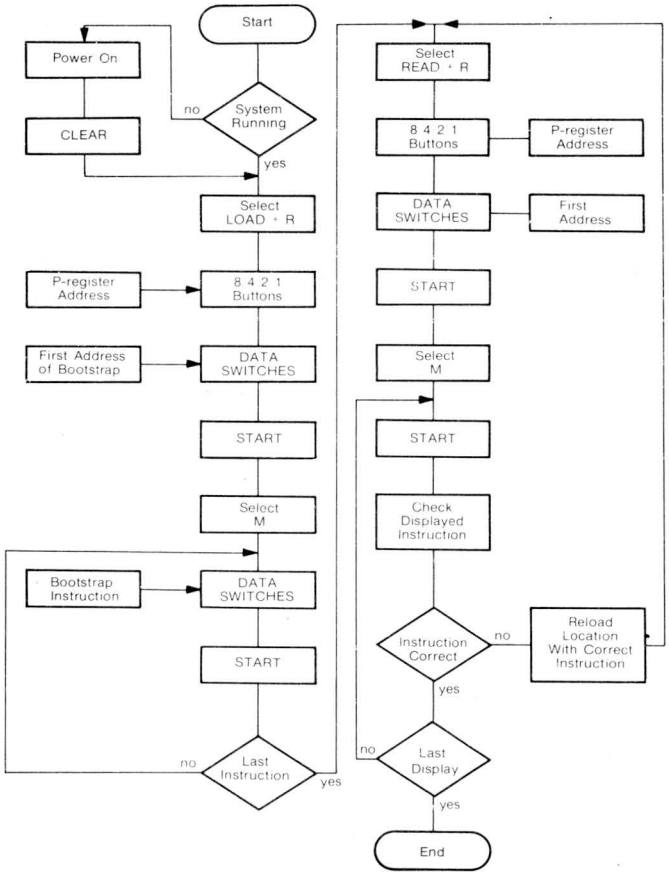


Figure 6 Loading the bootstrap

Once the bootstrap has been loaded, the system program medium (either punched tape, cards, magnetic tape or disc) can be loaded onto the appropriate peripheral and the sequence shown in Figure 7 used to load the program.

Step A Load the program medium onto the input peripheral, ensuring that power has been connected to it, that it is switched on at the mains and that it is under the control of the central processor.
NOTE: Instructions for loading and operating the peripherals are given in Chapter 2 of this part.

Step B Load the starting address of the bootstrap into the P-register, using the loading register routine.

Step C Push the RUN button.

Step D Push the START button; the indicator in the START button should light and punched tape reader should run and load the program into the memory.

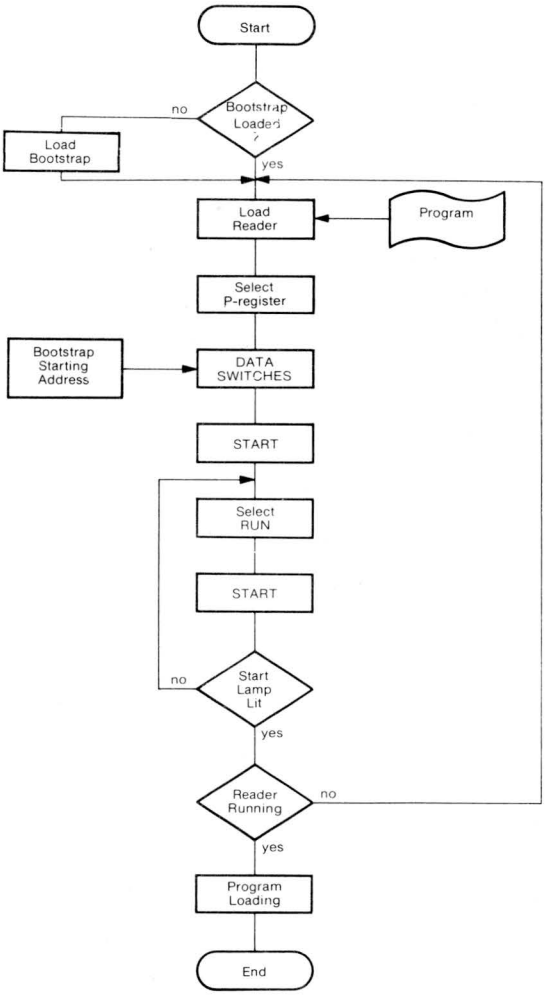


Figure 7 Loading the bootstrap and programs

FAILURE TO START

Step 1 The START indicator does not light. Check that power has been connected to the system and switched on. Check that the RUN button has been pushed.

Step 2 The punched tape reader does not run. Check that the reader START indicator is lit, that the reader has been switched on and is under the control of the central processor. Check that the reader has been loaded correctly. If the system has more than one reader, check that the correct reader has been loaded.

Step 3 Check that the bootstrap loader has been loaded correctly using the reading memory routine.

PROGRAM LOADER

The Program Loader is used to load programs written in relocatable format, but before it can be used it must first be loaded into memory by the Bootstrap Loader. This program consists of two parts, the actual loader and an operator communication part which allows the operator to select various modes of loading. Once the Program Loader starts to load a program, the operator communication part is overwritten.

There are seven control messages which the operator can give to the loader and their mnemonic and meaning are given below:

. CR LF	meaning no more initialization to be done
TR CR LF	meaning the next message will be read from the punched tape reader
CT CR LF	meaning construct a Chaining Table from
P1 ABCD CR LF	the programs listed below, where:
P2 ABCD CR LF	P1, P2,....Pn are the names of the programs which are to be included in the table, and:
⋮	
Pn ABCD CR LF	ABCD is the absolute address of the table. This absolute address contains the entry point address of the program.
L CR LF	meaning load the programs listed below, where:
P1 P2 ...Pn CR LF	P1 to Pn are the names of selected programs and:
NL CR LF	meaning do NOT load the programs listed below
P1 P2 ...Pn CR LF	

The programs named in these messages can be contained either on one tape or on several tapes. If the programs are all contained on one tape, each program is separated from the others by a :EOS (not end of segment) code. When several tapes are used, each tape must end with a :EOF (not end of file) code.

ST CR LF	meaning this is the starting address of the program

then

P XY CR LF	P signifies a program whose entry point is defined by XY

or

A ABCD CR LF	A signifies an address is to be given and ABCD is the address which the loader must enter once loading is completed
BA CR LF ABCD CR LF	This message allows the operator to define the base address for the loader. If this message is not given, the base address is assumed to be the one fixed at generation time.

Which of these above control messages are given will depend upon the programmer's instructions, but their operation is as follows:

Once the Program Loader has been loaded into memory it is started by pushing the START button on the control panel. It will then type out:

I :

on the operator's typewriter. If the control messages are to be input from the Punched Tape Reader, the control message tape should first be loaded in the Tape Reader, then the message:

TR CR LF

should be typed on the operator's typewriter. The loader will then read the message tape and await the loading of the program tape or tapes. When this has been done, the operator has only to push the START button to start the loader loading the required programs.

If the control messages are to be input from the operator's typewriter, the operator will type one or more of the following:

CT CR LF	followed by the parameters
L CR LF	followed by the parameters
NL CR LF	followed by the parameters
ST CR LF	followed by the parameters
BA CR LF	followed by the parameters

the last message will always be:

. CR LF

which tells the loader that initialization has been completed.

NOTE: When the L CR LF message is given, the loader itself decides when loading has been completed. When the NL CR LF message is given, the operator has to decide when loading has been completed.

After initialization has been completed, the operator loads the program tape on to the Punched Tape Reader, then the START button is pushed and the loader will start to load the program.

If more than one program tape is to read, the loader will halt when the end of tape is reached and the operator should then load a new tape and again push the START button.

When all the tapes have been loaded, the loader will start processing the program provided that the starting address has been given. If a starting address has not been given, the loader will halt until the operator gives a starting address.

If the operator tries to load too many programs into the available memory space, the loader will halt when the available memory space has been filled and will set the overflow condition bit.

CONTROL MESSAGES

The following control messages can be used when the Utilities and Dump program is being used. Before the message can be input the INT button must be pushed. This causes the following system message to be printed out:

M:

the operator can then type one of the following messages:

DR CR LF
meaning dump specified register contents (in hexadecimal) on the operator's typewriter

DH ADD1 ADD2 CR LF
meaning dump (print) the contents of memory from address 1 to address 2 on the operator's typewriter in hexadecimal.

DB ADD1 ADD2 CR LF
meaning binary dump of the contents of memory from address 1 to address 2 will be output by the tape punch.

WM ADD1 VAL1 VAL2 .....VALn CR LF
meaning write the values VAL1 to VALn in sequential addresses starting at address 1.

RUNNING THE SYSTEM

The following principles will generally apply.

Step 1 Load any data into the registers or the memory and load the data tape or program tape on to the appropriate peripheral.
Step 2 Check that the peripherals required by the program have been switched on and are under the control of the central processor.
Step 3 Load the starting address of the program into the P-register, using the loading register routine.
Step 4 Push the RUN button.
Step 5 Push the START button; the indicator in the START button will light and the program will run until one of the following conditions occurs:
A. A halt instruction is executed in the program.
B. The mode is changed from RUN to INST, READ or LOAD.
C. The INT button is pushed.

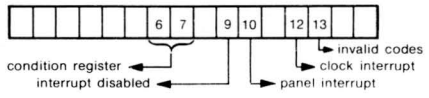
More details are supplied in the different sections of this book.

PROGRAM SUPERVISION

Programs can be supervised whilst running. The contents of the status word, registers and memory locations can be read out and checked using the INST mode. Any necessary changes to the program can then be made, using the loading register and loading memory routines, and the program can be restarted at any selected program instruction. In the INST mode the central processor executes one instruction each time the START button is pushed. The routine for using this mode is shown in Figure 8.

Step 1 Push the INST button.
Step 2 Load the address of the program instruction (at which you wish to start reading) into the P-register, using the loading register routine.
Step 3 Push the START button; the instruction (whose address was loaded into the P-register) will be executed and the P-register will be incremented.
Step 4 Push the READ and either the M, ST or R buttons (if the button R is selected, the 8, 4, 2, 1 buttons must be set to the address of the required register). If the M button is selected, the next instruction will be displayed; if the ST button is selected, the status word will be displayed, and if one of the fourteen general-purpose registers is selected, the data displayed will be the result of the executed instruction. When the P-register is selected, the data displayed are the address of the next instruction.

The format of the CPU status word is:



If successive memory locations are to be read out, repeat steps 3 and 4.
When the memory locations are not successive, repeat steps 2, 3 and 4.

To restart the program, the following three steps should be carried out.

A Load the address of the first instruction to be executed into the P-register, using the loading register routine.
B Push the RUN button.
C Push the START button; the indicator light in the START button will light and the program will run.

PRESET

Program supervision using the PRESET switch has the advantage that the program can be stopped at any preset address selected by the programmer or operator. The routine for using this switch is as follows:

Step 1 If the system is running push the INT button.
Step 2 Push the PRESET button.
Step 3 Set the DATA SWITCHES to indicate the address at which the program is to be stopped.

Step 4 Push the RUN and START buttons. The program will run until the program address equals the value set up on the DATA SWITCHES; when this occurs, the program will stop and the programmer or operator can use the INST, LOAD or READ modes as required.

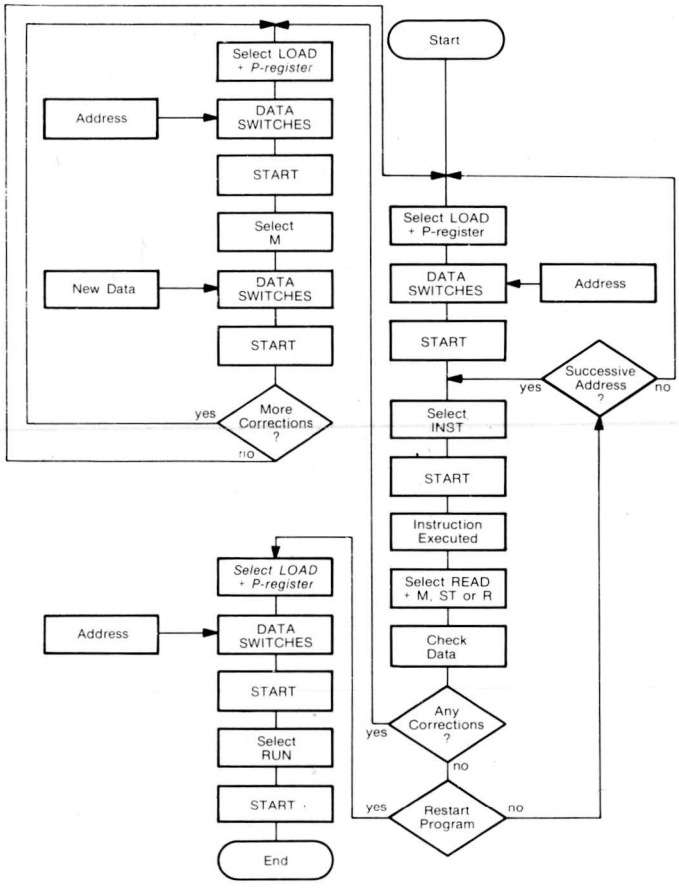


Figure 8 Using the INST mode

MINI-PANEL

Layout

The mini-panel shown in Figure 9 contains the following switches:

CLEAR
A push button to clear the control flip-flops of the P.S.W., the other bits are set to one (i.e. program level and condition register). The control unit flip-flops are reset to the inactive state.

INT
A pushbutton to generate an interrupt and allow the operator or programmer access to the running program.

START
A combined pushbutton and indicator; it is pushed to start the program and the button lights when the program is running.

KEYLOCK
A three-position switch, OFF, ON and LOCK. The OFF position switches off all power to the central processor and enables all the pushbuttons to function normally, and in the LOCK position the only button that functions is the INT button.

OPERATING PROCEDURE

With the mini-panel there are only two possible modes of operation; these are: RUN mode and IDLE mode.

Idle Mode

To start a program when the central processor is in idle mode, carry out the following procedure:

Step 1 Ensure that the power is connected to the system and that it has been switched on at the mains.

Step 2 Check that the CLEAR button is lit; if not, turn the keylock to the ON position.
Note: If the system has already been running, the above two steps can be omitted.

Step 3 Ensure that the keylock is in the ON position; if not, turn the key into the ON position.

Step 4 Push the START button. The indicator in the button will light and the program will start either at the address held in the P-register (e.g. after a halt instruction), or at address zero (if the CLEAR button has been pushed).

When the power failure/automatic restart option is available, the procedure to restart a program after a failure is as follows:

A Turn the keylock into the OFF position.

B Turn the keylock into the ON position and push the CLEAR button.

With this procedure, the automatic restart interrupt routine is executed.

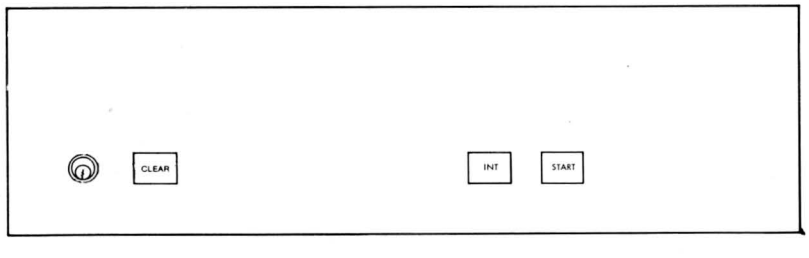


Figure 9 Mini-panel

Run Mode

Once the program has been started, the START button indicator will light and the processor will remain in the RUN mode until one of the following conditions occurs:

1 The INT button is pushed.
2 A HALT instruction is executed.
3 A power failure occurs.

When the INT button is pushed, the program is interrupted in the normal way. If the keylock is in the LOCK position, the INT button is the only button that has any action on the program.

When a HALT instruction or power failure occurs, the processor is switched from the run mode to the idle mode. However, this will not occur after a power failure if the automatic restart is available and the keylock is in the LOCK position.

OPERATOR'S TYPEWRITER

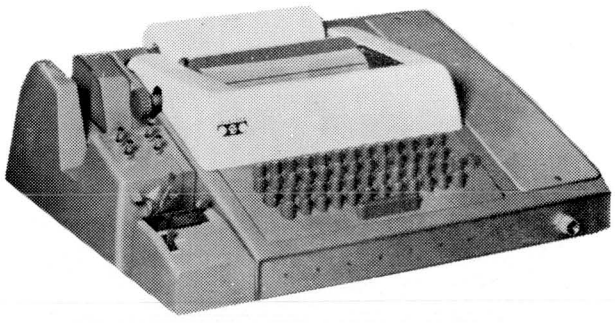


Figure 10 Operator's typewriter P841

The Teletype Operator's typewriter is used to give directives to and receive control messages from the system. The unit is operated as a normal typewriter, and therefore this section deals only with the paper supply, ribbon replacement and punched tape equipment.

Controls

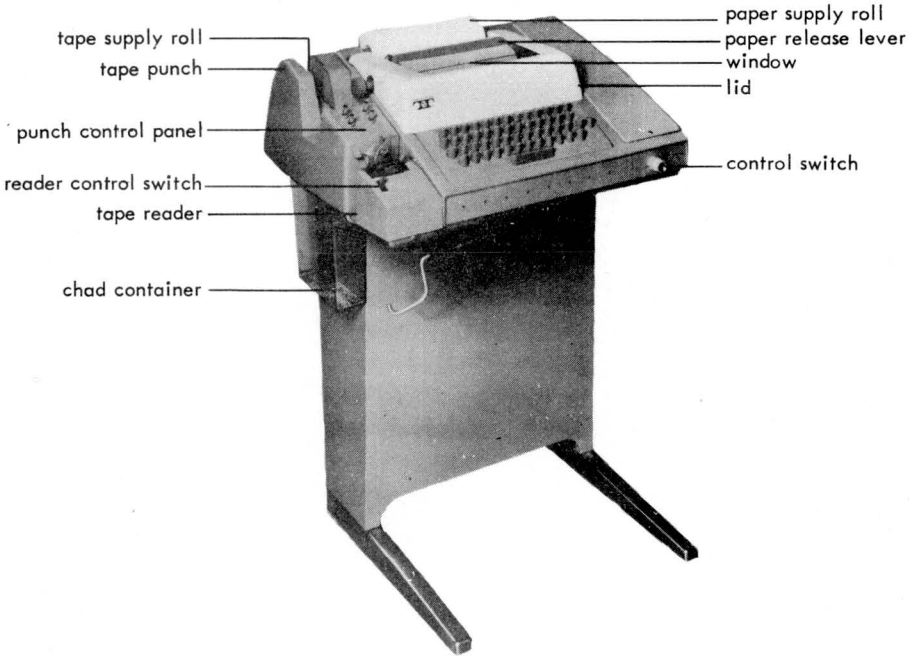


Figure 11 Typewriter Controls

LINE/OFF/LOCAL: A three-position switch on the front panel to control the operating mode of the typewriter. In the LINE position the typewriter input and output are connected to the central processor. In the OFF position the typewriter is switched off. In the LOCAL position the typewriter is operative but is not connected to the central processor.

Reader control: A four-position switch on the reader normally in the neutral position between START and STOP. When moved to the START position and released the reader will start to read tape. When moved to the STOP position and released the reader will stop. In the FREE position the reader is stopped and the tape is released and can be pulled through the reader by hand.

Punch controls

ON	A pushbutton to start the tape punch.
OFF	A pushbutton to stop the tape punch.
REL	A pushbutton that must be held down to release the tape and allow it to be threaded through the machine.
B. SP	A pushbutton to backspace the tape one code at a time.

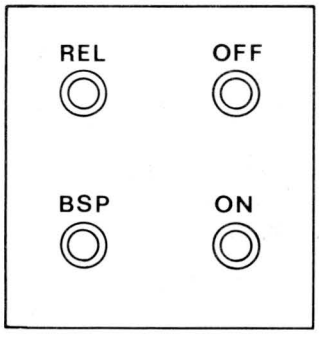


Figure 12 Tape punch control panel

Loading the paper

The typewriter uses a standard 12.5 cm diameter roll of 21 cm wide paper loaded on a spindle at the rear of the typewriter. The paper is then threaded forward under the platen. Use the following procedure to load a new roll of paper into the typewriter, as shown in Figure 13.

Step 1 Turn the typewriter control switch to OFF.

Step 2 Take the roll spindle from its slots in the typewriter and place it in a new roll of paper. An equal length of spindle should show at each end of the roll.

Step 3 Place the roll into the paper recess of the typewriter so that the ends of the spindle rest in the slots and the paper will unroll forwards from underneath the roll.

Step 4 Open the lid of the typewriter cover.

Step 5 Fold the leading edge of the paper back and crease it to give a smooth edge that is easy to thread under the platen and guides.

Step 6 Move the paper release lever forward to release the pressure roll.

Step 7 Take the paper up over the paper straightener and feed it down behind and under the platen, pushing the paper in as far as it will go.

Step 8 Move the paper release lever back and advance the paper, by turning the platen knob, until the paper can be passed under the paper guide in front of the platen.

Step 9 Use the paper release lever to straighten the paper (lay the paper back over the supply roll to check that the paper is straight).

Step 10 Close the lid of the typewriter cover.

Replacing the ribbon

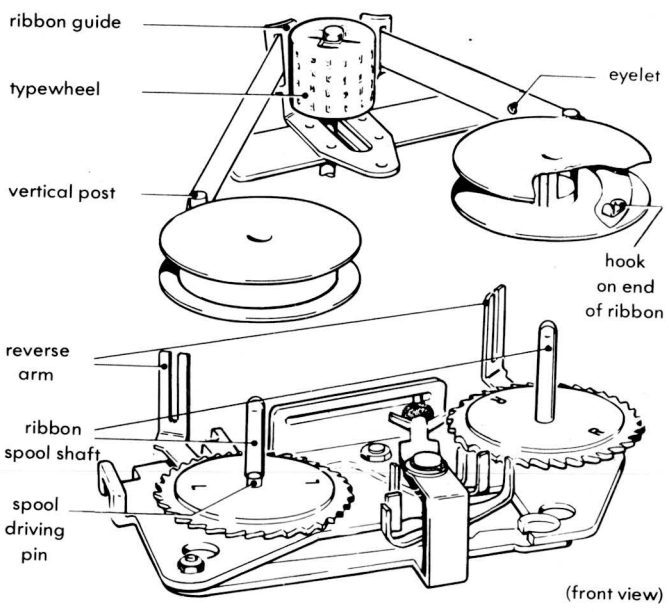


Figure 14 Ribbon Arrangement

The arrangement of the ribbon spools and the ribbon-threading path is shown in Figure 14. To replace the ribbon, first switch off the typewriter, then wind all the ribbon on to one spool (the spools should be free to turn in one direction). Remove the ribbon from the ribbon guide and lift out the spools. Unhook the end of the ribbon from the empty spool and discard the full spool.

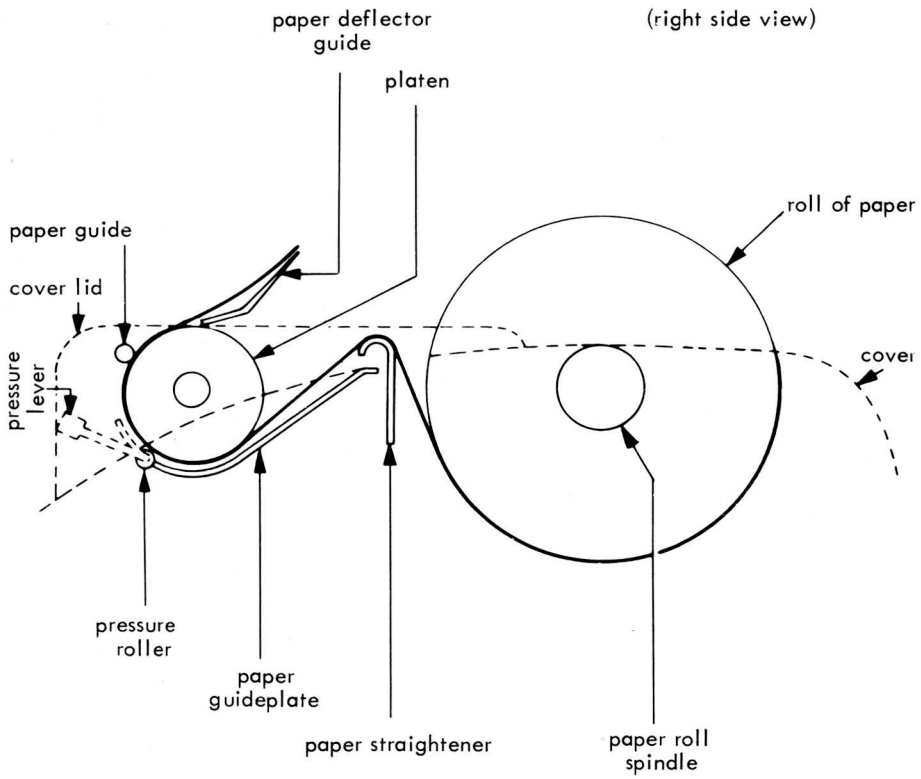


Figure 13 Paper threading path

Hook the end of the new ribbon on to the empty spool and place the spools on the spool shafts as shown in Figure 14. Thread the ribbon into the ribbon guides and between the typewheel and platen. Place the ribbon in the reversing arm guides as shown, making sure that the ribbon passes outside the vertical posts and that the reversing eyelet is between the reversing arm guide and the spool. Push the empty spool reversing arm towards the typewheel and wind a few turns of ribbon on to the empty spool, to check that the ribbon is running correctly through the guides and to take up any slack.

Operating the reader

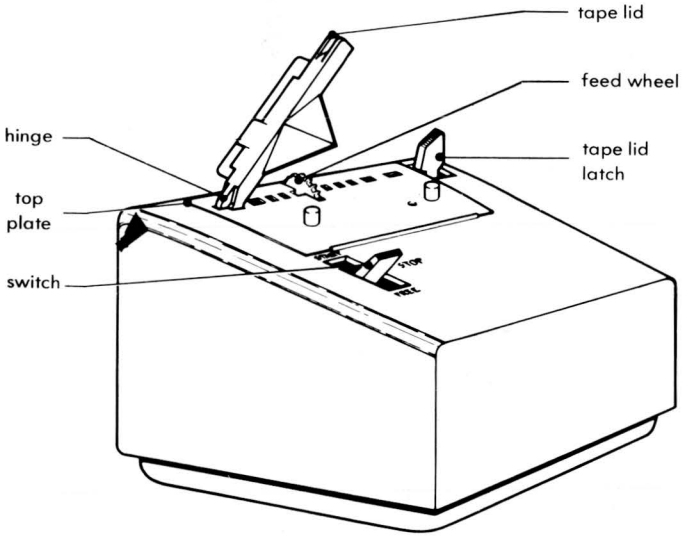


Figure 15 Tape reader

The reader accepts standard 1-inch tape punched in 8 channel ASCII code and types the coded information on the typewriter. When the unit is switched to LINE, this information is also sent to the central processor.

Step 1 Push the tape lid latch to the right to open the tape lid.
Step 2 Place the tape over the reader top plate between the tape guides so that the sprocket holes in the tape engage the feedwheel teeth.
Step 3 Hold the tape in position and close the tape lid, making sure that the latch engages.
Step 4 Push the reader control switch to START and release it. The reader will start and will run until either the end of the tape is reached, or you press the control switch to STOP.
Step 5 After the reader has stopped, the tape can be removed by pressing the reader control switch to FREE and pulling the tape through the reader.

Operating the tape punch

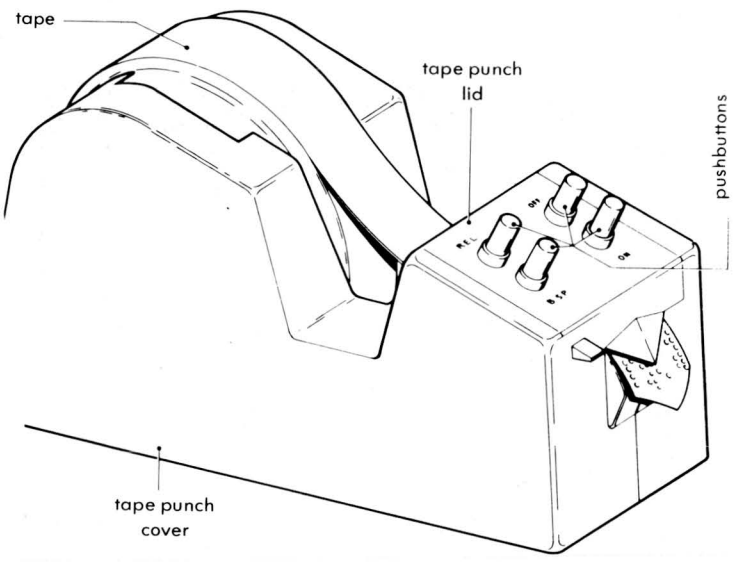


Figure 16 Tape punch (left front view)

The tape punch uses standard 1-inch tape and punches 8 channel ASCII code from the typewriter. Input can be via the typewriter keyboard or, if the typewriter control switch is in the LINE position, from the central processor.

Step 1 Lift the tape supply roll spindle out of the tape supply holder and place it in the centre of a new roll of tape.
Step 2 Drop the tape roll into the supply holder so that the tape feeds forward from the top of the roll.
Step 3 Hold down the REL button on the punch control panel and feed the end of the tape through the punch as shown in Figure 16.
Step 4 Turn the typewriter control switch to LOCAL.
Step 5 Push the punch ON button.
Step 6 Hold down the 'Here is' key on the typewriter keyboard to feed a few inches of the tape leader (punched with sprocket holes only) through the punch.
Step 7 Type the information you wish to have punched into the tape. If this information is to go to the central processor, as well as being punched into the tape, the typewriter control switch must be turned to LINE.

Error correction

Tape errors can be corrected by backspacing the tape into the punch, with the B.SP button on the punch control panel, and overpunching the error with the RUB OUT key on the typewriter.

Operator's maintenance

The operator's typewriter should be cleaned regularly to remove paper lint and dust. A soft, dry brush should be used to clean the working parts of the ribbon carrier, type mechanism, reader and punch. A soft cloth should be used to clean the cover. The chad container, under the punch, should be emptied regularly and should not be allowed to overfill. The type cylinder may be cleaned with one of the soft rubber type cleaners used for normal typewriters. In addition to the operator's maintenance, it is essential that the unit be serviced by a service engineer at 500-hour or 6-month intervals.

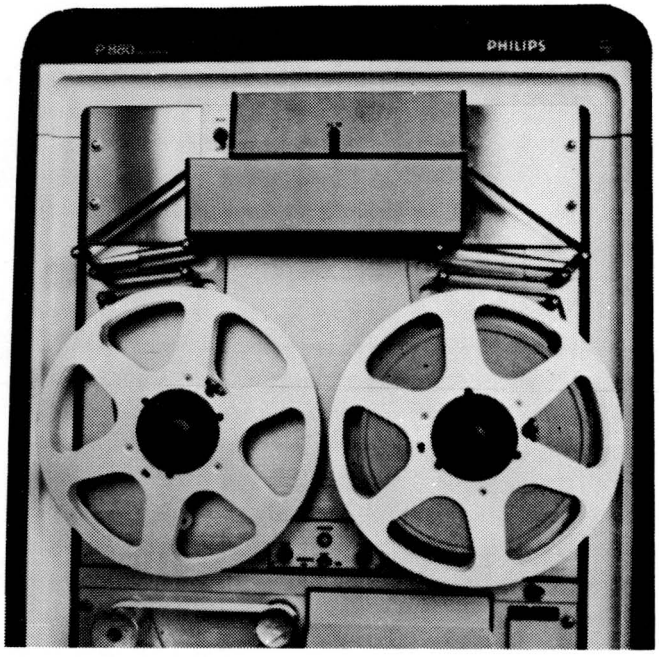


Figure 17 P801-001 Punched tape reader and tape handler

The P801-001 Digitronics punched tape reader has a speed of 333 characters per second and uses photo-electric reading. The reader is used in conjunction with a servo-operated tape handler, which winds the tape at a constant tension. Once the tape has been loaded on to the handler and threaded through the reader, operation of the reader is fully automatic under the control of the central processor; the handler follows the reader. Details of loading procedures and operating the controls are described below.

Controls and indicators

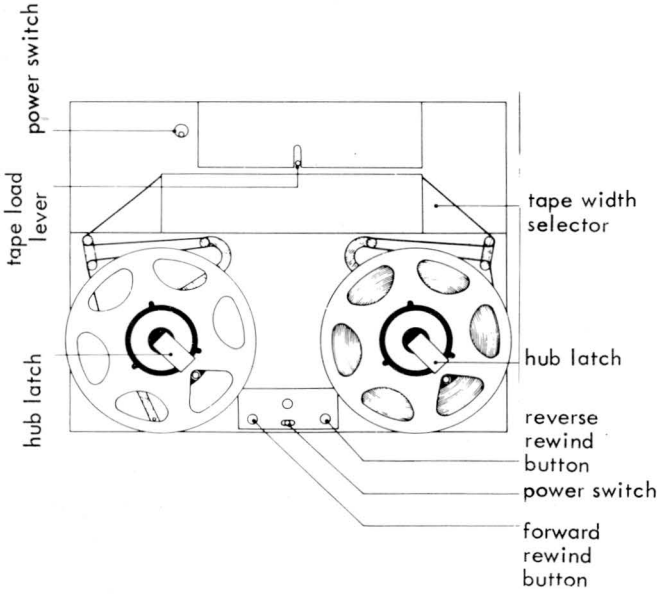


Figure 18 Operating controls

Reader

POWER: A three-position toggle switch (OFF, LOAD, RUN). In the LOAD position, the tape drive motorruns but the brake and the pinch roller are disengaged. In the RUN position, the motor runs and the pinch roller and brake are operated, under control of the central processor, to drive and stop the tape.

TAPE LOAD LEVER: A two-position lever to lift the front tape guide while the tape is being loaded.

TAPE WIDTH SELECTOR: A three-position control to select 6- or 7-(7/8 inch) or 8-(1 inch) channel tape width.
This control has a locking feature that prevents accidental moving of the selected channel width.

Handler

POWER INDICATOR: Lights when the unit is switched ON; the POWER switch may be in the ON position or in the REMOTE position with the remote control switch on.

POWER: A three-position toggle switch (OFF, ON, REMOTE). In the ON position the handler is switched ON and will wind the tape as needed by the reader, or under control of the rewind pushbuttons. In the REMOTE position the handler is under external control (this switch position is not normally used).

FORWARD REWIND: A pushbutton switch which is pushed to start the tape winding forwards at high speed, or stop the high-speed forward wind.

Loading the tape

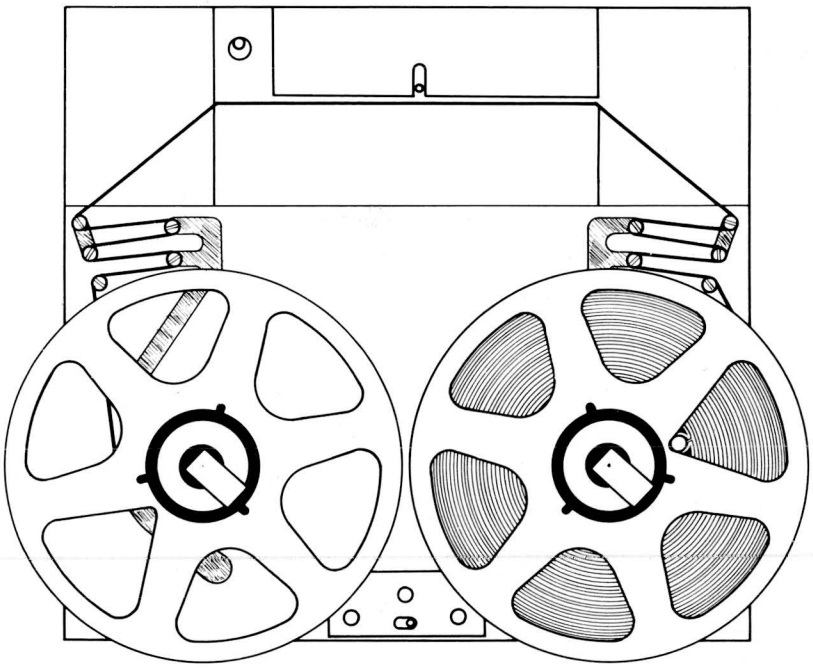


Figure 19 Tape path

Step 1 Switch the tape handler POWER switch to OFF.

Step 2 Switch the tape reader POWER switch to LOAD.

Step 3 Pull the reel hub latches forward to the fully open position.

Step 4 Fit the reel of tape to be read on the righthand (supply reel) hub and an empty reel on the lefthand (take-up reel) hub and close the reel hub latches.

Step 5 Pull a few feet of tape leader of the supply reel.

Step 6 Move the supply reel tape tension arm to the right as far as it will go and place the tape between the fixed and moving guides as shown in Figure 20.

Step 7 While holding the tape, as shown in Figure 20, allow the tension arm to return to its original position. (More tape will be pulled of the reel to form tape loops between the fixed and moving guides).

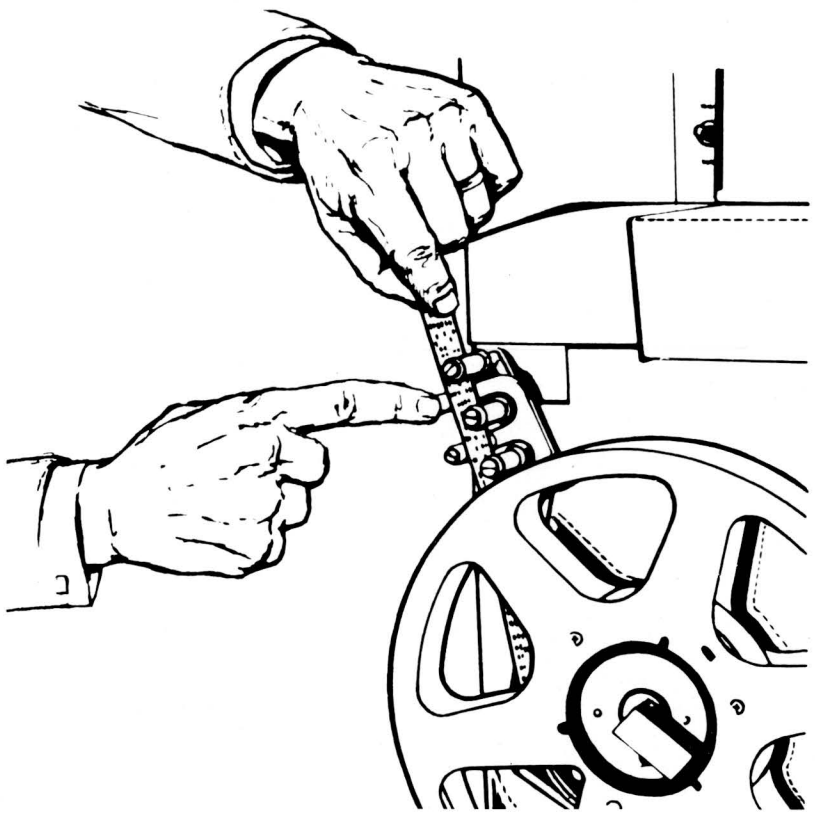


Figure 20
Placing tape between tension arm rollers and panel-mounted rollers

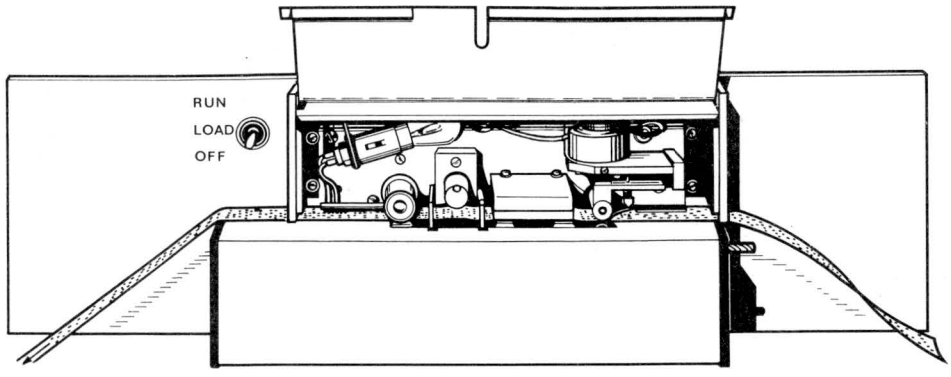


Figure 21 Reader tape path

Step 8 Lift the TAPE LOADER LEVER to the LOAD position.

Step 9 Place the tape in the reader. Although Figure 21 shows the tape path inside the reader, it is not normally necessary to open the reader cover because the tape can be slid in from the front of the reader.

Step 10 Lower the TAPE LOAD LEVER.

Step 11 Pull the take-up reel tension arm to the left as far as it will go and place the tape between the fixed and moving guides, as shown in Figure 20, before allowing the tension arm to return to its original position.

Step 12 Insert the end of the tape into the slot in the take-up reel and turn the reel counterclockwise to take up any slack in the tape.

Step 13 Switch the tape handler POWER switch to ON and the reader POWER switch to RUN. The unit is now ready to read the tape under the control of the central processor.

Rewinding the tape

Step 1 Switch the tape reader POWER switch to LOAD to release the tape brake.

Step 2 Press the REVERSE REWIND button on the tape handler. The tape will rewind on to the supply (right-hand) reel.

Step 3 Press the FORWARD REWIND button on the tape handler. The tape rewind will stop.

Highspeed forward rewind

Step 1 Switch the reader POWER switch to LOAD to release the tape brake.

Step 2 Press the FORWARD REWIND button on the tape handler. The tape will wind forward on to the take-up (lefthand) reel.

Step 3 Press the REVERSE REWIND button on the tape handler. The tape rewind will stop.

Operator's maintenance

The operator should make sure that the units are kept clean and free of paper lint and dust. A small, dry brush may be used to clean the interior of the reader head, the tape guides and the tension arms. Routine lubrication and cleaning should be carried out regularly by a service engineer; intervals between service should not exceed 10,000,000 start/stop operations.

TAPE PUNCH (150 ch/sec.)

The P804-001 FACIT tape punch uses standard 8 channel (1 inch) tape and operates at speeds of up to 150 characters per second.

A tape winder for winding the output from the punch is also incorporated in the punch unit.

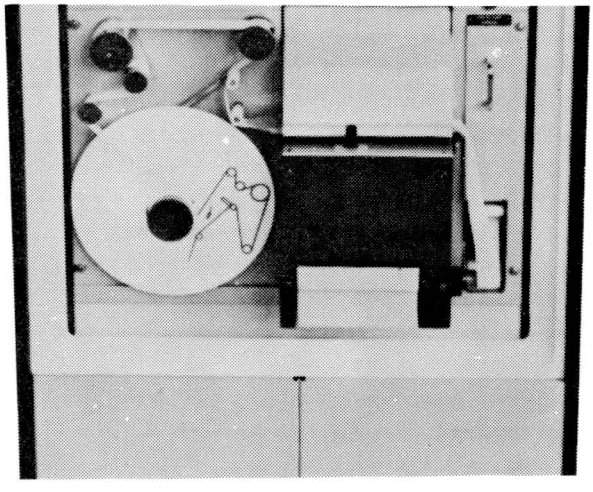


Figure 22 Tape Punch 150 ch/sec.

Loading the tape

Step 1 Pull magazine forward

Step 2 Place a new reel. The beginning of the tape is placed towards the rear of the machine.

Step 3 Pull the tape between the reel and the brake arm roller.

Step 4 Close the magazine.

Step 5 Pull tape under the power feel roller.

Step 6 Twist the tape 90 degrees in an anti-clockwise direction and pass it over the idler roller.

Step 7 Raise feed rocker and pass the tape between the feed arm and tape locking plate. Lower the rocker.

Step 8 Push toggle switch and the machine will start to run and feed the tape.

Step 9 Pass the tape under the rollers (4) and (5), around spigots (6), (7) and (8), over (9) and loop it around the take up spool.

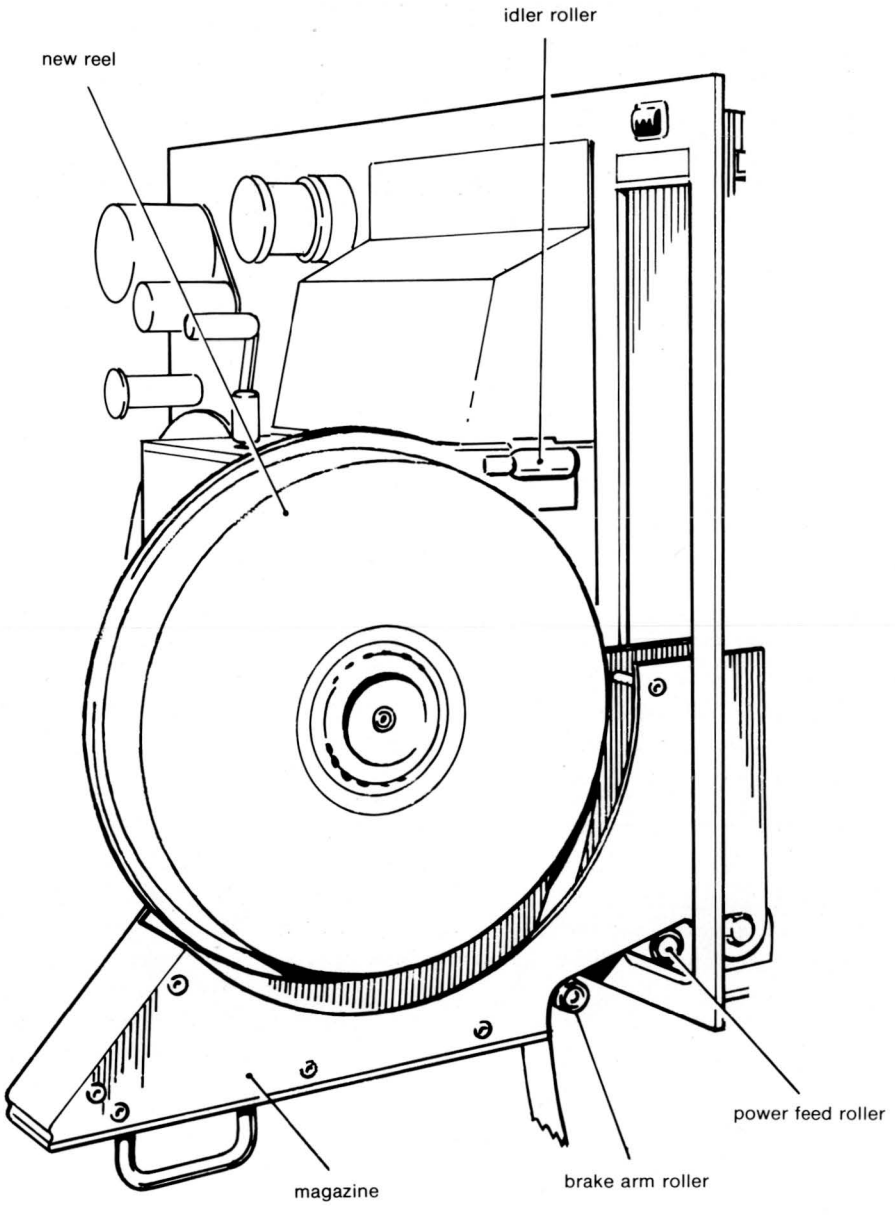


Figure 23 Loading the tape

Tearing of and removing punched tape

Step 1 Feed about 100 mm (4 inches) of blank tape through the machine using the toggle switch.

Step 2 Tear off the tape by pulling it straight down. Do not pull the tape forward as the tape may break, which will jam the punch mechanism.

Step 3 Reel up the slack tape and remove the spool, first pulling off the hub retainer.

Unloading machine at end of tape

Step 1 Cut tape between the power feel roller and idler.

Step 2 Use toggle switch to remove the remaining tape.

Step 3 When the tape is free from the feed rocker approximately 26 mm (1 inch) will still be under guillotine. To free this keep machine running with toggle switch and slowly pull remaining tape from the front of the punch.

Step 4 To remove remaining tape pull magazine forward and remove core and tape remnant.

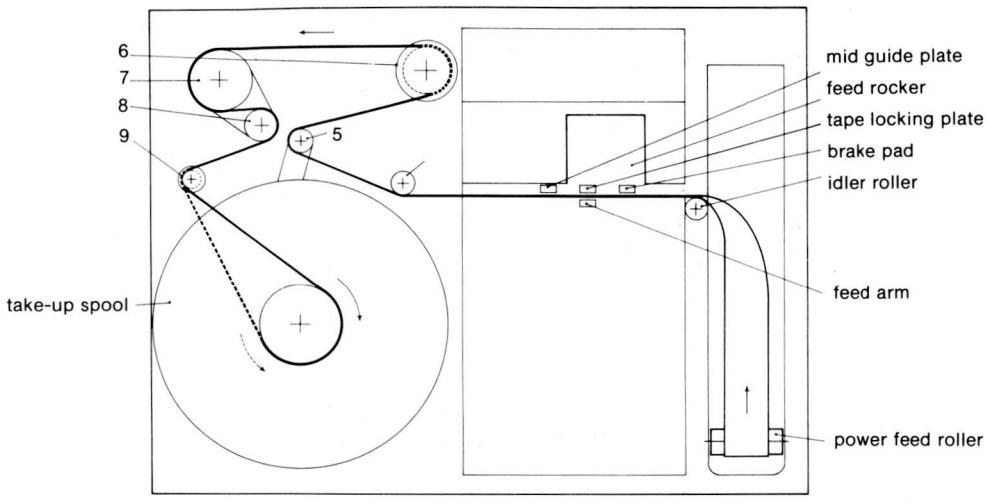


Figure 24 Tape path

TAPE PUNCH (75 ch/sec)

The P803-001 FACIT tape punch uses standard 8 channel (1 inch) tape and operates at speeds up to 75 characters per second. A control panel, containing the operator controls and indicator lights, is situated on top of the punch. Other controls not normally used by the operator, for clockwise or counterclockwise tape wind and for supply voltage selection, are located under the plastic cover. When the tape has been loaded with a roll of tape and switched on, it operates under control of the central processor without attention from the operator. The information required to load the tape into the punch and to remove rolls of punched tape is given below.

Controls and indicators

Operating controls and indicators are located on the top of the punch, and switches for the take-up motor are on each side of the motor. Figure 26 shows the location of the controls and indicators.

POWER ON	Switches on the mains supply to the punch.
D.C. ON	Switches on the internal d.c. supply.
READY	Lights green when the d.c. power is on.

Note: If the mains voltage falls below the level needed for reliable operation, the d.c. supply is switched off automatically and must be switched on again by the operator (D.C. ON switch) when full power has been restored.

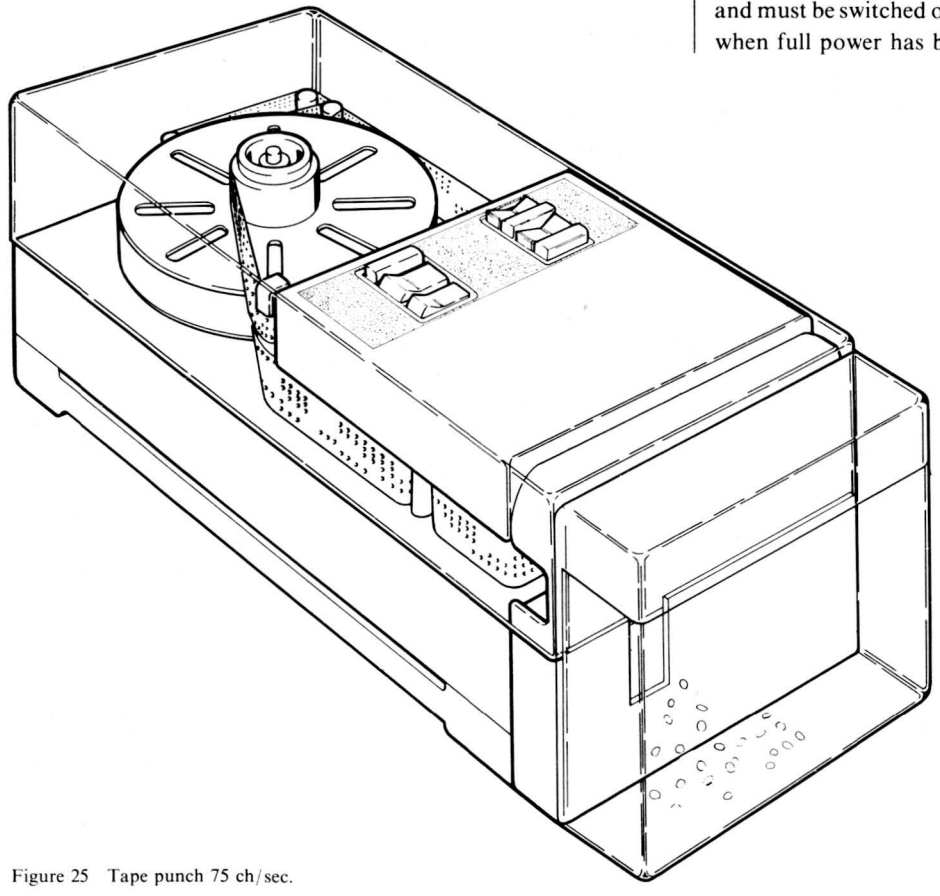


Figure 25 Tape punch 75 ch/sec.

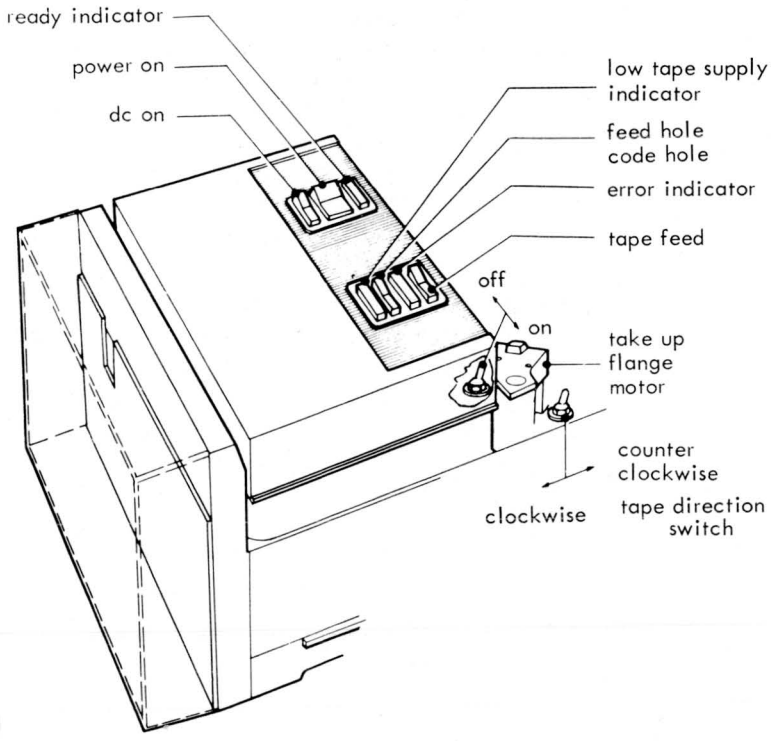


Figure 26 Controls

TAPE FEED: Feeds the tape from the supply to the take-up rolls without punching.

FEED HOLES/CODE: When pushed on the left, punches feed holes.
When pushed on the right, punches delete code (holes in all tracks).

ERROR: Lights if the tape breaks or is too tight.

TAPE SUPPLY: Lights when only a few feet of tape are left on the supply roll.

TAKE-UP MOTOR ON/OFF: Switches on the take-up motor.

TAKE-UP MOTOR DIRECTION: Controls the winding direction of the take-up motor.

Loading the tape

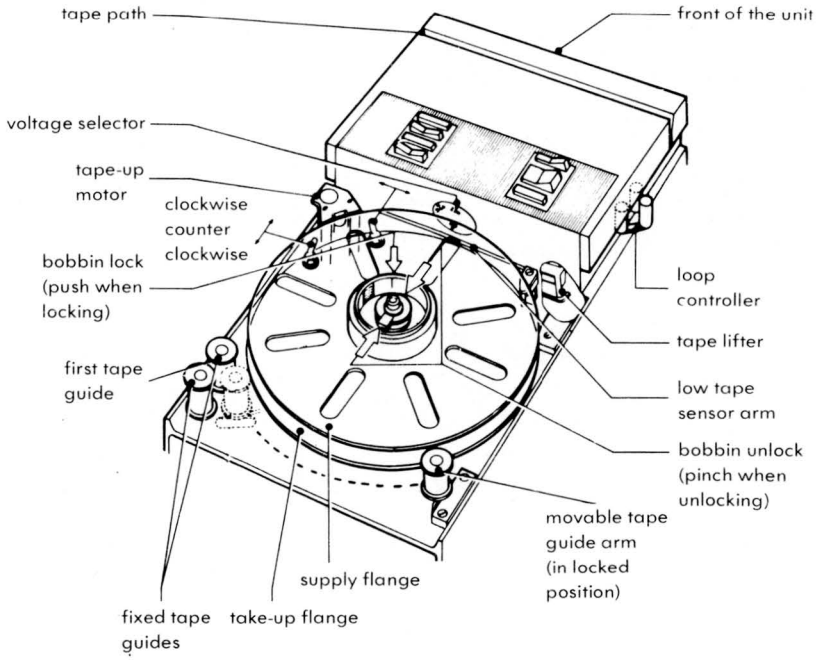


Figure 27 Tape supply and take-up deck

Step 1 Take of the plastic cover.

Step 2 Swing the tape guide arm to the lock position. This will release the low tape sensor arm, the loop controller and the feed mechanism.

Step 3 Move the take-up motor aside and lift off the take-up flange.

Step 4 Place a roll of tape on the supply flange so that the tape will unwind clockwise. Lock the bobbin on the flange by pushing the bobbin lock down. To remove a bobbin, unlock the bobbin lock by squeezing the lock release towards the shaft.

Step 5 Lead the tape around the first fixed guide in the righthand corner as shown in Figure 28, then around the moveable guide and back around the second fixed guide. Drop the tape into the tape path and inside the loop controller.

Step 6 Move the take-up motor aside and fit the take-up flange into place over the supply roll. Place a bobbin on the take-up flange and lock it by pushing down on the bobbin lock. Wind the tape one turn clockwise round the tape lifter and attach the tape to the bobbin. This is shown in Figure 29, where the end of the tape has been folded across the tape at 90 degrees leaving approximately 5 mm of tape protruding. Insert the tape end into the slot in the take-up flange and wind the bobbin a couple of turns.

Step 7 Check that the take-up motor is switched on and the tape direction switch is in the clockwise position. NOTE: If couterclockwise winding is required, it will be necessary to start the tape on the bobbin in the opposite direction to that shown in Figure 25 and the switch must then be in the counterclockwise position.

Step 8 Switch on POWER and D.C. and check that both the green READY indicator and the red ERROR indicator are lit.

Step 9 Swing the movable tape guide arm to the right until the ERROR indicator goes out. Hold the arm in this position and push the TAPE FEED switch. Release the tape guide arm when the tape becomes taut. Check that the ERROR indicator is out.

Step 10 Replace the plastic cover and the chad container. Check that the punch is working by punching feed holes or code. The punch is now ready to operate under the control of the central processor.

Removing the punched tape

Step 1 Remove the plastic cover.

Step 2 Push the TAPE FEED switch to feed all the tape that has been punched on to the take-up bobbin.

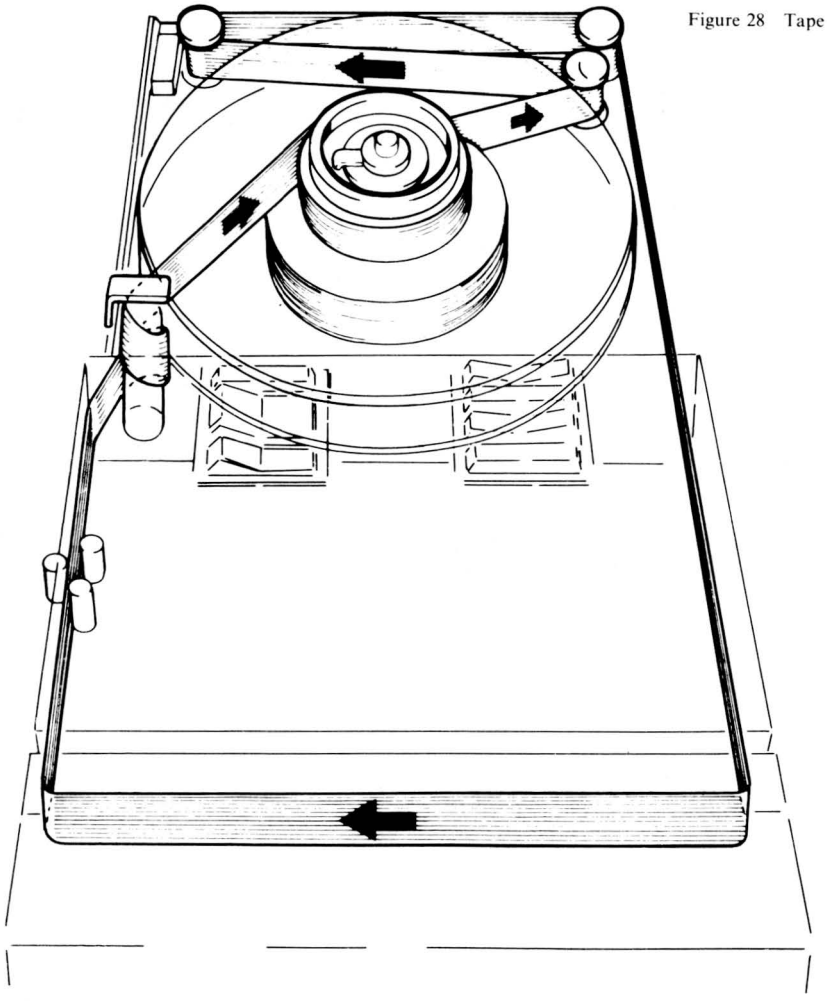
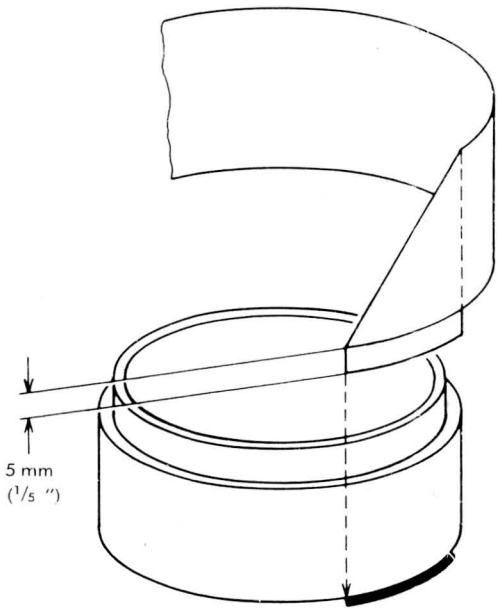


Figure 28 Tape path

Figure 29 Fastening the tape to the bobbin



Step 3 Turn the roll of punched tape back, by hand, to give some slack in the tape.

Step 4 Swing the movable tape guide to the lock position.

Step 5 Tear the tape on the tape tearer.

Step 6 Squeeze the bobbin lock release towards the shaft and remove the bobbin and punched tape.

Step 7 Lock a new bobbin into place and attach the tape end by folding as shown in Figure 29. Swing the moveable tape guide over to the right until the ERROR indicator light goes out.

Step 8 Push the TAPE FEED switch to feed a few turns of the tape on to the bobbin and take up the slack in the tape. Release the tape guide arm. Check that the ERROR indicator is out.

Step 9 Replace the plastic cover and check that the punch is working by punching feed holes or code. The punch is now ready to operate under the control of the central processor.

Operator's maintenance

The tape punch should be kept clean and free of paper dust and chad. The chad box should be emptied regularly and should not be allowed to become completely filled. If a malfunction does occur, the punch should be checked for torn or creased tape and for chad build-up before calling the service engineer. Routine lubrication and cleaning should also be carried out regularly by a service engineer.

CARD READER

The P806-001 card reader (SIEMAG P115) reads 250, 80 column cards per minute. The operation is under computer control and the operator's attention is only required for loading the hopper, emptying the stacker and clearing the defective cards.

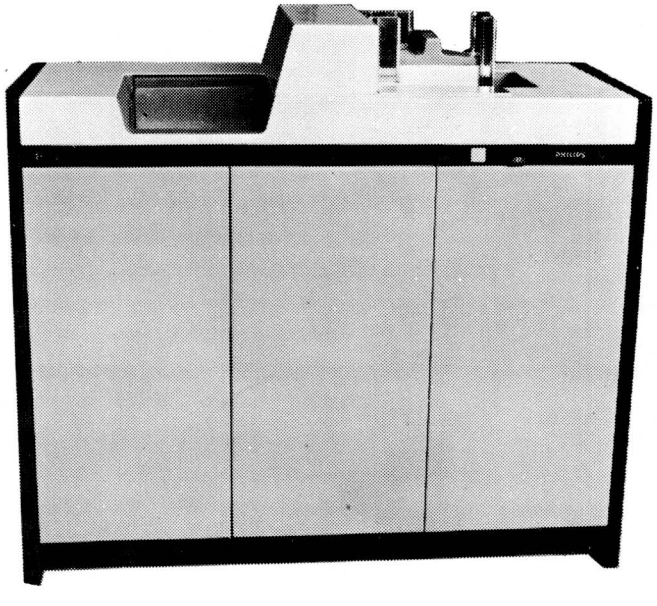
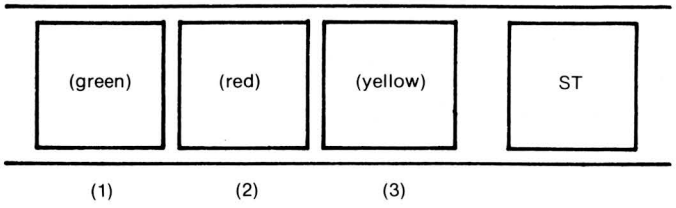


Figure 30 P806-001 Card Reader (250 cpm.)

Controls and indicators

Three control indicators and one pushbutton are mounted on the control panel.



Indicator

(1): This indicator lights green when the power supply on the P806-001 is switched on.

(2): Lights red when a read or transport fault condition occurs. Reread always the last card arrived in the stacker, after having cleared this condition. Indicator 2 and 3 light simultaneously.

(3): Lights yellow when the hopper is empty or the stacker is full.

ST-button:

This pushbutton has to be pressed when a fault condition has occurred. By pressing this button all electronic indication elements will be reset. It has also to be pushed after every malfunction.

Hopper and stacker

The card hopper may contain 700 cards. It is equipped with a card arranger to level the cards correctly.

The card stacker is spring-mounted to ensure constant height of fall of the cards. A micro-switch in the stacker is operated when the stacker is full.

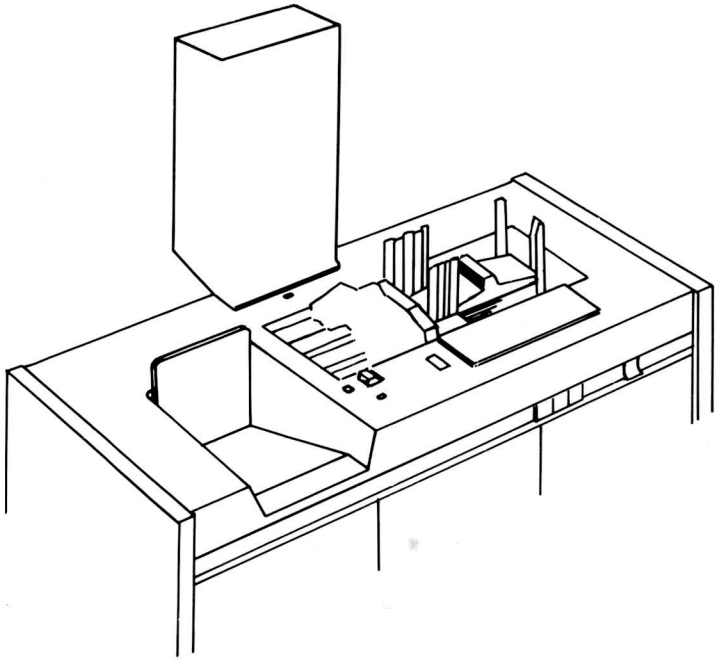


Figure 31 Top-view hopper and stacker

Loading the reader

The hopper may contain about 700 cards. A card weight ensures the right pressure when it is placed on the card deck before reading.

The bottom card of the deck is picked up by the pick-up shoe, then read and pushed into the stacker.

Place the card deck in the hopper with the first card down, column 1 to the left.

Step 1 Check that the green light is lit.
Step 2 Load the hopper with a card deck (700 cards max.). Level the card edges with the card arranger.
Step 3 Place the card weight on the cards.

Card jam

Even with careful handling cards may sometimes become jammed in the reader. Damaged cards should be carefully removed and duplicated. It is advised to check the cards on curled edges before they are placed into the hopper.

Step 1 Lift the spring cover with aid of a screw driver by putting it through the hole in the cover and pushing away the plate spring.
Step 2 Push the toggle switch to stop the running motor.
Step 3 Remove the jammed cards from the card track.
Step 4 Close the cover.

Card handling and storage

Card Handling

Care should always be exercised when handling cards to prevent them from being bent, torn or otherwise damaged. If such cards are fed into the reader they may be read incorrectly or become jammed in the mechanism. Cards which have become damaged should be duplicated.

Storage

Cards which have been punched, but which are not in use, should be stored vertically in the special trays provided in the computer room. The purpose of these trays is to prevent the cards from warping. Cards should not be stored directly on a floor or near heaters. Draughts and sudden cooling should also be avoided. Where a difference in either temperature or relative humidity exists between storage and computer room, cards transferred to the computer room must be allowed time to acclimatize.

LINE PRINTER

The P811-001 line printer (Data Products 2410) gives a printout on standard fanfold paper. The line printer operates at a speed of 245 lines per minute. The printer operates under computer control. A control panel for manual control, with indicators and toggle switches, allows to choose the operating mode and to control paper movement.

Figure 32 P811-001 Line Printer (245 l.p.m.)

DRUM GATE: Lights when drum gate is unlatched.

PAPER FAULT: Lights when paper is torn or missing.

PRINT INHIBIT (indicator): Lights when PRINT INHIBIT switch is in 'on' position.

MAIN POWER CIRCUIT BREAKER: Applies a.c. power to printer.

PRINT INHIBIT (switch): Inhibits hammer drivers during maintenance.

MASTER CLEAR: Initializes the printer to ensure that logic elements are in proper state.

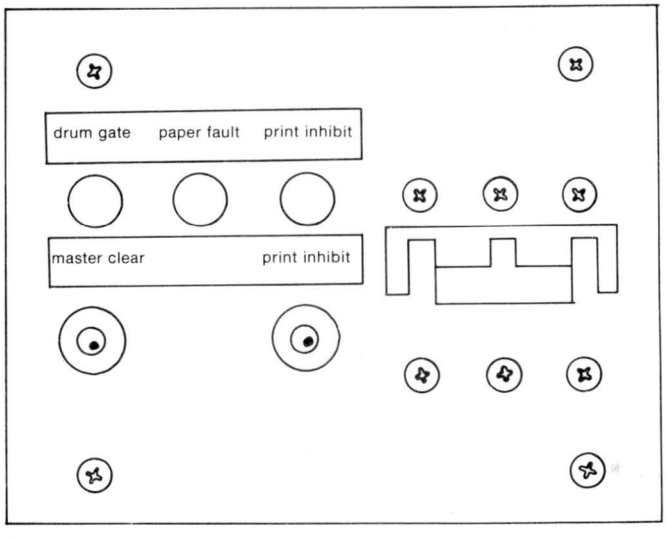


Figure 33 Maintenance panel

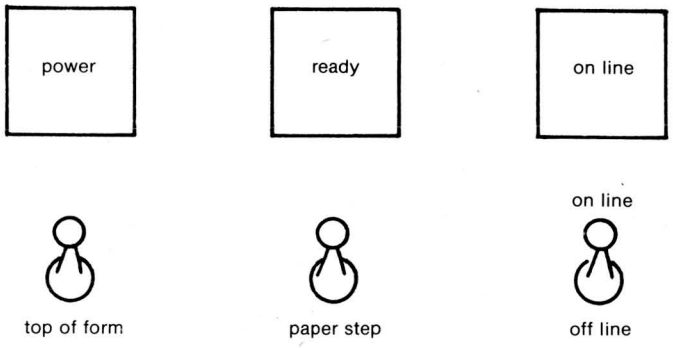


Figure 34 Control panel

Control panel (see Figure 34)
The control panel is located on top of the printer cabinet and has three indicators and three toggle switches.

POWER: The indicator lights when ac power is applied to the printer.

READY: Lights when printer is ready to receive information from the computer.

ON LINE: Lights when printer is in ON LINE mode.

TOP OF FORM:When this switch is activated it advances the paper to the top of form position where the first line is printed.

PAPER STEP: When this switch is activated it advances the paper one line. It is disabled when the printer is in ON LINE mode.

ON LINE/OFF LINE: This switch selects the mode of operation for the printer.

Operating procedure
Before the line printer can be used it must be loaded with fanfolded paper and ribbon and with a punched tape control loop. The paper tension and print pressure may also require adjustment when the kind of paper form or the number of copies is changed.

Step 1 Check that the paper, printing ribbon and punched tape loop are loaded.
Step 2 Check that the pressure control is adjusted for the correct number of copies.
Step 3 Adjust the horizontal and vertical paper positions.
Step 4 Wait for READY indicator to light.
Step 5 Ensure PRINT INHIBIT switch is down and PRINT INHIBIT indicator is off.
Step 6 Set ON LINE/OFF LINE switch to ON LINE position and release; ensure ON LINE indicator lights.
Step 7 When printing starts check printer operation and make paper adjustments, if necessary.

Loading the paper

Step 1 Lift maintenance panel cover; set circuit breaker (4) to OFF.
Step 2 Connect primary power cable from connector to ac source.
Step 3 Connect input cable from computer to connector
Step 4 Open left side access door; unlatch and swing out card cage A3.
Step 5 Set circuit breaker A4CB1 to ON.
Step 6 Close and latch card cage A3; close access door.
Step 7 Set circuit breaker 4 to ON; ensure the POWER indicator and PAPER FAULT indicator light.
Step 8 Lift and latch printer window; unlatch and open drum gate A2A1 and ensure that DRUM GATE indicator lights.
Step 9 Set TOP OF FORM switch to up position and release.
Step 10 Set COPIES CONTROL lever to number of copies desired.
Step 11 Open the spring-loaded pressure plates on both tractors.
Step 12 Place paper on tractor pins; perform step 13 if adjustment of tractors for paper width is required.
Step 13 Loosen paper width adjustment levers; set tractors to correct paper width and tighten levers.
Step 14 Align paper perforation to top-of-form indicators on hammer bank.
Step 15 Close the pressure plates on both tractors.
Step 16 Close and latch drum gate; ensure that PAPER FAULT and DRUM GATE indicators go off.
Step 17 Unlatch and close printer window.

When the line printer has been used and it is expected that it will not be used for some time, set ON LINE/OFF LINE switch to OFF LINE. Next, set circuit breaker (4) to OFF.

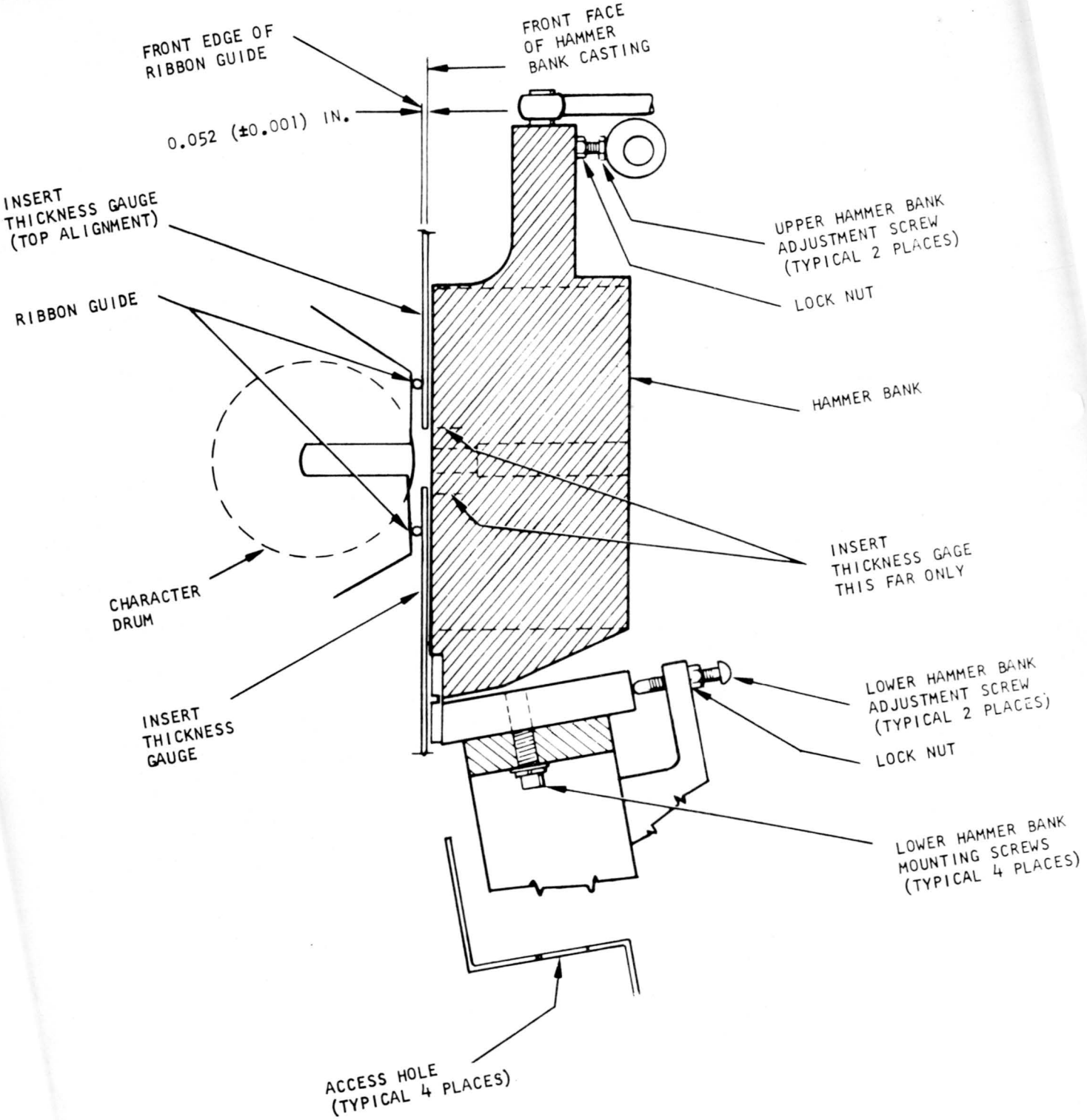


Figure 35 Hammer Bank Alignment

Replacing the ribbon

Step 1 Lift maintenance panel access cover and set circuit breaker (4) to OFF.

Step 2 Lift and latch printer window; move drum gate latch to the left and back.

Step 3 Swing drum gate fully open.

Step 4 Grasp corner of paper guide and swing open.

Step 5 Hold ribbon cores together and remove ribbon from box.

Step 6 Place fully wound ribbon core over top floating ribbon holder.

Step 7 Push against floating holder spring and place opposite core end on top fixed ribbon holder; ensure holder guide pin slips into core end.

Step 8 Unwind second ribbon and bring ribbon down and over drum.

Step 9 Place core on bottom ribbon holders as in step 8.

Step 10 Close paper-guide.

Step 11 Close drum gate; move drum gate latch forward.

Step 12 Unlatch and close printer window.

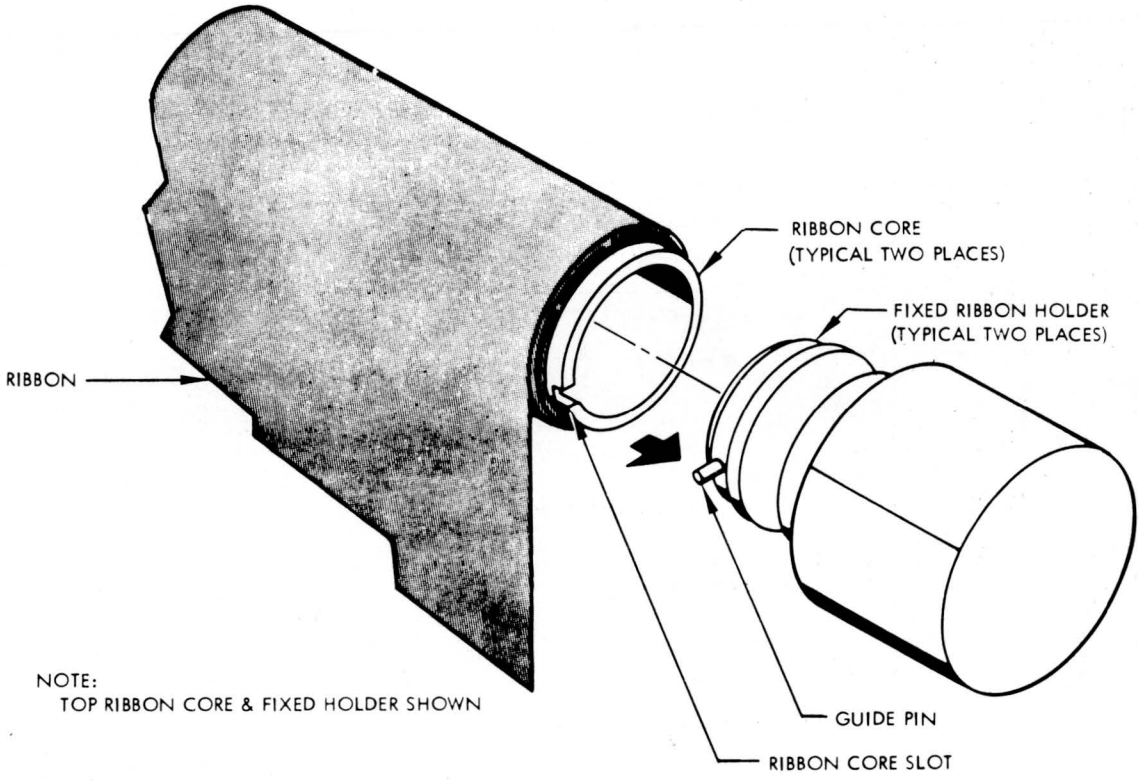


Figure 36 Installation of Ribbon Core on Fixed Ribbon Holder

The control tape

The control tape allows the programmer to skip, rather than space through, unused sections of a form. Eight-channel tape is used, giving the programmer eigth skip instructions including 'top of form' which uses tape channel 1. Figure 37 shows how the control tape punch relates to the printers. The hole punched in tape channel 1 allows the form to be skipped to the first line that is to be printed ('top of form'), in this case the customer's name. This line is the reference point from which all other skip points are counted and depends upon the form being correctly positioned on the tractors. After the address has been printed, the tape channel 2 skip instruction is used to skip the form to the data line. The form can then be skipped to each skip point set by the tape and, if a section of the form is not used, the corresponding skip instructions need not be given. If, for example, there were to be no sixth and seventh entries, an instruction to skip to tape channel 8 skip point could be given after the fifth entry, taking the form to where the total would be printed out.

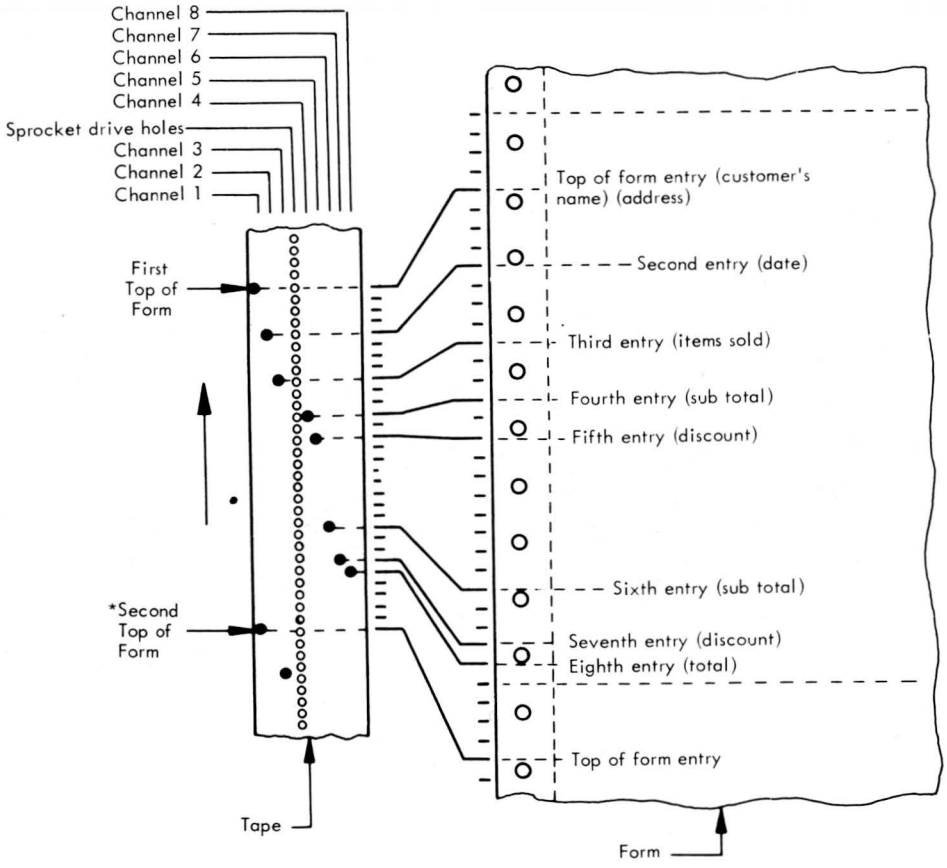


Figure 37 Tape loop preparation

Note:

Paper line spacing is six lines per inch. Each hole space on tape corresponds to one paper line space.

*only if tape loop is two or more form lengths long.

Punching a control tape

The line guide used in making up a control tape is shown in Figure 38.

Step 1 Place a length of tape and the required form on the tape loop guide as shown in Figure 37.

Step 2 Mark the tape in channel 1 ('top of form' channel) alongside the 'top of form' entry position.

Step 3 Mark the tape in the channels specified by the programmer alongside all the other print entry positions.

Step 4 Mark the tape in channel 1 alongside the next 'top of form' entry position.

Step 5 Using a manual or machine tape punch, replace the marks on the tape with holes.

Step 6 Form the tape into a loop so that the channel 1 holes overlap exactly and splice the tape in that position. If the form is so short that the resulting tape is too small to be practical, repeat steps 3, 4 and 5 as many times as is necessary to produce a loop of workable size.

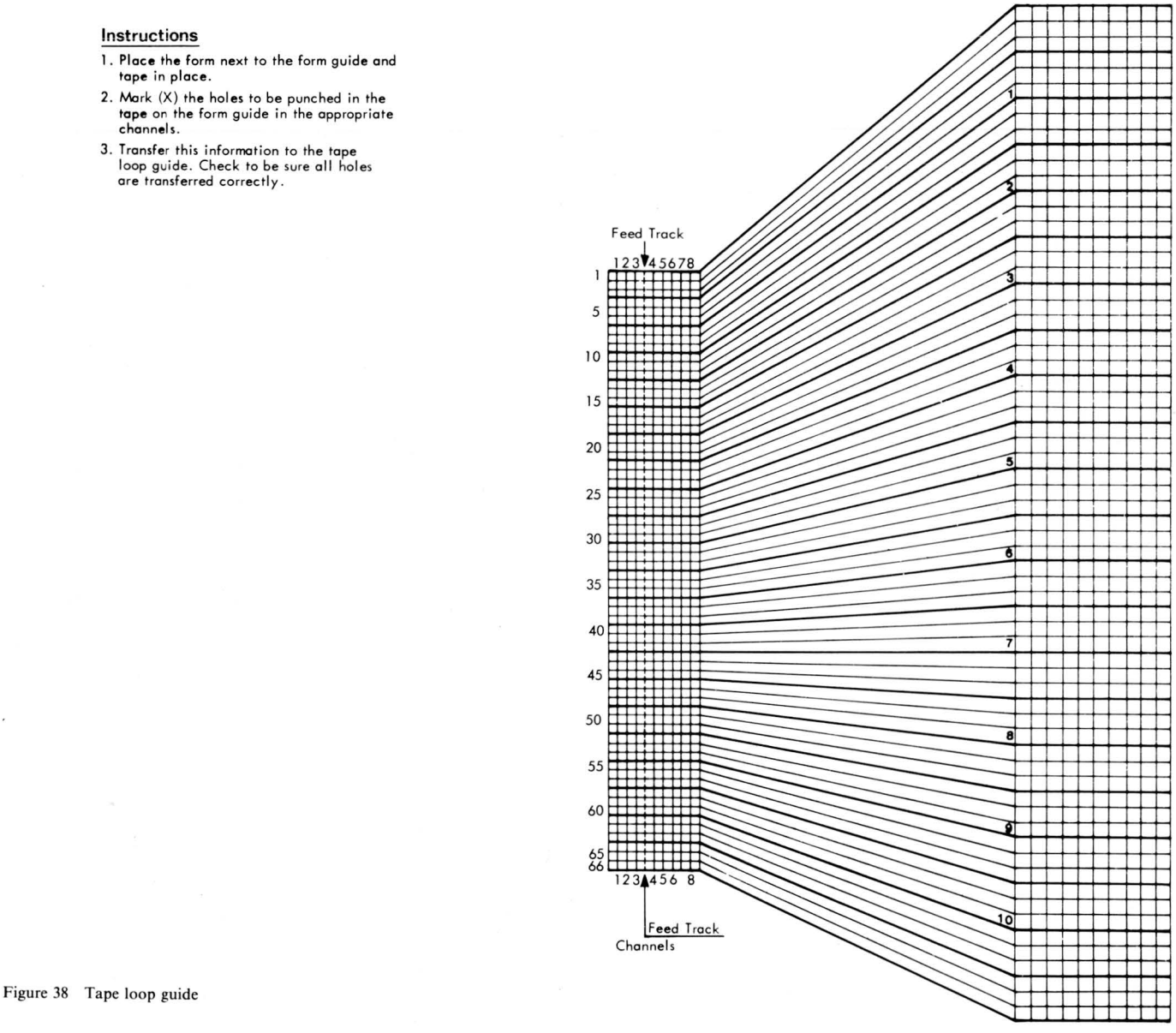


Figure 38 Tape loop guide

Loading the tape loop

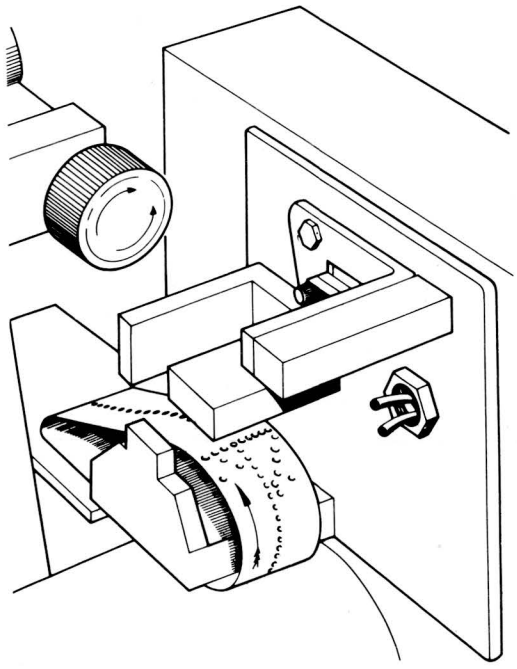


Figure 39 Tape reader

Step 1 Check that the POWER button is lit.
Step 2 Raise the printer top cover.
Step 3 Lift the tape reader handle until the drive sprocket shoe clears the tape drive sprocket wheel.
Step 4 Place the tape loop over the capstan and fit the sprocket holes in the tape over the teeth of the wheel. Make sure the tape is fitted the right way round (with channel 1 towards the short end of the capstan) and does not overlap the end of the capstan.
Step 5 Lower the tape reader handle while making sure that the tape stays on the sprocket wheel.
Step 6 Push the TOP OF FORM switch to advance the tape to the 'top of form' position.
Step 7 If the paper supply has already been loaded, open the pressure plates and move the paper so that the first print entry is in line with the uppermost 'x' on each side of the ribbon mask (if there is to be a paper advance before printing, use the middle 'x's).

Operator's maintenance

Inspect the drum for excessive dust and ink residue. If necessary, clean with alcohol or trichloroethylene, using a small stiff brush. Inspect the tractor mechanism for ease of movement along the splined shafts. Apply oil, if necessary. All other maintenance operations are carried out periodically by the service engineer. Fault conditions, which the user cannot remedy, should be reported together with all relevant information.

DISC UNIT

Figure 40 P822-001 Disc Unit (2.7M).

The P822-001 (Philips X1210) is a moving head disc with interchangeable disc packs. The disc cartridge is top-loaded onto the drawer type disc drive which can be mounted in a standard 19 inch rack.
The operator's attention is only required for loading and unloading the disc pack and for starting and stopping the unit.

Controls

The control panel contains the following control pushbutton switch and indicators.

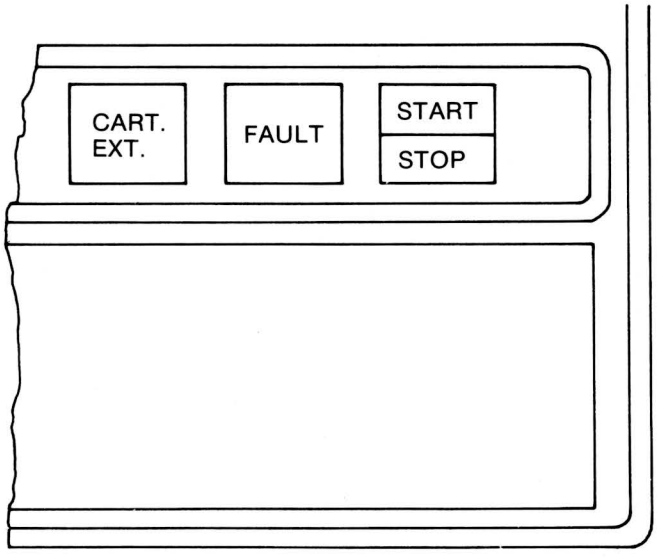


Figure 41 Front Panel Controls

START/STOP: A pushbutton switch to switch the unit on and off. It lights green when the unit is on.

CARTRIDGE EXCHANGE: Lights green when the power supplies have been connected to the unit and when it is safe to load or remove a cartridge.

FAULT: Indicator which lights red when a fault condition has occurred.

Loading a cartridge

Before loading the cartridge check that the only lamp lit is the EXCHANGE CARTRIDGE lamp.

If the unit is already loaded refer to the instructions for removing a cartridge.

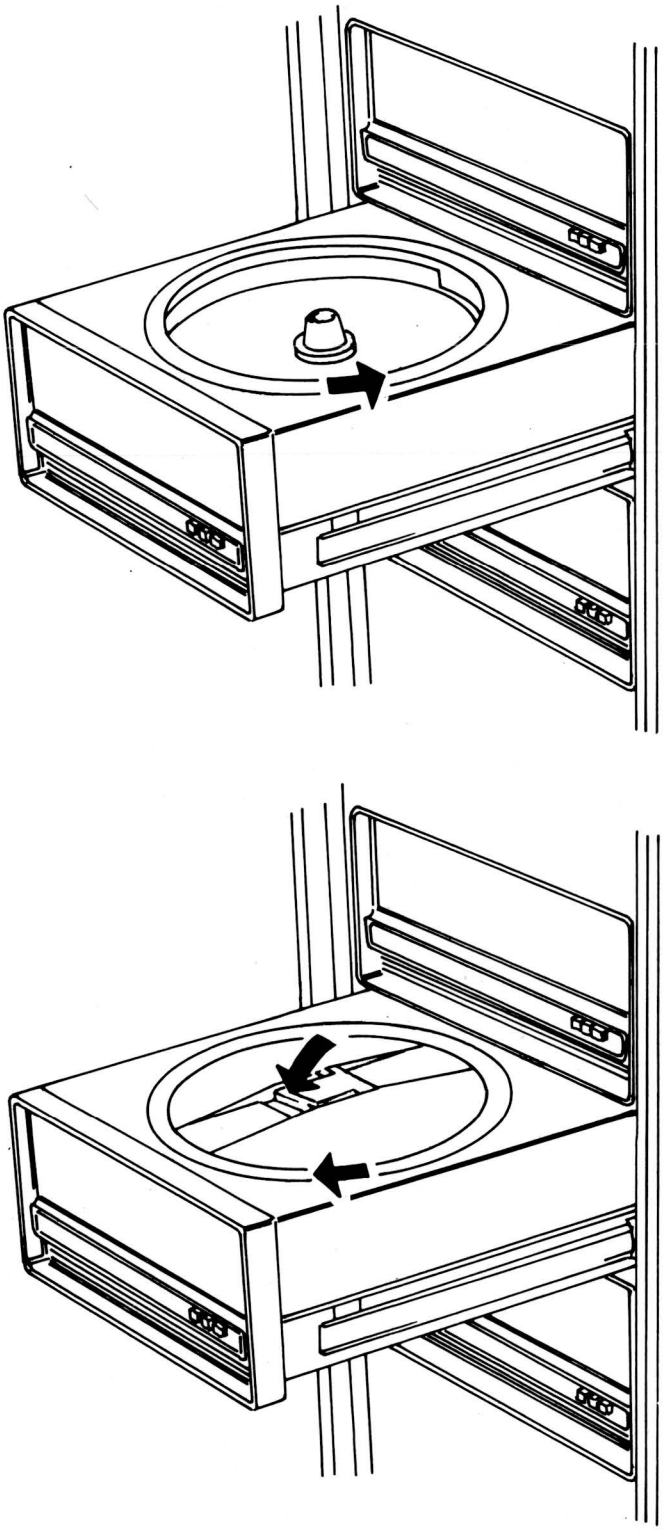


Figure 42 Loading a cartridge

Step 1 Slide the disc unit from the rack unit until further withdrawal is stopped by the slide stops.

Step 2 Check that the cartridge holder locking ring is fully turned anticlockwise.

Step 3 Place the cartridge carefully on the drive shaft while making sure the cartridge does not knock against the shaft.

Step 4 Lock the cartridge into the holder by turning the locking ring clockwise.

Step 5 Push the handle into the recess on the cartridge's top surface before pushing the disc unit into the rack.

Starting the disc unit

The unit can be started only if the cartridge has been loaded correctly and the unit has been completely pushed into the rack unit.

When the START button is pressed this button will light and the CARTRIDGE EXCHANGE light will extinguish.

Failure to start

If the CARTRIDGE EXCHANGE lamp remains lit, the FAULT lamp will light and the START/STOP indicator remains extinguished.

Press the START/STOP button twice. If the unit still fails to start, check that the cartridge loading sequence has been correctly performed. If the unit still fails to start call the service engineer.

Faults

A failure in the operation is indicated by the FAULT indicator. When it is lit stop the disc unit by pressing the START/STOP button. When the unit stops the indicator in this button will go out. The FAULT indicator remains lit and the CARTRIDGE EXCHANGE lamp will light.

Next, restart the unit. In case of a temporary fault condition, the START/STOP indicator will light and the other two indicators will extinguish.

If the fault condition is a permanent one the FAULT lamp will light again approximately 10 seconds after the START/STOP button has been pressed.

Stopping the disc unit

Push the START/STOP button. The light in the button will go out and the CARTRIDGE EXCHANGE lamp will light.

Removing the cartridge

Step 1 Check if the CARTRIDGE EXCHANGE lamp is lit.
Step 2 Pull the disc unit from the rack until further withdrawal is stopped by the slide stops.
Step 3 Lift cartridge handle and push it backwards towards the rack.
Step 4 Turn the locking ring anti-clockwise.
Step 5 Lift cartridge from the unit.

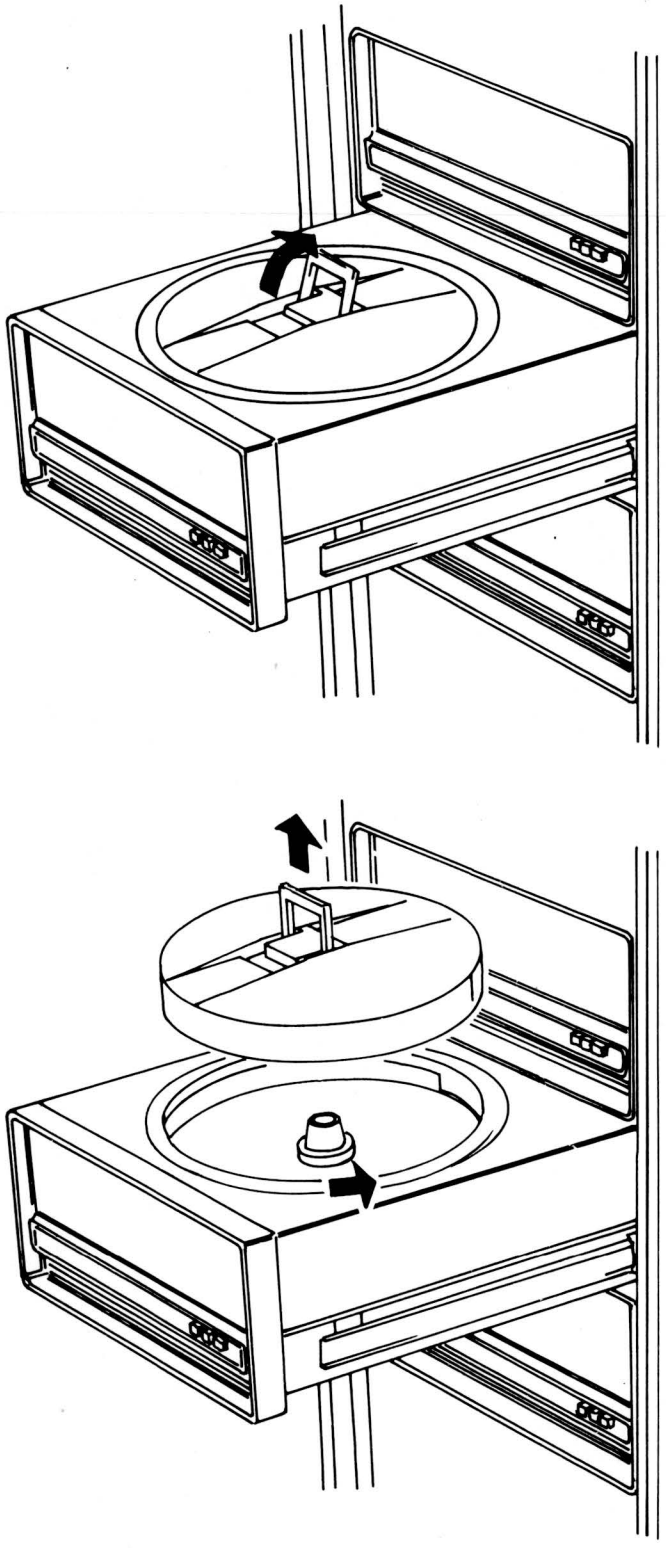


Figure 43 Removing the cartridge

PLOTTER

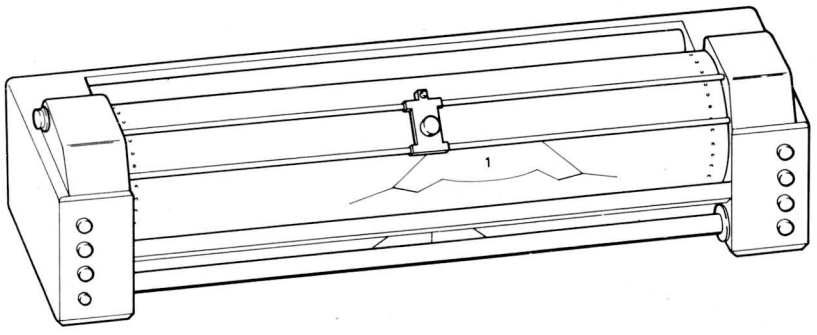


Figure 44 Digital incremental plotter

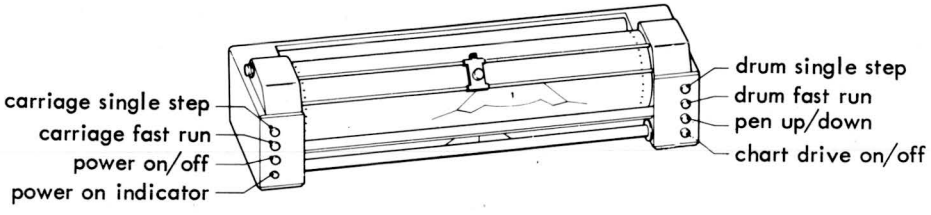


Figure 45 Control panel

The P813-001 Calcomp 565 plotter is a high-speed, drum-type plotter designed for plotting one variable against another. The Y-axis plot is produced by lateral movement of a pen and the X-axis plot by rotation of a chart-paper drum. Z-axis modulation is provided by lifting the pen from the paper. All three axes are controlled by the computer without operator attention after the paper has been loaded and the initial adjustments made. The plotter will accept either single sheets of chart-paper of rolls containing 120 feet of paper.

Controls and indicators

The seven operating controls and one indicator described below are mounted on the front panel of the plotter (Figure 45).

POWER ON/OFF: A two-position switch to switch the plotter on and off.

POWER ON indicator: A neon indicator to show that the plotter is switched on.

Carriage fast run: A three-position (centre off) switch to step the pen carriage rapidly to left or right.

Carriage single step: A three-position (centre off) switch to step the pen carriage, one step at each switch movement, to the right of left.

Chart drive ON/OFF: A two-position switch to switch off the chart roll take-up and supply motors when single sheets of chart-paper are used.

Pen UP/DOWN: A two-position switch to lift the pen from the writing position.

Note: When the plotter is first switched on or after the pen has been removed, it may be necessary to operate the switch more than once to lift the pen.

Drum fast run: A three-position (centre off) switch to step the chart drum rapidly up or down.

Drum single step: A three-position (centre off) switch to step the chart drum, one step at each switch movement, up or down.

Operating the plotter

Step 1 Check that the supply roll contains sufficient paper for the job that is to be run; if a single sheet of paper is to be used, check that it is of adequate size.

Step 2 Check that the paper is smooth and flat against the drum.

Step 3 Turn the POWER ON/OFF switch to ON. The POWER ON indicator should light. Turn on the chart drive.

Step 4 Position the pen carriage and drum to the required starting point on the paper using the CARRIAGE

and DRUM FAST RUN and SINGLE STEP controls. The exact point at which the pen will start to write can be seen by removing the pen and fitting the reticle in to the pen socket. Alternatively, the upper reticle can be used with the pen still in place to see a point on the paper exactly one inch above the pen.

Step 5 Turn the PEN switch first to DOWN an then to UP. The plotter is now ready to operate under computer control.

Installing a chart roll

Before installing a new roll of chart-paper, make sure that the roll is evenly wound on the core; the end of the roll must not be 'coned'. Straighten a coned roll by tamping the end against a flat surface.

Step 1 Turn the POWER ON/OFF switch to OFF.

Step 2 Remove the pen assembly from the carriage by loosening the knurled nut at the bottom of the pen holder and lifting the assembly out of the carriage.

Note: The pen assembly is easily damaged and should be handled carefully.

Step 3 Rotate the right rear paper spool by hand until the drive key is pointing upwards.

Step 4 Hold the new roll of chart-paper so that the key slot in the core is pointing upwards. Push the lefthand end to the idler spool to the left with the thumb and slip the lefthand end of the roll on to the spool. Do not use the paper roll to push the idler spool aside as this tends to cone the roll and cause misalignment.

Step 5 Lower the paper roll into the paper well and slide the righthand end on to the drive spool. Make sure that the drive key engages in the key slot in the roll core.

Step 6 Install an empty paper roll core on the two front spools below the drum in the same way as the paper roll was installed.

Step 7 Pull the end of the paper over the drum so that the sprocket holes on both edges of the paper engage the sprocket teeth on the drum. Guide the chart-paper under the carriage rods, behind the tear bar, and behind the roll core. Fasten the end of the paper to the roll core with adhesive tape. Use the DRUM FAST RUN switch in the DOWN position to wind a few turns of the paper on to the take-up spool.

Step 8 Replace the pen assembly in the carriage and tighten the knurled nut at the bottom of the pen holder. Scales on the inside faces of the rear paper spool flanges indicate (in feet) the approximate amount of paper that remains on the roll. The black scale is for chart-paper 0.076 mm thick while the blue scale is for 0.005 mm paper.

Installing a single sheet of chart-paper

Step 1 Turn the POWER ON/OFF switch to OFF.

Step 2 Remove the pen assembly from the carriage by loosening the knurled nut at the bottom of the pen holder and lifting the assembly out of the carriage.

Note: The pen assembly is easily damaged and should be handled carefully.

Step 3 Slide the chart-paper under the carriage rods on to the drum surface.

Step 4 Fasten the top edge of the paper to the drum with two or three short pieces of adhesive tape. Rotate the drum by hand, keeping the paper smooth and flat against the drum surface. Fasten the bottom edge of the paper to the drum with two or three short pieces of adhesive tape.

Step 5 Replace the pen assembly in the carriage and tighten the knurled nut at the bottom of the pen holder.

Removing the chart-paper

The roll of chart-paper or single sheet of paper should be removed in the reverse sequence to the installation procedure described above. Finished work can be removed by taking out the lower paper roll and tearing the paper on the tear bar. The roll core should then be refitted and the new end of the paper taped to it as described above. After a single sheet of paper has been removed from the drum, any adhesive left on the drum should be removed with a cloth or tissue paper moistened in cleaning solvent or alcohol.

Operational checkout

The following procedure provides an overall check of the plotter and can be used before starting to record from the computer.

Step 1 Install the pen assembly in the carriage and tighten the knurled nut at the bottom of the pen holder.

Step 2 Turn the POWER ON/OFF and CHART DRIVE switches to ON. If the chart roll is not to be used, the CHART DRIVE switch should be attached to the drum.

Step 3 Turn the PEN switch to DOWN, then UP to check that the pen lifts off the drum surface. Turn the switch back to the DOWN position.

Step 4 Turn the DRUM FAST RUN switch to the UP position and check that the pen traces a vertical line on the chart.

Step 5 Turn the CARRIAGE FAST RUN switch to the LEFT position and check that the pen traces a horizontal line on the chart and that the carriage motor stops when the carriage reaches the limit of travel.

Step 6 Turn the DRUM FAST RUN switch to DOWN and check that the pen traces a vertical line on the chart.

Step 7 Turn the CARRIAGE FAST RUN switch to the RIGHT position and check that the pen traces a horizontal line on the chart and that the carriage motor stops when the carriage reaches the limit of travel.

Step 8 Operate, alternatively, the CARRIAGE SINGLE STEP and DRUM SINGLE STEP switches to check that both the carriage and the drum move only one step each time a switch is operated.

Step 9 Move the carriage to the left margin of the paper. Turn the CARRIAGE FAST RUN switch to RIGHT and the DRUM FAST RUN switch to DOWN. Allow the plotter to run until the carriage reaches the right side margin before turning both switches to OFF.

The pen should have traced a 45° line across the chart. Check the line carefully for any evidence of discontinuity.

Step 10 Operate the DRUM SINGLE STEP switch several times to move the pen away from the line drawn in (9).

Step 11 Turn the CARRIAGE FAST RUN switch to LEFT and the DRUM FAST RUN switch to UP. Allow the plotter to run until the carriage reaches the left side margin before turning both switches to OFF. The pen should have traced a 45° line across the chart. Check that the line is continuous and parallel to the line drawn in (9).

Step 12 Repeat (9), (10) and (11) changing the switch position to produce two 45° lines at right-angles to the first two.

Alignment recticles (see Figure 46)

Two alignment reticles are fitted to the pen carriage so that the pen can be positioned accurately on a zero point by using the single step controls. The lower reticle replaces the pen assembly and shows the exact point where the pen will start to write. The upper reticle remains in place when the pen is installed and shows a point exactly one inch (25.4 mm) above the point of the pen. The reticles are held in their sockets by 0-rings and, to avoid damage to the 0-rings, the reticles should be rotated as they are pushed into place.

Operator's maintenance

The carriage rods, drum surface and the metal plunger inside the pen should be cleaned after each eight hours or so of use. The carriage rods and drum are best cleaned with a dry cloth, although a little cleaning solvent may be used to remove accumulated dirt or tape adhesive. Clean the pen plunger with a dry cloth by pushing a corner of the cloth through the plunger. If the plunger becomes clogged, dip it in cleaning solvent and wipe dry, using a cotton bud to dry out the inside of the solenoid. All other maintenace operations are carried out periodically by the service engineer.

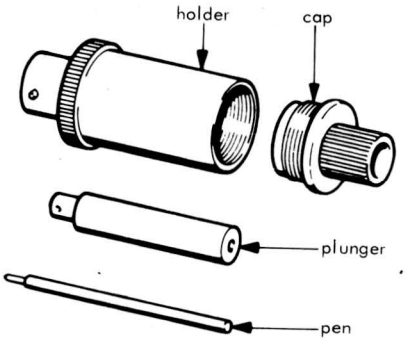


Figure 47 Pen assembly

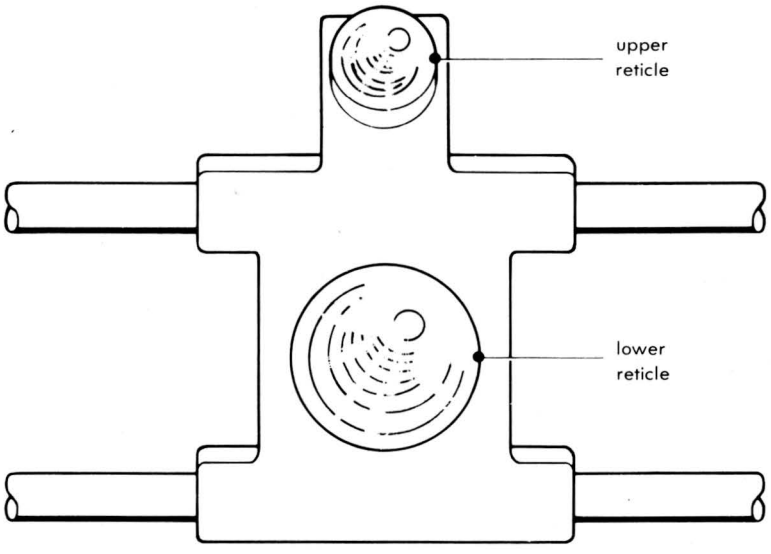


Figure 46 Reticle adjustment

PART 3 INSTRUCTION SET

Memory Reference Instructions

LD: Load register

1	0 0 0 0	R1	MD	R2	0
1	4	4	2	4	1

Syntax
[<ident>]␣LD[⋆]␣<r1>,<m>[,<r2>]

The 16 bits of the register specified by R1 are replaced by the contents of the effective memory address (M).

Type	*Function*		*Mode (MD)*	*Execution Time*	
				P855	*P860*
T4	(M)	→ R1	10	3.6	2.52
T5	(M+(R2))	→ R1	10	3.6	2.52
T6	((M))	→ R1	11	4.8	3.36
T7	((M+(R2)))	→ R1	11	4.8	3.36

Condition register
CR = 0 if (R1) = 0
1 if (R1) > 0
2 if (R1) < 0

ST: Store register

1	0 0 0 0	R1	MD	R2	1
1	4	4	2	4	1

Syntax
[<ident>]␣ST[⋆]␣<r1>,<m>[,<r2>]

The 16 bits of the register specified by R1 replace the contents of the memory effective address (M).

Type	*Function*		*Mode (MD)*	*Execution Time*	
				P855	*P860*
T4	(R1)	→ M	10	4.8	3.36
T5	(R1)	→ M+(R2)	10	4.8	3.36
T6	(R1)	→ ((M))	11	6.0	4.20
T7	(R1)	→ ((M + (R2)))	11	6.0	4.20

Condition register
Unchanged.

AD: Addition

1	0 0 1 0	R1	MD	R2	I/s
1	4	4	2	4	1

Syntax
[<ident>]␣AD[S] [⋆]␣<r1>,<m>[,<r2>]

The contents of the memory effective address are added to the contents of the 16-bit register specified by R1. The sum is placed either in R1 or in the memory effective address, depending on the state of Load or Store indicator.

Type	*Function*		*Mode (MD)*	*L/S*	*Execution Time*	
					P855	*P860*
T4	(R1)+(M)	→ R1	10	0	3.6	2.52
T4	(R1)+(M)	→ M	10	1	4.8	3.36
T5	(R1)+(M+(R2))	→ R1	10	0	3.6	2.52
T5	(R1)+(M+(R2))	→ M+(R2)	10	1	4.8	3.36
T6	(R1)+((M))	→ R1	11	0	4.8	3.36
T6	(R1)+((M))	→ (M)	11	1	6.0	4.20
T7	(R1)+((M+(R2)))	→ R1	11	0	4.8	3.36
T7	(R1)+((M+(R2)))	→ (M+(R2))	11	1	6.0	4.20

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in case of overflow

Remarks
When L/S bit = 1, R1 must be ≠ 0.

SU: Subtract word

1	0 0 1 1	R1	MD	R2	I/s
1	4	4	2	4	1

Syntax

[<ident>]␣SU[S][⋆]␣<r1>,<m>[,<r2>]

The contents of the memory effective address are subtracted from the contents of the 16-bit register specified by R1. The result is placed either in R1 or in effective memory address, depending on the state of Load or Store indicator.

Type	*Function*			*Mode (MD)*	*L/S*	*Execution Time*	
						P855	*P860*
T4	(R1) - (M)	→	R1	10	0	3.6	2.52
T4	(R1) - (M)	→	M	10	1	4.8	3.36
T5	(R1) - (M + (R2))	→	R1	10	0	3.6	2.52
T5	(R1) - (M + (R2))	→	M + (R2)	10	1	4.8	3.36
T6	(R1) - ((M))	→	R1	11	0	4.8	3.36
T6	(R1) - ((M))	→	(M)	11	1	6.0	4.20
T7	(R1) - ((M + (R2)))	→	R1	11	0	4.8	3.36
T7	(R1) - ((M + (R2)))	→	(M + (R2))	11	1	6.0	4.20

Condition register

CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in event of overflow

Remarks

When L/S bit = 1, R1 must be ≠ 0.

AN: Logical AND

1	0 1 0 0	R1	MD	R2	I/s
1	4	4	2	4	1

Syntax

[<ident>]␣AN[S] [⋆]␣<r1>,<m>[,<r2>]

Logical product

Bit in R1	Bit in second operand	Logical product
0	0	0
0	1	0
1	0	0
1	1	1

The logical product of the contents of the memory effective address and the contents of the 16-bit register specified by R1 is placed in R1-register or in memory effective address, depending on the state of Load or Store indicator.

Type	*Function*			*Mode (MD)*	*L/S*	*Execution Time*	
						P855	*P860*
T4	(R1) ∧ (M)	→	R1	10	0	3.6	2.52
T4	(R1) ∧ (M)	→	M	10	1	4.8	3.36
T5	(R1) ∧ (M + (R2))	→	R1	10	0	3.6	2.52
T5	(R1) ∧ (M + (R2))	→	M + (R2)	10	1	4.8	3.36
T6	(R1) ∧ ((M))	→	R1	11	0	4.8	3.36
T6	(R1) ∧ ((M))	→	(M)	11	1	6.0	4.20
T7	(R1) ∧ ((M + (R2)))	→	R1	11	0	4.8	3.36
T7	(R1) ∧ ((M + (R2)))	→	(M + (R2))	11	1	6.0	4.20

Condition register

CR = 0 if product = 0
1 if product > 0
2 if product < 0

OR: Logical OR

1	0 1 0 1	R1	MD	R2	I/s
1	4	4	2	4	1

Syntax
[<ident>]␣OR[S][⋆]␣<r1>,<m>[,<r2>]

Logical Union

Bit in R1	Bit in second operand	Logical Union
0	0	0
0	1	1
1	0	1
1	1	1

The logical OR of the contents of the memory effective address (M) and the contents of the 16-bit register specified by R1 is placed either in the R1-register or in the memory effective address depending on the state of the Load/Store indicator.

Type	*Function*		*Mode (MD)*	*L/S*	*Execution Time*	
					P855	*P860*
T4	(R1) ∨ (M)	→ R1	10	0	3.6	2.52
T4	(R1) ∨ (M)	→ M	10	1	4.8	3.36
T5	(R1) ∨ (M+(R2))	→ R1	10	0	3.6	2.52
T5	(R1) ∨ (M+(R2))	→ M+(R2)	10	1	4.8	3.36
T6	(R1) ∨ ((M))	→ R1	11	0	4.8	3.36
T6	(R1) ∨ ((M))	→ (M)	11	1	6.0	4.20
T7	(R1) ∨ ((M+(R2)))	→ R1	11	0	4.8	3.36
T7	(R1) ∨ ((M+(R2)))	→ (M+(R2))	11	1	6.0	4.20

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0

XR: Exclusive OR

1	0 1 1 0	R1	MD	R2	I/s
1	4	4	2	4	1

Syntax
[<ident>]␣XR[S][⋆]␣<r1>,<m>[,<r2>]

Exclusive OR

Bit in R1	Bit in second operand	Exclusive OR
0	0	0
0	1	1
1	0	1
1	1	0

The exclusive OR of the contents of 16-bit memory effective address and the contents of 16-bit register specified by R1 is placed either in R1 or in memory effective address depending on the state of Load/Store indicator.

Type	*Function*		*Mode (MD)*	*L/S*	*Execution Time*	
					P855	*P860*
T4	(R1) ∀ (M)	→ R1	10	0	3.6	2.52
T4	(R1) ∀ (M)	→ M	10	1	4.8	3.36
T5	(R1) ∀ (M+(R2))	→ R1	10	0	3.6	2.52
T5	(R1) ∀ (M+(R2))	→ M+(R2)	10	1	4.8	3.36
T6	(R1) ∀ ((M))	→ R1	11	0	4.8	3.36
T6	(R1) ∀ ((M))	→ (M)	11	1	6.0	4.20
T7	(R1) ∀ ((M+(R2)))	→ R1	11	0	4.8	3.36
T7	(R1) ∀ ((M+(R2)))	→ (M+(R2))	11	1	6.0	4.20

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0

IM: Increment memory

1	0 0 1 0	0 0 0 0	MD	R2	1
1	4	4	2	4	1

Syntax
[<ident>]␣IM[⋆]␣<m>[,<r2>]

The contents of the effective memory address, increased by one, replace the contents of the effective memory address.

Type	*Function*			*Mode (MD)*	*Execution Time*	
					P855	*P860*
T4	(M) + 1	→	M	10	4.8	3.36
T5	(M + (R2)) + 1	→	M + (R2)	10	4.8	3.36
T6	((M)) + 1	→	(M)	11	6.0	4.20
T7	((M + (R2))) + 1	→	(M + (R2))	11	6.0	4.20

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in event of overflow.

C2: Two's Complement

1	0 0 1 1	0 0 0 0	MD	R2	1
1	4	4	2	4	1

Syntax
[<ident>]␣C2[⋆]␣<m>[,<r2>]

The two's complement of the contents of the memory effective address, indicated by the word following the instruction, replace the old contents.

Type	*Function*			*Mode (MD)*	*Execution Time*	
					P855	*P860*
T4	0 - (M)	→	M	10	4.8	3.36
T5	0 - (M + (R2))	→	M + (R2)	10	4.8	3.36
T6	0 - ((M))	→	(M)	11	6.0	4.20
T7	0 - ((M + (R2)))	→	(M + (R2))	11	6.0	4.20

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in event if overflow.

ML: Multiple load

1	0 1 1 1	n	MD	R2	0
1	4	4	2	4	1

Syntax
[<ident>]␣ML[⋆]␣<n>,<m>[,<r2>]
<n> = number of registers 0<number<15

The contents of n consecutive registers (the first is register 1) are replaced by the contents of n consecutive memory locations (first location is indicated by the effective address). n may have any value between 1 and 15.

Type	*Function*			*Mode(MD)*	*Execution Time*	
T4	(M) ... (M + n)	→	A1 ... An	10	*P855*	n×1.2 +2.4
					P860	n×0.84+1.68
T5	(M + (R2)) ... (M + (R2) + n)	→	A1 ... An	10	*P855*	n×1.2 +2.4
					P860	n×0.84+1.68
T6	((M)) ... (MM) + n)	→	A1 ... An	11	*P855*	n×1.2 +3.6
					P860	n×0.84+2.52
T7	((M + (R2))) ... ((M + (R2)) + n)	→	A1 ... An	11	*P855*	n×1.2 +3.6
					P860	n×0.84+2.52

Condition register
CR = 0 if (A1) = 0
1 if (A1) >
2 if (A1) <

MS: Multiple store

1	0 1 1 1	n	MD	R2	1
1	4	4	2	4	1

Syntax
[<ident>]␣MS[⋆]␣<n>,<m>[,<r2>]
<n> = number of registers 0<number<,15

The contents of n consecutive registers (the first is register 1) replace the contents of n consecutive memory locations (first location is indicated by the effective address). n may have any value between 1 and 15.

Type	*Function*	*Mode(MD)*	*Execution Time*	
T4	A1 ... An → M ... M + n	10	*P855*	n×1.9 +2.4
			P860	n×1.56+1.68
T5	A1 ... An → M + (R2) ... M + (R2) + n	10	*P855*	n×1.9 +2.4
			P860	n×1.56+1.68
T6	A1 ... An → (M) ... (M) + n	11	*P855*	n×1.9 +2.4
			P860	n×1.56+1.68
T7	A1 ... An → (M + (R2)) ... (M + (R2)) + n	11	*P855*	n×1.9 +2.4
			P860	n×1.56+1.68

Condition register
Unchanged.

ABI: Absolute branch

1	0 0 0 1	CND		MD	R2	0
1	4	3	1	2	4	1

Syntax
[<ident>]␣ABI [<cnd>][⋆]␣<m>[,<r2>]

This instruction means that the address of the next instruction to be executed is found either at the memory effective address or in normal sequence, depending on the contents of CND field and CR. If CND is equal to 111 the next instruction is always at the memory address (unconditional branch).

CND contents	Meaning	CND contents	Meaning
000	= 0	100	≠ 0
001	= 1	101	≠ 1
010	= 2	110	≠ 2
011	= 3	111	unconditional

Type	*Function*			*Mode (MD)*	*Execution Time*	
	Branch Effective		*No Branch*		*P855*	*P860*
T4	(M)	→ P	(P)+4 → P	10	3.6	2.52
T5	(M+(R2))	→ P	(P)+4 → P	10	3.6	2.52
T6	((M))	→ P	(P)+4 → P	11	4.8	3.36
T7	((M+(R2)))	→ P	(P)+4 → P	11	4.8	3.36
			No Branch:		1.9	1.56

Condition register
Unchanged.

LINKING TO SUBROUTINES

The contents of the program status word (PSW) and the contents of the P-register are stored in successive locations in a memory stack, whose ending address has been previously specified by the programmer. A branch is then performed to a subprogram located at the address given in the instruction. If any other parameters of the calling program have to be saved, this should be done before the CF instruction is issued, using a Load instruction with a stack pointer in register 15.

CFI: Call Function

1	1 1 1 0	R1	MD	R2	1
1	4	4	2	4	1

Syntax
[<ident>]␣CFI:[⋆]␣<r1>,<m>[,<r2>]

This instruction provides a link to a subroutine by storing successively the contents of the P-register and the contents of the program status word (PSW) in a memory stack. The stack pointer specified by the contents of R1 is automatically updated. Then a branch is made to the contents of the memory effective address.

Type	*Function*			*Mode (MD)*	*Execution Time*	
					P855	*P860*
T4, T5	(P) → (R1),		(R1) − 2 → R1	n.a.		
T6, T7	(PSW) → (R1)		(R1) − 2 → R1	n.a.		
then:						
T4	(M)	→ P		10	7.44	5.64
T5	(M+R2)	→ P		10	7.44	5.64
T6	((M))	→ P		11	8.64	6.48
T7	((M+(R2)))	→ P		11	8.64	6.48

The Condition Register is not altered by this operation. Its contents, showing the result of a previous operation, are stored in the memory stack for use on return from the subprogram.

Remarks
If register 15 is used to contain the stack pointer, a *stack overflow* interrupt will be generated when the word address 128_{10} is reached by the pointer.

C1: One's complement

1	1 1 1 1	R1	MD	R2	I/s
1	4	4	2	4	1

Syntax
[<ident>]␣C1[S][⋆]␣[<r1>],<m>[,<r2>]

Logic Complement
Bits which contained 1 in the specified word or register become 0, and vice versa.

The logic complement of the memory effective address replaces either the contents of the 16-bit register specified by R1 or the contents of the memory effective address, depending on the state of the Load/Store indicator.

Type	*Function*		*Mode (MD)*	*L/S*	*Execution Time*	
					P855	*P860*
T4	$\overline{(M)}$	→ R1	10	0	3.6	2.52
T4	$\overline{(M)}$	→ M	10	1	4.8	3.36
T5	$\overline{(M+(R2))}$	→ R1	10	0	3.6	2.52
T5	$\overline{(M+(R2))}$	→ M+(R2)	10	1	4.8	3.36
T6	$\overline{((M))}$	→ R1	11	0	4.8	3.36
T6	$\overline{((M))}$	→ (M)	11	1	6.0	4.20
T7	$\overline{((M+(R2)))}$	→ R1	11	0	4.8	3.36
T7	$\overline{((M+(R2)))}$	→ (M+(R2))	11	1	6.0	4.20

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0

CW: Compare word

1	1 1 0 1	R1	MD	R2	I/s
1	4	4	2	4	1

Syntax
[<ident>]␣CW[⋆]␣<r1>,<m>[,<r2>]

The contents of the 16-bit register specified by R1 are compared with the contents of the memory effective address indicated by the word following the instruction. The result is stored in the condition register.

Type	*Function*	*Mode (MD)*	*Execution Time*	
			P855	*P860*
T4	(R1) ÷ (M) → CR	10	4.8	3.36
T5	(R1) ÷ (M + (R2)) → CR	10	4.8	3.36
T6	(R1) ÷ ((M)) → CR	11	6.0	4.20
T7	(R1) ÷ ((M + (R2))) → CR	11	6.0	4.20

Condition register
CR = 0 if (R1) = 2nd. operand
1 if (R1) > 2nd. operand
2 if (R1) < 2nd. operand

CC: Compare character

1	1 1 0 1	R1	MD	R2	1
1	4	4	2	4	1

Syntax
[<ident>]␣CC[⋆]␣<r1>,<m>[,<r2>]

The least significant bits of the register specified by R1 are compared with the right (C = 1) or left (C = 0) character of the effective memory address. The most significant bit of a character is not considered as a sign bit.
The result of the operation is stored in the condition register.

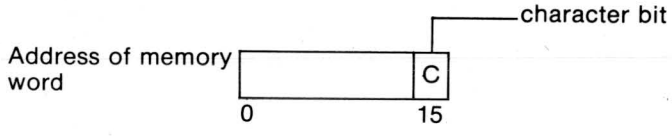
Type	*Function*	*Mode (MD)*	*Execution Time*	
			P855	*P860*
T4	(R1)r ÷ (M)l/r → CR	10	3.6	2.52
T5	(R1)r ÷ (M) + (X))l/r → CR	10	3.6	2.52
T6	(R1)r ÷ ((M))l/r → CR	11	4.8	3.36
T7	(R1)r ÷ ((M + (X)))l/r → CR	11	4.8	3.36

Condition register
CR = 0 if (R1)r = (2nd. operand)l/r
1 if (R1)r > (2nd. operand)l/r
2 if (R1)r < (2nd. operand)l/r

CHARACTER TRANSFER

An 8-bit character is transferred from a memory word to a register specified in the R1-field, or vice versa. The choise of the left-hand or right-hand character of the memory word is indicated by the setting of bit 15 of the word or register which contains the address of the memory word: i.e. if bit 15 is set to 1 the rightmost character is indicated; if to 0, the leftmost character.
Only the rightmost 8 bits of the register specified in R1 are concerned in the operation.
This instruction is musch used in input/output operations.

1	1 1 0 0	R1	MD	R2	I/s
1	4	4	2	4	1



LC: Load character

Syntax
[<ident>]␣LC[⋆]␣<r1>,<m>[,<r2>]

The right (C = 1) or left (C = 0) 8 bits of the memory effective address take place of the least significant 8 bits of the register specified by R1.

Type	*Function*		*Mode (MD)*	*Execution Time*	
				P855	*P860*
T4	(M)l/r	→ R1r	10	3.6	2.52
T5	(M+(R2))l/r	→ R1r	10	3.6	2.52
T6	((M))l/r	→ R1r	11	4.8	3.36
T7	((M+(R2)))l/r	→ R1r	11	4.8	3.36

Condition Register
Not affected by this operation.

Remarks
R1 must be ≠ 0.
Load/Store bit: 0.

SC: Store character

Syntax
[<ident>]␣SC[⋆]␣<r1>,<m>[,<r2>]

The least significant 8 bits of the register specified by R1 replace the right (C = 1) or left (C = 0) 8-bit contents of the memory effective address.

Type	*Function*		*Mode (MD)*	*Execution Time*	
				P855	*P860*
T4	(R1)r	→ (M)r/l	10	4.8	3.36
T5	(R1)r	→ (M+(R2))r/l	10	4.8	3.36
T6	(R1)r	→ ((M))r/l	11	6.0	4.20
T7	(R1)r	→ ((M+(R2)))r/l	11	6.0	4.20

Condition register
Unchanged

Remarks
R1 must be ≠ 0.
Load/Store bit: 1.

DA: Double add (optional)

1	1 0 1 0	0 0 0 0	MD	R2	0
1	4	4	2	4	1

Syntax
[<ident>]␣DA[⋆]␣<m>[,<r2>]

The contents of the memory effective address and memory effective address plus two are added to the contents of registers 1 and 2 (A1-A2). The sum is placed in A1-A2. The sign bit of A2 is never used; the sign bit of parameters and result is the sign bit of register 1.

Type	*Function*		*Mode (MD)*	*Execution Time*	
				P855	*P860*
T4	(M,M +1) + (A1, A2)	→ A1, A2	10	5.5	4.08
T5	(M + (R2)), M + (R2) + 1) + (A1, A2)	→ A1, A2	10	5.5	4.08
T6	((M), (M) + 1) + (A1, A2)	→ A1, A2	11	6.7	4.92
T7	((M + (R2))), ((M + (R2) + 1) + (A1, A2)	→ A1, A2	11	6.7	4.92

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in case of overflow

DS: Double subtract (optional)

1	1 0 1 1	0 0 0 0	MD	R2	0
1	4	4	2	4	1

Syntax
[<ident>]␣DS[⋆]␣<m>[,<r2>]

The contents of the memory effective address and the memory effective address plus two are subtracted from the contents of registers 1 and 2 (A1-A2). The result is placed in A1-A2. The sign bit of A2 is never used; the sign bit of parameters and result is the sign bit of register 1.

Type	*Function*		*Mode (MD)*	*Execution Time*	
				P855	*P860*
T4	(A1, A2) - (M, M + 1)	→ A1, A2	10	5.5	4.08
T5	(A1, A2) - (M + (R2), M + (R2) + 1)	→ A1, A2	10	5.5	4.08
T6	(A1, A2) - ((M), (M) + 1)	→ A1, A2	11	6.7	4.92
T7	(A1, A2) - ((M + (R2)), ((M + (R2)) + 1)	→ A1, A2	11	6.7	4.92

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in case of overflow

MU: Multiply (optional)

1	1 0 0 0	0 0 0 0	MD	R2	0
1	4	4	2	4	1

Syntax
[<ident>]␣MU[⋆]␣<m>[,<r2>]

The contents of the memory effective address are multiplied by the contents of register 2 (A2). The result is loaded as a 31-bit product in registers 1 and 2. The most-significant bit of register 2 is reset to zero. The sign of the product is stored in the sign bit of register 1.
Overflow occurs if result $<-2^{30}$, or result $>2^{30}-1$, in that case the two registers contain only the 30 least significant bits while the sign bit may or may not be correct.

Type	*Function*		*Mode (MD)*	*Execution Time*	
				P855	*P860*
T4	(A2)×(M)	→ A1 , A2	10	9.72	8.37
T5	(A2)×(M+(R2))	→ A1 , A2	10	9.72	8.37
T6	(A2)×((M))	→ A1 , A2	11	10.92	9.21
T7	(A2)×((M+(R2)))	→ A1 , A2	11	10.92	9.21

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in case of overflow

DV: Divide (optional)

1	1 0 0 1	0 0 0 0	MD	R2	0
1	4	4	2	4	1

Syntax
[<ident>]␣DV[⋆]␣<m>[,<r2>]

The contents of registers 1 and 2 (A1 and A2) are divided by the contents of the memory effective address. The quotient is placed in register 2, the remainder in register 1. The sign of the remainder is equal to the original sign of registers 1 and 2, except when the remainder is equal to zero (always positive).
Overflow occurs when the quotient exceeds 15 bits: the contents of registers 1 and 2 are destroyed, except when the divisor is equal to zero.

Type	*Function*		*quotient*	*remainder*	*Mode (MD)*	*Execution Time*	
						P855	*P860*
T4	(A1 , A2)/(M)	→	A2	A1	10	11.12	9.81
T5	(A1 , A2)/(M+(R2))	→	A2	A1	10	11.12	9.81
T6	(A1 , A2)/((M))	→	A2	A1	11	12.36	10.65
T7	(A1 , A2)/((M+(R2)))	→	A2	A1	11	12.36	10.65

Condition register
CR = 0 if (A2) = 0
1 if (A2) > 0
2 if (A2) < 0
3 in case of overflow

RELATIVE BRANCH INSTRUCTIONS: RF, RB

Two instructions are provided to make it possible to branch backwards or forward by a specified number of positions if a stated condition is fulfilled.

Like the Absolute Branch instruction, these instructions contain a condition field (CND), which indicates the following conditions:

CND contents	Meaning	CND contents	Meaning
0 0 0	= 0	1 0 0	≠ 0
0 0 1	= 1	1 0 1	≠ 1
0 1 0	= 2	1 1 0	≠ 2
0 1 1	= 3	1 1 1	unconditional

If the Condition Register contents resulting from the previous operation are identical with the condition in the CND-field, the P-register (instruction counter) is incremented by the number of words in bits 8 to 14 of the instruction, and the instruction sequence starting at the address thus reached is performed. If the condition is not fulfilled, the next instruction in normal sequence is performed.

The *Condition register* is not affected by these operations.

RF: Relative Forward Conditional Branch

0	1 0 1 0	CND	D	0
1	4	3	7	1

Syntax
[<ident>]␣RF[<cnd>]␣<m>

This instruction means that the next instruction to be executed is found either at the memory effective address or in normal sequence, depending on the contents of CND field and CR. If CND is equal to 111 the next instruction is at the memory effective address (unconditional relative branch). Effective address = Instruction counter + (8 bit even displacement value D).

Type	*Function*			*Execution Time*	
				P855	*P860*
T8	(P) + D	→ P	(Branch effective)	1.9	1.56
	(P) + 2	→ P	(No Branch)	1.9	1.56

RB: Relative Backwards Conditional Branch

0	1 0 1 1	CND	D	0
1	4	3	7	1

Syntax
[<ident>]␣RB[<cnd>]␣<m>

This instruction means that the next instruction to be executed is found either at the memory effective address or in normal sequence, depending on the contents of CND-field and CR. If CND is equal to 111 the next instruction is at the memory effective address (unconditional relative branch). Effective address = Instruction counter - (8-bit even displacement value D).

Type	*Function*			*Execution Time*	
				P855	*P860*
T8	(P) - D	→ P	(Branch effective)	1.9	1.56
	(P) + 2	→ P	(No Branch)	1.9	1.56

Remarks
It should be noted that
␣RB␣<cnd>,␣*
is equivalent to *branch to self* and causes a continuous loop.

Register/Register Instructions

LDS: Load register/register

1	0 0 0 0	R1	MD	R2	I/s
1	4	4	2	4	1

Syntax
[<ident>]␣LDR[*]␣<r1>,<r2>

The 16 bits of the register specified by R1 are replaced either by the contents of register specified by R2 (direct instruction) or by the contents of the memory address indicated in register specified by R2 (indirect instruction). In this last addressing mode if R2 specifies the A15 register, this one being considered as a stack pointer is updated.

Type	*Function*	*Mode (MD)*	*L/S*	*Execution Time*	
				P855	*P860*
T1	(R2) → R1	00	n.s.	1.9	1.56
T3A	((R2)) → R1	01	0	2.4	1.68
T3B	(A15)+2 → A15, ((A15)) → R1	01	0	3.1	2.4

Condition register
CR = 0 if (R1) = 0
1 if (R1) > 0
2 if (R1) < 0

STR: Store register/register

1	0 0 0 0	R1	01	R2	1
1	4	4	2	4	1

Syntax
[<ident>]␣STR␣<r1>,<r2>

The 16 bits of the register specified by R1 replace the contents of the memory address indicated in register specified by R2 (indirect instruction).
If R2 specifies the A15 register, this one being considered as a stack pointer, is updated.

Type	*Function*	*Execution Time*	
		P855	*P860*
T3A	(R1) → (R2)	2.4	1.68
T3B	(R1) → (A15), (A15) − 2 → A15	3.1	2.4

Condition register
CR = Unchanged.

Remarks
An interrupt 'Stack overflow' is generated when, for T3B type, the word address $128_{(10)}$ is reached by the pointer.

ADR: Addition register/register

1	0 0 1 0	R1	MD	R2	I/s
1	4	4	2	4	1

Syntax
[<ident>]␣ADR[s|*]␣<r1>,<r2>

The 16 bits of the register specified by R1 are added:
- either to the contents of the 16-bit register specified by R2 (direct instruction) in this case the sum is always placed in R1 register;
- or to the contents of the memory address indicated in the register specified by R2 (indirect instruction) in this case the sum is placed either in R1 or in memory address depending on the state of load or store indicator.

Type	*Function*	*Mode (MD)*	*L/S*	*Execution Time*	
				P855	*P860*
T1	(R1)+(R2) → R1	00	n.s.	1.9	1.56
T3	(R1)+((R2)) → R1	01	0	2.4	1.68
T3	(R1)+((R2)) → (R2)	01	1	3.6	2.52

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in case of overflow

Remarks
When L/S bit = 1, R1 must be $\neq 0$.

SUR: Subtract register/register

1	0 0 1 1	R1	MD	R2	I/s
1	4	4	2	4	1

Syntax
[<ident>]␣SUR[s|*]␣<r1>,<r2>

The contents of the register specified by R2 (direct instruction) or the contents of the memory address indicated in the Register specified by R2 (indirect instruction) are subtracted from the contents of the 16-bit register specified by R1. The result is placed:

- (direct instruction) in the R1 register
- (indirect instruction) either in the R1 register or in the memory address indicated in the register specified by R2, depending on the state of load or store indicator.

Type	*Function*			*Mode (MD)*	*L/S*	*Execution Time*	
						P855	*P860*
T1	(R1) - (R2)	→	R1	00	n.s.	1.9	1.56
T3	(R1) - ((R2))	→	R1	01	0	2.4	1.68
T3	(R1) - ((R2))	→	(R2)	01	1	3.6	2.52

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in case of overflow

Remarks
When L/S bit = 1, R1 must be ≠ 0.

ANR: Logical AND register/register

1	0 1 0 0	R1	MD	R2	I/s
1	4	4	2	4	1

Syntax
[<ident>]␣ANR[s|*]␣<r1>,<r2>

Logical product

Bit in R1	Bit in 2nd operand	Logical product
0	0	0
0	1	0
1	0	0
1	1	1

The logical product of the contents of the 16-bit register R1 and the contents of the 16-bit register specified by R2 (direct instruction) or the contents of the memory address indicated in the register specified by R2 (indirect address) is stored in:

- (direct instruction) R1 register
- (indirect instruction) either in the R1 register or in the memory address indicated in the register specified by R2, depending on the state of the load/store indicator.

Type	*Function*			*Mode (MD)*	*L/S*	*Execution Time*	
						P855	*P860*
T1	(R1) (R2)	→	R1	00	0	1.9	1.56
T3	(R1) ∧ ((R2))	→	R1	01	0	2.4	1.68
T3	(R1) ∧ ((R2))	→	(R2)	01	1	3.6	2.52

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0

ORR: Logical OR register/register

1	0 1 0 1	R1	MD	R2	I/s
1	4	4	2	4	1

Syntax
[<ident>]␣ORR[s|*]␣<r1>,<r2>

Bit in R1	Bit in 2nd operand	Logical OR
0	0	0
0	1	1
1	0	1
1	1	1

The logical OR of the contents of the 16-bit register specified by R1 and the contents of the 16-bit register specified by R2 (direct instruction) or the contents of the memory address indicated in the register specified by R2 (indirect instruction) is placed:

- (direct instruction) in R1 register
- (indirect instruction) either in the R1 register or in the memory address indicated in the register specified by R2 depending on the state of the load/store indicator.

Type	*Function*			*Mode (MD)*	*L/S*	*Execution Time*	
						P855	*P860*
T1	(R1) ∨ (R2)	→	R1	00	0	1.9	1.56
T3	(R1) ∨ ((R2))	→	R1	01	0	2.4	1.68
T3	(R1) ∨ ((R2))	→	(R2)	01	1	3.6	2.52

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0

XRR: Exclusive OR register/register

1	0 1 1 0	R1	MD	R2	I/s
1	4	4	2	4	1

Syntax
[<ident>]␣XRR[s|*]␣<r1>,<r2>

Bit in R1	Bit in 2nd operand	Exclusive OR
0	0	0
0	1	1
1	0	1
1	1	0

The exclusive OR of the contents of the 16-bit register specified by R1 and the contents of the 16-bit register specified by R2 (direct instruction) or the contents of the memory address indicated in the register specified by R2 (indirect instruction) is placed:

- (direct instruction) in the R1 register
- (indirect instruction) either in the R1 register or in the memory address indicated in the register specified by R2 depending on the state of the load/store indicator.

Type	*Function*			*Mode (MD)*	*L/S*	*Execution Time*	
						P855	*P860*
T1	(R1) ⊻ (R2)	→	R1	00	0	1.9	1.56
T3	(R1) ⊻ ((R2))	→	R1	01	0	2.4	1.68
T3	(R1) ⊻ ((R2))	→	(R2)	01	1	3.6	2.52

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0

IMR: Increment memory/register

1	0 0 1 0	0 0 0 0	01	R2	1
1	4	4	2	4	1

Syntax
[<ident>]␣IMR␣<r2>

The contents of the effective memory address indicated in the register specified by R2 (indirect) are increased by one.

Type	*Function*			*Execution Time*	
				P855	*P860*
T3	((R2)) + 1	→	(R2)	3.6	2.52

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in case of overflow

C2R: Two's complement

1	0 0 1 1	0 0 0 0	01	R2	1
1	4	4	2	4	1

Syntax
[<ident>]␣C2R␣<r2>

The two's complement of the contents of the memory effective address, indicated by the word following the instruction, replaces the old contents.

Type	*Function*			*Execution Time*	
				P855	*P860*
T3	0 - ((R2))	→	(R2)	3.6	2.52

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in case of overflow

MLR: Multiple load/register

1	0 1 1 1	n	01	R2	0
1	4	4	2	4	1

Syntax
[<ident>]␣MLR␣<n>,<r2>
<n>= 0<number of registers≤15

The contents of n consecutive registers (the first is register A1) are replaced by the contents of n consecutive memory locations, the address of the first location being indicated in R2. N may have any value between 1 and 15.
If R2 specifies A15, its contents are updated since it is considered to be the stack pointer.

Type	*Function*			*Execution Time*	
				P855	*P860*
T3A	((R2))	→	A1	(n + 1) × 1.2	(n + 1) × 0.84
	((R2) + 2)	→	A2		
	⋮				
	((R2) + 2n - 2)	→	An		
T3B	((A15) + 2n	→	A15	n × 1.2 + 1.9	n × 0.84 + 1.56
	((A15))	→	A1		
	((A15) - 2)	→	A2		
	⋮				
	((A15) - 2n + 2	→	An		

Condition register
CR = 0 if (A1) = 0
1 if (A1) > 0
2 if (A1) < 0

MSR: Multiple Store/register

1	0 1 1 1	n	01	R2	1
1	4	4	2	4	1

Syntax
[<ident>]␣MSR␣<n>,<r2>
<n>= 0<number of registers≤15

The contents of n consecutive registers (first one is A1) replace the contents of n consecutive memory locations, the address of the fist one being indicated by R2. N may have any value between 1 and 15.
If R2 specifies A15, its contents are updated since it is considered to be the stack pointer.

Type	*Function*			*Execution Time*	
				P855	*P860*
T3A	(A1)	→	(R2)	n×1.9+1.2	n×1.56+0.84
	(A2)	→	(R2)+2		
	⋮				
	(An)	→	(R2)+2n-2		
T3B	(A1)	→	(A15)	n×1.9+1.2	n×1.56+0.84
	(A2)	→	(A15)-2		
	⋮				
	(An)	→	(A15)-2n+2		
	(A15)-n	→	(A15)		

Condition register
Unchanged

Remark
An interrupt 'Stack Overflow' is generated when, for type T3B, the word address $128_{(10)}$ is reached by the pointer.

ABR: Absolute conditional branch to register

1	0 0 0 1	CND	▨	MD	R2	I/s
1	4	3	1	2	4	1

Syntax
[<ident>]␣ABR[<cnd>][*]␣<r2>

This instruction means that the next instruction to be executed is found either in the register specified by R2 or at the memory address indicated by the register or in normal sequence, depending on the contents of CND field and CR. If CND is equal to 111, the next instruction is at the memory effective address (unconditional branch).

CND contents	Meaning	CND contents	Meaning
000	= 0	100	≠ 0
001	= 1	101	≠ 1
010	= 2	110	≠ 2
011	= 3	111	unconditional

Type	*Function*		*Mode (MD)*	*L/S*	*Execution Time*	
	Branch Effective	*No Branch*			*P855*	*P860*
T1	(R2)→P	(P)+2→P	00	n.s.	1.9	1.56
T3	((R2))→P	(P)+2→P	01	0	2.4	1.68
			no branch		1.9	1.56

Condition register
Unchanged

CFR: Call function register

1	1 1 1 0	R1	MD	R2	1
1	4	4	2	4	1

Syntax
[<ident>]␣CFR[*]␣<r1>,<r2>

This instruction provides a link to a subroutine by storing successively the contents of the P-register and the program status word in a memory stack. The stack pointer specified by R1 is automatically updated. Then a branch is made to the direct or indirect address specified by R2.

Type	*Function*	*Mode (MD)*	*Execution Time*	
			P855	*P860*
T1, T3	(P) → (R1), (R1) - 2 → R1	n.a.		
	(PSW) → (R1), (R1) - 2 → R1	n.a.		
then:				
T1	(R2) → P	00	5.76	4.68
T3	((R2)) → P	01	6.24	4.80

Condition register
Unchanged

Remark
If register A15 is used to contain the stack pointer, a stack overflow interrupt will be generated when the word address $128_{(10)}$ is reached by the pointer.

C1R: One's complement register/register

1	1 1 1 1	R1	MD	R2	I/s
1	4	4	2	4	1

Syntax:
[<ident>]␣C1R[S][⋆]␣[<r1>],<r2>

The logic complement of the contents of the 16-bit register specified by R2 (direct instruction) or the contents of the memory address indicated in the register specified by R2 replaces the contents of:

- (direct instruction) the register specified by R1
- (indirect instruction) either the register specified by R1 or the memory address indicated in the register specified by R2, depending on the state of the load/store indicator.

Type	*Function*	*Mode (MD)*	*L/S*	*Execution Time*	
				P855	*P860*
T1	$(\overline{R2})$ → R1	00	n.s.	1.9	1.56
T3	$(\overline{(R2)})$ → R1	01	0	2.4	1.68
T3	$(\overline{(R2)})$ → (R2)	01	1	3.6	2.52

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0

Remarks
When L/S bit = 0, R1 must be ≠ 0.

CWR: Compare word register/register

1	1 1 0 1	R1	MD	R2	I/s
1	4	4	2	4	1

Syntax
[<ident>]␣CWR[*]␣<r1>,<r2>

The contents of the 16-bit register specified by R1 are compared with the contents of the 16-bit register specified by R2 (direct instruction) or with the contents of the memory address indicated in the register specified by R2 (indirect instruction). The result is stored in the condition register.

Type	*Function*			*Mode (MD)*	*L/S*	*Execution Time*	
						P855	*P860*
T1	(R1) ÷ (R2)	→	CR	00	n.s.	1.9	1.56
T3	(R1) ÷ ((R2))	→	CR	01	0	2.4	1.68

Condition register
CR = 0 if (R1) = (2nd operand)
1 if (R1) > (2nd operand)
2 if (R1) < (2nd operand)

CCR: Compare character

1	1 1 0 1	R1	01	R2	1
1	4	4	2	4	1

Syntax
[<ident>]␣CCR␣<r1>,<r2>

The least significant bits of the register specified by R1 are compared with the right (C = 1) or left (C = 0) 8 bits of the memory address indicated in R2.
The most significant bit of a character is not considered as a sign bit.
The result of the operation is stored in the condition register.

Type	*Function*			*Execution Time*	
				P855	*P860*
T3	(R1)r ÷ ((R2))l/r	→	CR	3.6	2.52

Condition register
CR = 0 if (R1)r = ((R2))l/r
1 if (R1)r > ((R2))l/r
2 if (R1)r < ((R2))l/r

LCR: Load character/register

1	1 1 0 0	R1	01	R2	0
1	4	4	2	4	1

Syntax
[<ident>]␣LCR␣<r1>,<r2>

The right (C = 1) or left (C = 0) 8-bit contents of the memory address specified by R2 take place of the least significant 8 bits of the register specified by R1.

Type	*Function*			*Execution Time*	
				P855	*P860*
T3	((R2))l/r	→	R1r	3.6	2.52

Condition register
Unchanged.

SCR: Store character/register

1	1 1 0 0	R1	01	R2	1
1	4	4	2	4	1

Syntax
[<ident>[␣SCR␣<r1>,<r2>

The least significant 8 bits of the register specified by R1 replace the right (C = 1) or left (C = 0) 8 bits contents of the memory effective address indicated by the R2-register.

Type	*Function*			*Execution Time*	
				P855	*P860*
T3	(R1)r	→	(R2)r/l	3.6	2.52

Condition register
Unchanged.

RTN: Return from function

1	1 1 1 0	0 0 0 0	01	R2	0
1	4	4	2	4	1

Syntax
[<ident>]␣RTN␣<r2>

This instruction allows a program to be resumed after the execution of an interrupt routine or a subroutine, by reloading the condition register and the P-register from a memory stack. The stack pointer specified by R2 is automatically updated.

Type	*Function*			*Execution Time*	
				P855	*P860*
T3	(R2)+4	→	R2		
	((R2))	→	P		
	((R2)-2)	→	PSW	4.3	3.24

Condition register
Reloaded from stack, PSW (6, 7) →CR.

Remark
R2 must be ≠ 0.

ECR: Exchange character register/register

1	1 1 0 0	R1	0 0	R2	
1	4	4	2	4	1

Syntax
[<ident>]␣ECR␣<r1>,<r2>

The left and right characters contained in the R2-register are exchanged and then located into the R1-register.

Type	*Function*		*Execution Time*	
		L/S	*P855*	*P860*
T1	(R2)l → R1r and (R2)r → R1left	n.s.	1.9	1.56

Condition register
Unchanged.

MUR: Multiply registers/registers (optional)

1	1 0 0 0	0 0 0 0	MD	R2	0
1	4	4	2	4	1

Syntax

[<ident>]␣MUR[⋆]␣<r2>

The contents of the register specified by R2 (direct instruction), or the contents of the **memory address** indicated in R2 (indirect instruction) are multiplied by the contents of A2. The result is loaded as a 30-bit product in A1-A2. The most significant bit of register A2 is reset to zero. The sign of the product is stored in the sign bit of register A1.
Overflow occurs if result $> 2^{30}-1$, or result $\leqslant -2^{30}$; in that case the two registers contain only the 30 least significant bits while the sign bit may or may not be correct.

Type	*Function*			*Mode (MD)*	*Execution Time*	
					P855	*P860*
T1	(A2)×(R2)	→	A1,A2	00	8.04	7.68
T3	(A2)×((R2))	→	A1,A2	01	8.52	7.80

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in case of overflow

DVR: Divide registers/registers (optional)

1	1 0 0 1	0 0 0 0	MD	R2	0
1	4	4	2	4	1

Syntax
[<ident>]␣DVR[⋆]␣<r2>

The contents of registers A1 and A2 are divided by the contents of R2 (direct instruction) or the contents of the **memory address** indicated in R2 (indirect instruction). The quotient is placed in A2, the remainder in A1.
Overflow occurs if the quotient exceeds 15 bits: in that case the contents of A1 and A2 are not significant.

Type	*Function*			*Mode (MD)*	*Execution Time*	
		quotient	*remainder*		*P855*	*P860*
T1	(A1, A2)/(R2)	→ A2	A1	00	9.48	9.12
T3	(A1, A2)/((R2))	→ A2	A1	01	9.96	9.24

Condition register
CR = 0 if A2 = 0
1 if A2 > 1
2 if A2 < 0
3 in case of overflow

DAR: Double add registers/registers (optional)

1	1 0 1 0	0 0 0 0	MD	R2	0
1	4	4	2	4	1

Syntax
[<ident>]␣DAR[⋆]␣<mode>[,<r2>]

The contents of the registers specified by R2 and R2 + 1 (direct instruction) or the contents of the two consecutive memory words, the address of the first one being indicated in R2 (indirect instruction) are added to the contents of A1 and A2. The sign bit of A2 is never used; the sign bit of the result is the sign bit of A1.

Type	*Function*		*Mode (MD)*	*Execution Time*	
				P855	*P860*
T1	(R2, R2 + 1) + (A1, A2)	→ A1, A2	00	3.4	3.00
T3	((R2), (R2 + 1)) + (A1, A2)	→ A1, A2	01	4.3	3.24

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in case of overflow

DSR: Double subtract registers/registers (optional)

1	1 0 1 1	0 0 0 0	MD	R2	0
1	4	4	2	4	1

Syntax
[<ident>]␣DSR[⋆]␣<mode>[,<r2>]

The contents of the registers specified by R2 and R2 + 1 (direct instruction) or the contents of the two consecutive memory words, the address of the first one being indicated in R2 (indirect instruction) are subtracted from the contents of A1 and A2. The sign bit of A2 is never used.

Type	*Function*			*Mode (MD)*	*Execution Time*	
					P855	*P860*
T1	(A1, A2) − (R2, R2 + 1)	→	A1, A2	00	3.4	3.00
T3	(A1, A2) − ((R2), (R2 + 1))	→	A1, A2	01	4.3	3.24

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in case of overflow

TM: Test mask (optional)

1	0 1 0 0	R1	0 0	R2	1
1	4	4	2	4	1

Syntax
[<ident>]␣TM␣<r1>,<r2>

The logical product of the contents of the register specified in R1 and the contents of the register specified in R2 is compared to zero. The result of the comparison is stored in the condition register. The initial contents of R1 and R2 remain unchanged.

Type	*Function*	*Execution Time*	
		P855	*P860*
T1	[(R1) ∧ (R2)] ÷ 0 → CR	1.92	1.56

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0

TNM: Test not mask

1	0 1 1 0	R1	0 0	R2	1
1	4	4	2	4	1

Syntax
[<ident>]␣TNM␣<r1>,<r2>

The exclusive OR of the contents of the register specified in R1 and the contents of the register specified in R2 is compared to zero. The result of the comparison is stored in the condition register.
The initial contents of by R1 and R2 specified registers remain unchanged.

Type	*Function*	*Execution Time*	
		P855	*P860*
T1	[(R1) ⩛ (R2)] ÷ 0 → CR	1.92	1.56

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0

Constant Instructions

CONSTANT INSTRUCTIONS

In this group of instructions, the data item to be operated upon is either contained within the instruction itself (short format: bit 8 - 15), or in the word following the instruction (long format). In short format, the 8-bit constant is always positive, because the most significant bit position of the constant is not recognized as a sign bit by the logic. A negative constant can only be specified in long format, the word following the instruction being used to generate the constant.

Only the registers A0 to A7 may be specified in the R3-field

Format T8 (short)

0	OP code	R3	positive constant
1	4	3	8

Format T2 (long)

1	OP code	R1	MD	R2	I/s
1	4	4	2	4	1

*)

positive or negative constant
16

*) This second word will not be shown for each separate instruction on the following pages.

LDK: Load constant

T8

0	0 0 0 0	R3	K
1	4	3	8

T2

1	0 0 0 0	R1	0 1	0 0 0 0	0
1	4	4	2	4	1

Syntax

T8 [<ident>]␣LDK␣<r3>,<k>

T2 [<ident>]␣LDKL␣<r1>,<k1>

T8 K is loaded into bits 8 to 15 of the register specified in R3; the most significant bits of R3 are reset to zero.

T2 KL replaces the contents of the register specified in R1.

Type	*Function*	*Execution Time*	
		P855	*P860*
T8	$K \rightarrow R3_{8-15}, 0 \rightarrow R3_{0-7}$	1.9	1.56
T2	$KL \rightarrow R1$	2.4	1.68

Condition register

T8 Unchanged

T2 CR = 0 if KL = 0
1 if KL > 0
2 if KL < 0

ADK: Add constant

T8

1	0 0 1 0	R3	K
1	4	3	8

T2

1	0 0 1 0	R1	0 1	0 0 0 0	0
1	4	4	2	4	1

Syntax
T8 [<ident>]␣ADK␣<r3>,<k>
T2 [<ident>]␣ADKL␣<r1>,<k1>

T8 K is added to the contents of the register specified in R3; the result is placed in R3.

T2 The contents of the register specified in R1 are added to KL; the result is placed in R1.

Type	*Function*	*Execution Time*	
		P855	*P860*
T8	(R3) + K → R3	1.9	1.56
T2	(R1) + KL → R1	2.4	1.68

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in case of overflow

SUK: Subtract constant

T8

0	0 0 1 1	R3	K
1	4	3	8

T2

1	0 0 1 1	R1	0 1	0 0 0 0	0
1	4	4	2	4	1

Syntax
T8 [<ident>]␣SUK␣<r3>,<k>
T2 [<ident>]␣SUKL␣<r1>,<k1>

T8 K is subtracted from the contents of the register specified in R3, the result is placed in R3.

T2 KL is subtracted from the contents of the 16-bit register specified in R1; the result is placed in R1.

Type	*Function*	*Execution Time*	
		P855	*P860*
T8	(R3) - K → R3	1.9	1.56
T2	(R1) - KL → R1	2.4	1.68

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in case of overflow

ANK: Logical AND with constant

T8

0	0 1 0 0	R3	K
1	4	3	8

T2

1	0 1 0 0	R1	0 1	0 0 0 0	0
1	4	4	2	4	1

Syntax

T8 [<ident>]␣ANK␣<r3>,<k>

T2 [<ident>]␣ANKL␣<r1>,<kl>

T8 The logical product of K and the contents of bits 8 - 15 of the register specified in R3 is placed in R3, bits 0 - 7 being reset to zero (See page 2 for explanation of the AND-operation).

T2 The logical product of the contents of the register specified in R1 and KL is placed in R1.

Type	*Function*	*Execution Time*	
		P855	*P860*
T8	$(R3)_{8-15} \wedge K \rightarrow R3_{8-15}$	1.9	1.56
T2	$(R1) \wedge KL \rightarrow R1$	2.4	1.68

Condition register

CR = 0 if result = 0
1 if result > 0
2 if result < 0

ORK: Logical OR with constant

T8

0	0 1 0 1	R3	K
1	4	3	8

T2

1	0 1 0 1	R1	0 1	0 0 0 0	0
1	4	4	2	4	1

Syntax

T8 [<ident>]␣ORK␣<r3>,<k>

T2 [<ident>]␣ORKL␣<r1>,<kl>

T8 A logical OR is performed on the contents of bits 8 to 15 of the register specified in R3 and K.
The result is placed in bits 8 - 15 of R3, bits 0 - 7 being reset to zero (See page 3 for explanation of OR).

T2 The logical OR of the contents of the register specified in R1 and KL is placed in R1.

Type	*Function*	*Execution Time*	
		P855	*P860*
T8	$(R3)_{8-15} \vee K \rightarrow R3_{8-15}$	1.9	1.56
T2	$(R1) \vee KL \rightarrow R1$	2.4	1.68

Condition register

CR =0 if result = 0
1 if result > 0
2 if result < 0

XRK: Exclusive OR with constant

T8

0	0 1 1 0	R3	K
1	4	3	8

T2

1	0 1 1 0	R1	0 1	0 0 0 0	0
1	4	4	2	4	1

Syntax
T8 [<ident>]␣XRK␣<r3>,<k>
T2 [<ident>]␣XRKL␣<r1>,<kl>

T8 An exclusive OR operation (See page 3 for explanation) is performed on the contents of bits 8 - 15 of the register specified in R3 and K. The result is placed in R3. The most significant 8 bits of which remain unchanged.

T2 The exclusive OR of the contents of the register specified in R1 and KL is placed in R1.

Type	*Function*	*Execution Time*	
		P855	*P860*
T8	$(R3)_{8-15} \veebar K \rightarrow R3_{8-15}$	1.9	1.56
T2	$(R1) \veebar KL$	2.4	1.68

Condition register
CR= 0 if result = 0
1 if result > 0
2 if result < 0

MLK: Multiple load constant

1	0 1 1 1	n	01	0 0 0 0	0
1	4	4	2	4	1

Syntax
[<ident>]␣MLK␣<n>
<n> = number of registers 0 < number ⩽ 15

KL1, KL2 --- KLn replace the contents of registers A1, A2 --- An. n may have any value between 1 and 15.

Type	*Function*	*Execution Time*
T2	KL1, KL2, ... KLn → A1, A2 ... An	*P855:* (n+1)×1.2 *P860:* (n+1)×0.84

Condition register
CR = 0 if (A1) = 0
1 if (A1) > 0
2 if (A1) < 0

AB: Absolute conditional branch

T8

0	0 0 0 1	CND	K	
1	4	3	7	1

T2 *)

1	0 0 0 1	CND		0 1	0 0 0 0	0
1	4	3	1	2	4	1

Syntax
T8 [<ident>]␣AB[<cnd>]␣<k>
T2 [<ident>]␣ABL[<cnd>␣<k1>

This instruction means that the next instruction to be executed is found either at the address specified by the constant (K indicating one of the first 128 locations, KL being specified in the word following the constant), or in normal sequence, depending on the contents of CND and CR.
If CND is equal to 111_2, the next instruction is at the memory effective address.

Type	*Function*		*Execution Time*	
	Branch Effective	*No Branch*	*P855*	*P860*
T8	K → P	(P)+2 → P	1.9	1.56
	(p) → P		2.4	1.68
T2	KL → P	(P)+2 → P	*No Branch:* 1.9	1.56

Condition register
unchanged

*) Bit 15 of 2nd word not used

CF: Call function

1	1 1 1 0	R1	01	0 0 0 0	1
1	4	4	2	4	1

Syntax
[<ident>]␣CF␣<r1>,<k1>

This instruction provides a link to a subroutine by storing successively the contents of the P register and the program status word in a memory stack. The stack pointer is specified by the contents of R1 and is automatically updated. Then, a branch is made to the address specified by KL.

Type	*Function*		*Execution Time*	
			P855	*P860*
T2	(P) → R1,	(R1) − 2 → R1	6.24	4.80
	(PSW) → R1,	(R1) − 2 → R1		
	KL → P			

Condition register
Unchanged

Remark
If register 15 (R1=1111) is used to contain the stack pointer, a 'stack overflow' interrupt will be generated when the word address 128_{10} is reached by the pointer.

*) Bit 15 of 2nd word not used.

CWK: Compare word with constant

1	1 1 0 1	R1	0 1	R2	0
1	4	4	2	4	1

Syntax
[<ident>]␣CWK␣<r1>,<kl>

The contents of the register specified in R1 are compared with KL. The result is stored in the condition register.

Type	*Function*	*Execution Time*	
		P855	*P860*
T2	(R1) ÷ KL → CR	1.9	1.56

Condition register
CR = 0 if (R1) = KL
1 if (R1) > L
2 if (R1) < KL

CCK: Compare character

1	1 1 0 1	R1	01	0 0 0 0	1
1	4	4	2	4	1

Syntax
[<ident>]␣CCK␣<r1>,<kl>

Bits 8-15 of the register specified by R1 are compared with bits 0-7 of KL. The most significant bit of a character is not considered as a sign bit. The result of the comparison is stored in the condition register.

Type	*Function*	*Execution Time*	
		P855	*P860*
T2	(R1)r ÷ KL1 → CR	1.9	1.56

Condition register
CR = 0 if (R1)r = KL left
1 if (R1)r > KL left
2 if (R1)r < KL left

LCK: Load character

1	1 1 0 0	R1	01	0 0 0 0	0
1	4	4	2	4	1

Syntax
[<ident>]␣LCK␣<r1>,<kl>

Bits 0-7 of KL are placed in bits 8-15 of the register specified in R1.

Type	*Function*	*Execution Time*	
		P855	*P860*
T2	KL1 → R1right	1.9	1.56

Condition register
Unchanged

MUK: Multiply with constant (optional)

1	1 0 0 0	0 0 0 0	01	0 0 0 0	0
1	4	4	2	4	1

Syntax
[<ident>]␣MUK␣<kl>

KL is multiplied by the contents of A2. The result is loaded as a 30-bit product in A1 and A2. The most significant bit of A2 is reset to zero. Overflow occurs if result $>2^{30} - 1$, or result $\leqslant -2^{30}$, in that case the two registers contain only the 30 least significant bits while the sign bit may or may not be correct.

Type	*Function*	*Execution Time*	
		P855	*P860*
T2	(A2)×KL → A1, A2	8.52	7.80

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in case of overflow

DVK: Divide by constant (optional)

1	1 0 0 1	0 0 0 0	01	0 0 0 0	0
1	4	4	2	4	1

Syntax
[<ident>]␣DVK␣<k1>

The contents of A1,A2 are divided by KL. The quotient is placed in A2, the remainder in A1. Overflow occurs when the quotient exceeds 15 bits, in that case the contents of A1 and A2 are not significant.

Type	*Function*		*Execution Time*	
	quotient	*remainder*	*P855*	*P860*
T2	(A1,A2)/KL → A2	A1	9.96	9.24

Condition register
CR = 0 if (A2) = 0
1 if (A2) > 0
2 if (A2) < 0
3 in case of overflow

DAK: Double add with constant (optional)

1	1 0 1 0	0 0 0 0	01	0 0 0 0	0
1	4	4	2	4	1

Syntax
[<ident>]␣DAK␣<kl>,<kl>

KL(32 bits) is added to the contents of A1,A2. The sum is placed in A1,A2.

Type	*Function*	*Execution Time*	
		P855	*P860*
T2	KL+(A1, A2) → A1, A2	4.3	3.24

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in case of overflow

DSK: Double subtract with constant (optional)

1	1 0 1 1	0 0 0 0	01	0 0 0 0	0
1	4	4	2	4	1

Syntax
[<ident>]␣DSK␣<kl>,<kl>

KL (32 bits) is subtracted from the contents of A1,A2. The result is placed in A1,A2.

Type	*Function*	*Execution Time*	
		P855	*P860*
T2	(A1, A2) − KL → A1, A2	4.3	3.24

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in case of overflow

Shift Instructions

SHIFT INSTRUCTIONS

SLA: Single left arithmetic shift

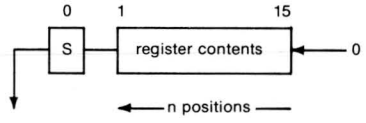
0	0 1 1 1	R3	0 0 0	n
1	4	3	3	5

Syntax
[<ident>]␣SLA␣<r3>,<n>
<n> = number of positions

The contents of the register specified by R3 are shifted left n positions. Overflow occurs when the sign bit was modified during the operation.

Type *Function* *Execution Time*

T8



P855: n×0.36+2.28
P860: n×0.36+1.92

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in case of overflow

Remark
R3 ≠ 0

SLA1: Single left arithmetic shift

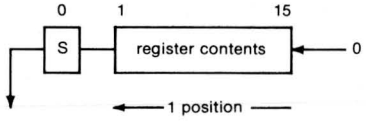
0	0 1 1 1	R3	0 0 0	0 0 0 0 1
1	4	3	3	5

Syntax
[<ident>]␣SLA1␣<r3>

The contents of the register specified by R3 are shifted left 1 position. Overflow occurs when the sign bit was modified during the operation.

Type *Function* *Execution Time*

T8



P855: n×0.36+2.28
P860: n×0.36+1.92

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in case of overflow

Remark
R3 ≠ 0

SRA: Single right arithmetic shift

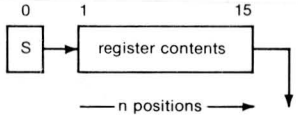
0	0 1 1 1	R3	0 0 1	n
1	4	3	3	5

Syntax
[<ident>]␣SRA␣<r3>,<n>

The contents of the register specified by R3 are shifted right n positions. The sign bit does not change, it is shifted into the vacated positions of the register.
If n ⩾ 15, all bits of the register will be the same as the sign bit.

Type *Function* *Execution Time*

T8



P855: n×0.36+2.28
P860: n×0.36+1.92

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0

Remark
R3 ≠ 0

SRA1: Single right arithmetic shift

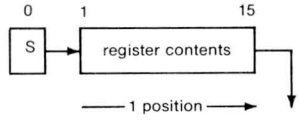
0	0 1 1 1	R3	0 0 1	0 0 0 0 1
1	4	3	3	5

Syntax
[<ident>]␣SRA1␣<r3>

The contents of the register specified in R3 are shifted right 1 position. The sign bit does not change, it is shifted into the vacated position of the register.

Type *Function* *Execution Time*

T8



P855: n×0.36+2.28
P860: n×0.36+1.92

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0

Remark
R3 ≠ 0

SLL: Single left logical shift

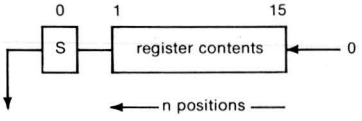
0	0 1 1 1	R3	0 1 0	n
1	4	3	3	5

Syntax
[<ident>]␣SLL␣<r3>,<n>

The contents of the register specified by R3 are shifted left n positions. Vacated bit positions become zero.
After 16 or more shifts the register contains zero.

Type *Function* *Execution Time*

T8



P855: n×0.36+2.28
P860: n×0.36+1.92

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0

Remark
R3 ≠ 0

SLL1: Single left logical shift

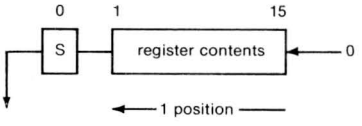
0	0 1 1 1	R3	0 1 0	0 0 0 0 1
1	4	3	3	5

Syntax
[<ident>]␣SLL1␣<r3>

The contents of the register specified in R3 are shifted left 1 position. Zero fills in the vacated bit position.

Type *Function* *Execution Time*

T8



P855: n×0.36+2.28
P860: n×0.36+1.92

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0

Remark
R3 ≠ 0

SHIFT INSTRUCTIONS

SLA: Single left arithmetic shift

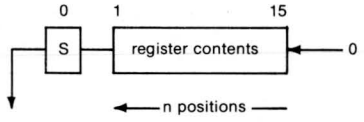
0	0 1 1 1	R3	0 0 0	n
1	4	3	3	5

Syntax
[<ident>]␣SLA␣<r3>,<n>
<n> = number of positions

The contents of the register specified by R3 are shifted left n positions. Overflow occurs when the sign bit was modified during the operation.

Type *Function* *Execution Time*

T8



P855: n×0.36+2.28
P860: n×0.36+1.92

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in case of overflow

Remark
R3 ≠ 0

SLA1: Single left arithmetic shift

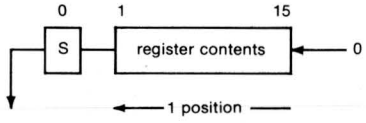
0	0 1 1 1	R3	0 0 0	0 0 0 0 1
1	4	3	3	5

Syntax
[<ident>]␣SLA1␣<r3>

The contents of the register specified by R3 are shifted left 1 position. Overflow occurs when the sign bit was modified during the operation.

Type *Function* *Execution Time*

T8



P855: n×0.36+2.28
P860: n×0.36+1.92

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in case of overflow

Remark
R3 ≠ 0

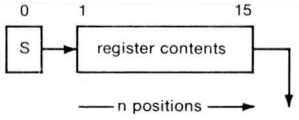
SRA: Single right arithmetic shift

0	0 1 1 1	R3	0 0 1	n
1	4	3	3	5

Syntax
[<ident>]␣SRA␣<r3>,<n>

The contents of the register specified by R3 are shifted right n positions. The sign bit does not change, it is shifted into the vacated positions of the register.
If n ⩾ 15, all bits of the register will be the same as the sign bit.

Type *Function* *Execution Time*

T8 

P855: n×0.36+2.28
P860: n×0.36+1.92

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0

Remark
R3 ≠ 0

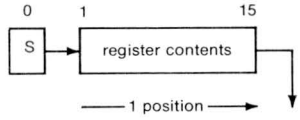
SRA1: Single right arithmetic shift

0	0 1 1 1	R3	0 0 1	0 0 0 0 1
1	4	3	3	5

Syntax
[<ident>]␣SRA1␣<r3>

The contents of the register specified in R3 are shifted right 1 position. The sign bit does not change, it is shifted into the vacated position of the register.

Type *Function* *Execution Time*

T8 

P855: n×0.36+2.28
P860: n×0.36+1.92

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0

Remark
R3 ≠ 0

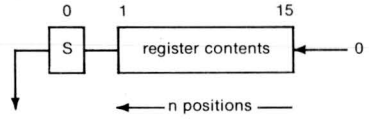
SLL: Single left logical shift

0	0 1 1 1	R3	0 1 0	n
1	4	3	3	5

Syntax
[<ident>]␣SLL␣<r3>,<n>

The contents of the register specified by R3 are shifted left n positions. Vacated bit positions become zero.
After 16 or more shifts the register contains zero.

Type *Function* *Execution Time*

T8 

P855: n×0.36+2.28
P860: n×0.36+1.92

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0

Remark
R3 ≠ 0

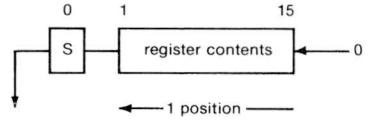
SLL1: Single left logical shift

0	0 1 1 1	R3	0 1 0	0 0 0 0 1
1	4	3	3	5

Syntax
[<ident>]␣SLL1␣<r3>

The contents of the register specified in R3 are shifted left 1 position. Zero fills in the vacated bit position.

Type *Function* *Execution Time*

T8 

P855: n×0.36+2.28
P860: n×0.36+1.92

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0

Remark
R3 ≠ 0

SRL: Single right logical shift

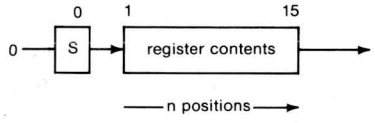
0	0 1 1 1	R3	0 1 1	n
1	4	3	3	5

Syntax
[<ident>]␣SRL␣<r3>,<n>

The contents of the register specified in R3 are shifted right n positions. Vacated bit positions become zero.
After 16 or more shifts the register contains zero.

Type *Function* *Execution Time*

T8



P855: n×0.36+2.28
P860: n×0.36+1.92

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0

Remark
R3 ≠ 0

SRC: Single right circular shift

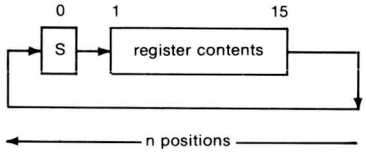
0	0 1 1 1	R3	1 1 1	n
1	4	3	3	5

Syntax
[<ident>]␣SRC␣<r3>,<n>

The contents of the register specified in R3 are shifted right, end around, n positions.

Type *Function* *Execution Time*

T8



P855: n×0.36+2.28
P860: n×0.36+1.92

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0

Remark
R3 ≠ 0

SLC: Single left circular shift

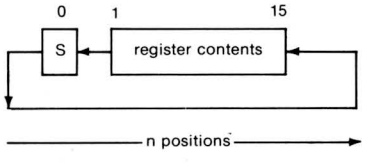
0	0 1 1 1	R3	1 1 0	n
1	4	3	3	5

Syntax
[<ident>]␣SLC␣<r3>,<n>

The contents of the register specified in R3 are shifted left, end around, n positions.

Type *Function* *Execution Time*

T8



P855: n×0.36+2.28
P860: n×0.36+1.92

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0

Remark
R3 ≠ 0

SRC1: Single right circular shift

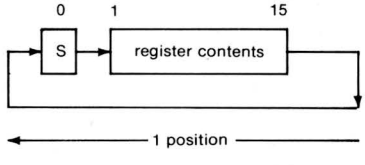
0	0 1 1 1	R3	1 1 1	0 0 0 0 1
1	4	3	3	5

Syntax
[<ident>]␣SRC1␣<r3>

The contents of the register specified in R3 are shifted right, end around, 1 position.

Type *Function* *Execution Time*

T8



P855: n×0.36+2.28
P860: n×0.36+1.92

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0

Remark
R3 ≠ 0

SRN: Single right and normalize shift

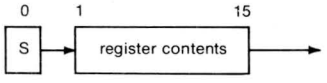
0	0 1 1 1	R3	1 0 1	R2	
1	4	3	3	4	1

Syntax
[<ident>]␣SRN␣<r3>,<r2>

The contents of the register specified in R3 are shifted right until a 1-bit appears in the rightmost position of the register.
The sign bit does not change and is copied to the right once for each shift position.
The number of shift positions is placed in the register specified in R2.

Type *Function* *Execution Time*

T8



P855: n×0.36+2.28
P860: n×0.36+1.92

Condition register
Unchanged.

Remarks
R3 ≠ 0
If the contents of R3 are initially zero, the number of shifted positions is 16.

SLN: Single left and normalize shift

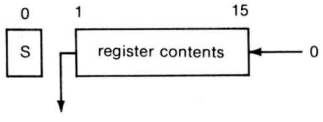
0	0 1 1 1	R3	1 0 0	R2	
1	4	3	3	4	1

Syntax
[<ident>]␣SLN␣<r3>,<r2>

The contents of the register specified in R3 are shifted left until the two most significant bits differ. The sign bit remains unchanged, zero bits are inserted in the least significant positions. The number of shifted positions is stored in the register specified in R2.

Type *Function* *Execution Time*

T8



P855: n×0.36+2.28
P860: n×0.36+1.92

Condition register
Unchanged.

Remarks
R3 ≠ 0.
If the contents of R3 are initially zero, the number of shifted positions is 16.

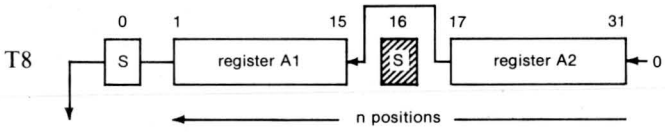
DLA: Double left arithmetic shift

0	0 1 1 1	0 0 0	0 0 0	n
1	4	3	3	5

Syntax
[<ident>]␣DLA␣<n>

The A1 and A2 registers are treated as one 31-bit register, its contents shifted left n positions. The sign bit of A2 is unchanged, while zeros are shifted into vacated positions of register A2. Overflow occurs when the sign bit was changed during execution of the instruction.

Type *Function*



Execution Time

P855: n×0.36+2.28
P860: n×0.36+1.92

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0
3 in case of overflow

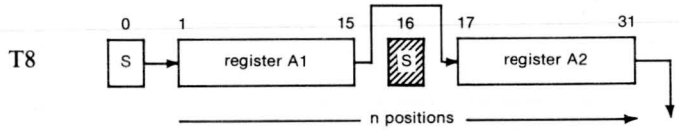
DRA: Double right arithmetic shift

0	0 1 1 1	0 0 0	0 0 1	n
1	4	3	3	5

Syntax
[<ident>]␣DRA␣<n>

The A1 and A2 registers are treated as one 31-bit register, its contents shifted right n positions. The sign bit of A2 is unchanged.
The sign bit of A1 is also unchanged, but propagated into vacated positions of the register. After 30 or more shifts the register is filled with the sign bit of A1, except the sign bit of A2, which is unchanged.

Type *Function*



Execution Time

P855: n×0.36+2.28
P860: n×0.36+1.92

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0

DLL: Double left logical shift

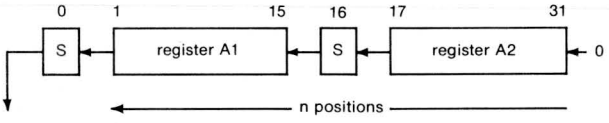
0	0 1 1 1	0 0 0	0 1 0	n
1	4	3	3	5

Syntax
[<ident>]␣DDL␣<n>

The A1 and A2 registers are treated as one 32-bit register, its contents shifted left n positions. Zeros are shifted into vacated positions.
The maximum number of shifts is 31.

Type *Function*

T8



Execution
Time

P855: n×0.36+2.28
P860: n×0.36+1.92

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0

DRL: Double right logical shift

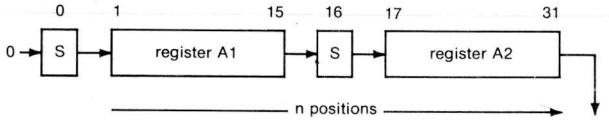
0	0 1 1 1	0 0 0	0 1 1	n
1	4	3	3	5

Syntax
[<ident>]␣DRL␣<n>

The A1 and A2 registers are treated as one 32-bit register, its contents shifted right n positions. Zeros are shifted into vacated positions.
The maximum number of shifts is 31.

Type *Function*

T8



Execution
Time

P855: n×0.36+2.28
P860: n×0.36+1.92

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0

DLC: Double left circular shift

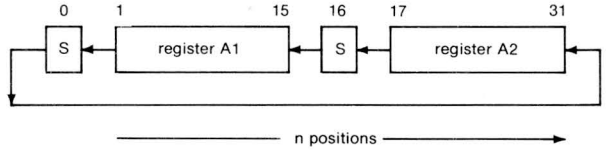
0	0 1 1 1	0 0 0	1 1 0	n
1	4	3	3	5

Syntax
[<ident>]␣DLC␣<n>

The A1 and A2 registers are treated as one 32-bit register, its contents shifhted left, end around, n positions.

Type *Function*

T8



Execution
Time

P855: n×0.36+2.28
P860: n×0.36+1.92

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0

DRC: Double right circular shift

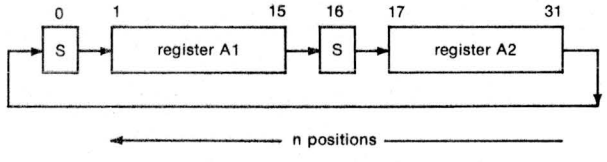
0	0 1 1 1	0 0 0	1 1 1	n
1	4	3	3	5

Syntax
[<ident>]␣DRC␣<n>

The A1 and A2 registers are treated as one 32-bit register, its contents shifted right, end around, n positions.

Type *Function*

T8



Execution Time

P855: n×0.36+2.28
P860: n×0.36+1.92

Condition register
CR = 0 if result = 0
1 if result > 0
2 if result < 0

DLN: Double left and normalize shift

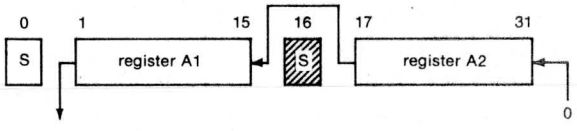
0	0 1 1 1	0 0 0	1 0 0	R2	
1	4	3	3	4	1

Syntax
[<ident>]␣DLN␣<r2>

The A1 and A2 registers are treated as one 31-bit register, its contents shifted left until two high-order bits are different. The sign bit remains unchanged. Zeroes are shifted into vacated positions of the register. The number of shifted positions is stored in the register specified in R2.

Type *Function*

T8



Execution Time

P855: n×0.36+2.28
P860: n×0.36+1.92

Condition register
Unchanged.

Remark
If the contents of A1-A2 are initially zero, the number of shifted positions is 31.

DRN: Double right and normalize shift

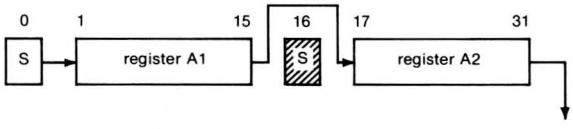
0	0 1 1 1	0 0 0	1 0 1	R2	
1	4	3	3	4	1

Syntax
[<ident>]␣DRN␣<r2>

The A1 and A2 registers are treated as one 31-bit register, its contents shifted right until a 1-bit appears in the least significant position of the register. The sign bit remains unchanged, it is propagated to the right for each shift position.
The total number of shift positions is stored in the register specified in R2.

Type *Function*

T8



Execution Time

P855: n×0.36+2.28
P860: n×0.36+1.92

Condition register
Unchanged.

Remark
If the contents of A1-A2 are initially zero, the number of shifted positions is 31.

Input/Output Instructions

CIO: Control Input/Output

0	1 0 0 0	R3	1	F	dev.
1	4	3	1	1	6

Syntax
[<ident>]␣CIO␣<r3>,[0|1],<dev>

This I/O instruction has the following functions:
- Start an I/O operation on a peripheral device (F = 1)
- Stop a data transfer or reset the state of a peripheral device control unit.

During execution <dev> field is sent, via the I/O bus, to the peripheral device control unit. The contents of the 16-bit register specified by R3 are also sent on the I/O bus to provide further information for some sophisticated control units. A Start I/O command is not accepted if the device address is unknown or if the corresponding device is busy. Halt I/O is always accepted.

Type	*Execution Time*	
	P855	*P860*
T8	3.6	2.52

Condition register
CR = 0 command accepted
1 command not accepted
3 device address unknown

Remark
R3 ≠ 0

INR: Input to register

0	1 0 0 1	R3	0	F	dev.
1	4	3	1	1	6

Syntax
[<ident>]␣INR␣<r3>,[0|1],<dev>

If the I/O instruction is accepted a data word or character is transferred from the device to the register specified by R3. According to the type of the device the lower bits (right placed) or the 16 bits are significant. In the first case the higher bits of R3 are reset to zero. During execution 'F' and <dev> fields are sent to the device via the I/O bus. F bit may be used to specify a particular input function. This I/O instruction is not accepted if the device control unit is not in exchange state. After a not accepted INR instruction the contents of the R3-register are not significant.

Type	*Execution Time*	
	P855	*P860*
T8	3.6	2.52

Condition register
CR = 0 command accepted
1 command not accepted
3 device address unknown

Remark
R3 ≠ 0

SST: Sense status

0	1 0 0 1	R3	1	1	dev.
1	4	3	1	1	6

Syntax
[<ident>]␣SST␣<r3>,<dev>

If SST is accepted a status character or word is transferred from the device to the register specified by R3. According to the type of the device only the right part of the register is used, in this case the left placed bits of R3 are reset to zero. The status word may have variable length. The following bits, if significant for the device concerned, have fixed positions:

15: not operable
14: throughput error
13: data fault
12: incorrect length

A device does not accept an SST instruction if it is not in 'ask for SST' state.
After a not accepted SST instruction the contents of the R3-register are not significant.

Type	*Execution Time*	
	P855	*P860*
T8	3.6	2.52

Condition register
CR = 0 command accepted
1 command not accepted
3 device address inknown

Remarks
<dev> must be ≠ 0.
R3 ≠ 0.

OTR: Output from register

0	1 0 0 0	R3	0	F	dev.
1	4	3	1	1	6

Syntax
[<ident>]␣OTR␣<r3>,[0|1],<dev>

If OTR is accepted a data character or word from the register specified by R3 is transferred to the device. According to the type of device the lower bits (right placed) or R3 or the whole 16 bits of R3 are taken into account by the control unit.

During execution, 'F' and <dev> field are sent to the device via the I/O bus. F may be used to specify a particular output function (e.g. binary or ASCII).

OTR is not accepted if the device control unit is not in the 'Exchange' state.

Type	*Execution Time*	
	P855	*P860*
T8	3.6	2.52

Condition register
CR = 0 command accepted
1 command not accepted
3 device address unknown

Remark
R3 ≠ 0

TST: Test status

0	1 0 0 1	R3	1	0	dev.
1	4	3	1	1	6

Syntax
[<ident>]␣TST␣<r3>,<dev>

This instruction may be used before starting any I/O operation to test if the device control unit is in the ready state. It is always accepted by the control unit.
During the execution of the TST instruction a status word is sent from the control unit to the register specified by R3.
A 1-bit in position 15 of R3 indicates the control unit is busy (not ready). Other bits are not significant.

Type	*Execution Time*	
	P855	*P860*
T8	3.6	2.52

Condition register
CR = 0 Accepted
3 device address unknown

Remark
R3 ≠ 0

WMP: Write mask protection

1	1 0 0 0	R3	0	1	0 0 0 0 0 0
1	4	3	1	1	6

Syntax
[<ident>]␣WMP␣<r3>

The contents of the register specified in R3 replaces the contents of the memory protection key register.
Flip-flops in this register are numbered from 0 to 15 and correspond to the memory sectors 0 to 15.
A 1-bit in this key register indicates that the corresponding memory sector (= 1024 16-bit words) is protected; a 0-bit means that the memory sector is not protected.
For memory protection, see also Volume 1: Hardware.

Type	*Function*	*Execution time*	
		P855	*P860*
T8	(R3) → M.P. key register	2.6	2.28

Condition register
Unchanged.

Remark
R3 ≠ 0

RIL: Read interrupt lines

0	1 0 0 1	R3	0 0	0 0 0 0 0 0
1	4	3	2	6

Syntax
[<ident>]␣RIL␣<r3>

This instruction is used to load in the register specified by R3 the state of the 16 interrupt lines, numbered from 0 to 15.

The active and non-masked lines are indicated by 1 in the corresponding bit position.

The inactive or masked or non-existent lines are indicated by 0.

Type	*Function*	*Execution Time*	
		P855	*P860*
T8	Interrupt lines → R3	2.6	2.28

Condition register
Unchanged.

Remark
R3 ≠ 0

WIM: Write interrupt mask

0	1 0 0 0	R3	0 0	0 0 0 0 0 0
1	4	3	2	6

Syntax
[<ident>]␣WIM␣<r3>

The contents of the 16-bit register specified by R3 replace the contents of the mask register (15 bits).

The bits in the mask register are numbered from 1 to 15 and correspond to the interrupt lines 1 to 15.

A one-bit in the mask register indicates the corresponding line is inhibited; a zero-bit means the line is not inhibited.

Type	*Function*	*Execution Time*	
		P855	*P860*
T8	(R3) → mask register	2.6	2.28

Condition register
Unchanged.

Remark
R3 ≠ 0.

WM2: Write mask protection no. 2 (P860 only)

0	1 0 0 0	R3	1 1	0 0 0 0 0 0
1	4	3	2	6

Syntax
[<ident>]␣WM2␣<r3>

The contents of the register specified in R3 replace the contents of the memory protection key register no. 2.
Flip-flops in the key register are numbered from 0 to 15 and correspond to the memory sectors 16 to 31.
A 1-bit in the key-register indicates that the corresponding memory sector is protected; a 0-bit that it is not protected.

Type	*Function*	*Execution Time* *P860*
T8	(R3) → M.P. key register	2.28

Condition register
Unchanged.

Remark
R3 ≠ 0.

RCA: Read channel address

0	1 0 0 1	R3	01	0 0 0 0 0 0
1	4	3	2	6

Syntax
[<ident>]␣RCA␣<r3>

This instruction is used to load into the register specified in R3 the channel number which has caused a MIDB interrupt request. It deactivates the BUL line and resets the MIDB interrupt flip-flop.

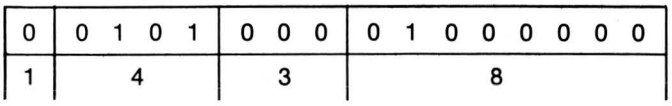
Type	*Function*	*Execution Time* *P855*	*P860*
T8	CNL → R3	3.6	2.52

Condition register
Unchanged.

Remark
R3 ≠ 0.

Miscellaneous Instructions

ENB: Enable interrupt



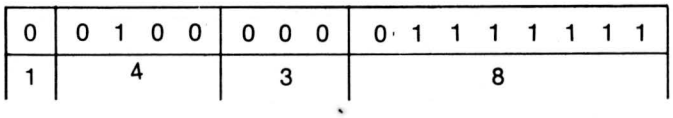
Syntax
[<ident>]␣ENB

This instruction sets the machine status to 'permit interrupt'.

Type	*Execution Time*	
	P855	*P860*
T8	1.9	1.56

Condition register
Unchanged.

HLT: Halt



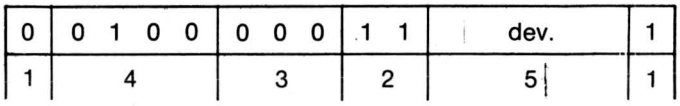
Syntax
[<ident>]␣HLT

This instruction sets the machine to halt mode. No further instructions or interrupts will be serviced until the console start button is pressed at which time normal execution resumes.

Type	*Execution Time*	
	P855	*P860*
T8	1.9	1.56

Condition register
Unchanged.

RIT: Reset internal interrupt



Syntax
[<ident>]␣RIT

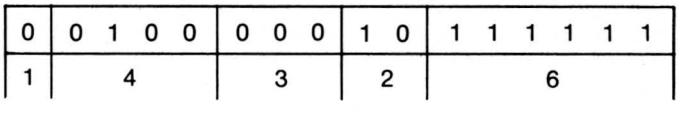
This command is used to clear one of the internal interrupt bits which is indicated by a zero-bit in the <dev> field (other bits being one). Internal interrupts are:

bit number 10 = control panel
11 = power failure/automatic restart
12 = real time clock
13 = program error
14 = memory protect error.

Type	*Execution time*	
	P855	*P860*
T8	1.9	1.56

Condition register
Unchanged.

INH: Inhibit interrupt



Syntax
[<ident>]␣INH

This instruction resets 'permit interrupt' status to prohibit all the program interrupts except the power failure line.

Type	*Execution Time*	
	P855	*P860*
T8	1.9	1.56

Condition register
Unchanged.

LKM: Link to monitor

0	0 1 0 1	0 0 0	0 0 0 0 0 1 0 0
1	4	3	8

Syntax
[<ident>]␣LKM

This instruction sets a program interrupt flip-flop to transfer from user mode to system mode.

Type	*Execution time*	
	P855	*P866*
T8	1.9	1.56

Condition register
Unchanged.

SMD: Set mode

0	0 1 0 1	0 0 0	0 0	0 0 0 0 0 1
1	4	3	2	6

Syntax
[<ident>]␣SMD

This instruction is used when the memory protection option is available. It sets the indicator mode to switch from system mode to user mode.

Type	*Execution time*	
	P855	*P860*
T8	1.9	1.56

Condition register
Unchanged.

PART 4 SYSTEM SOFTWARE

Introduction to P855 and P860 Software

Programming a modern computer can involve the use of a large or small number of components, both hardware and software. Various aspects of hardware, such as the central processor, peripheral devices and interrupt system, have been discussed in other parts of this manual; it is here that we consider the software aspect.

A program is built up of instructions usually written in a language which can be easily understood by the programmer but not by the computer. This language is called Assembly Language or Autocode.
Before it can be executed, therefore, the program must be translated into the language of the computer by a program called an Assembler. The resulting object program can then be linked to other programs if desired, tested by a debugging package and finally executed under the supervision of some sort of control program.

The number and complexity of the software components to be used at any one time is limited by the amount of memory available with the computer and by the number and type of the devices attached to it.

The P855 and P860 have the same machine language but vary considerably in the amount of memory space available: 4k to 16k for the P855 and 4k to 32k for the P860. Most of the standard software can be used on either computer, depending on the memory size.
For example, if more than 4k of memory is available, the extended version of the Basic Assembler can be used, with extra facilities such as the definition of common areas.

A Linkage Editor can be used to link together object programs which make external references to one another. If all external references can be satisfied, the resulting program can be output onto magnetic or punched tape, or be loaded into memory during the linkage process.
The Update Package or Text Editor can be used to correct source programs (on a line or character basis) or object programs, by deleting components which are no longer required and inserting new ones.

The available Monitors are of a modular structure, so that the user can select those modules which he requires for his configuration. The Basic Executive Monitor provides monitor requests and, on option, operator communication. It handles one program at a time.
The Basic Real Time Monitor uses a foreground and a background area in memory and can handle more than one program concurrently. It offers extensive facilities for operator communication and monitor requests.

A debugging module is provided as part of the executive package. By typing in a number of messages on the operator's typewriter the user can write a number of words into memory, get a dump of the contents of a specified section of memory onto the typewriter or tape punch.
A separate debugging package, running under monitor control is also provided.

The system also includes routines to perform several arithmetic functions (multiply, divide, double add, double subtract, floating-point arithmetic) in case these optional instructions are not included in the hardware. It also provides a number of standard input/output routines. These routines can be called by means of a Call Function (CF) instruction in the user program, certain registers being used to specify various parameters.

Programming conventions

INTERRUPT RESPONSE TIMES

The interrupt latency can be defined as the time to get from an arbitrary program execution point to the first instruction of the service routine for an interrupt source. This depends on:

- Whether there are 'software levels' or 'hardware priority levels',
- The execution time of the longest instruction;
- The memory cycle time of the computer.

During an interrupt the following sequence takes place:

1 The current instruction is terminated.
2 The interrupt vector is processed as follows:
  - PSW is saved in the stack
  - A15 is decremented
  - Interrupt inhibit is set
  - Indirect branch is made to one of the 48 dedicated locations in lower memory, depending on the priority hardware level activated.
    The locations contain the start address of device service routines.

The latency is then calculated as:

Tl = Tm + Tv
where Tl = latency time
Tm = execution time of longest instruction
Tv = time to process the interrupt vector.
and Tm = 14 logic cycles + 16 memory cycles
(Mult. Store)
Tv = 2 logic cycles + 7 memory cycles
Tl = 16 logic cycles + 23 memory cycles
(l.c. = 720 nsec)
(m.c. = 840 (860)
= 1.2 (855)
∴ Tl (P860) = 30.84 usec
(P855) = 39.12 usec

These figures indicate the 'worst case'. When the Multiple load/store instruction is not used, the 'best case' can be said to give:

Tl (P860): 13.8 usec.
(P855): 18.46 usec.

STACK HANDLING

The size of the stack should be carefully calculated by the programmer. Its size will depend upon:

- How many interrupts will be in the stack at any one time.
- How much information will be saved for each interrupt.
- How much memory space is available after loading the program or programs.

To calculate the number of stack locations (16-bit words) needed by each interrupt that will be in the stack, allow one location for each of the following:

- The contents of the P-register
- The contents of the Program Status Word (PSW)
- The contents of the mask register
- The contents of each accumulator which needs saving
- Each parameter which needs saving

The contents of the P-register, the PSW and the mask register are always saved for each interrupt, the contents of the other registers and the needed parameters are selected by the programmer. A typical example is shown below.

Example
Maximum number of interrupts which will be in the stack is 3.
1st. Interrupt - requires contents of 3 registers plus two parameters.
2nd. Interrupt - requires contents of 1 register.
3rd. Interrupt - requires contents of 14 registers plus 1 parameter.

therefore:

1st. Interrupt - needs 3 (P + PSW + Mask registers) + 3 (registers) + 2 (parameters) = 8 locations.
2nd. Interrupt - needs 3 (P + PSW + Mask registers) + 1 (register) = 4 locations.
3rd. Interrupt - needs 3 (P + PSW + Mask registers) + 14 (registers) + 1 (parameter) = 18 locations.

so the total number of stack locations needed by this example program would be:

8 + 4 + 18 = 30 locations

Access to the stack is made using register 15 as a stack pointer, the contents of this register always pointing to the next available memory address in the stack. Each time new data are loaded into the stack the contents of the stack pointer are automatically decremented so as to point to the next available address and conversely when data are retrieved from the stack, the contents of the pointer are incremented.

The stack is always filled towards the dedicated area of memory (locations 0 to 128) and before a program is started, the starting address of the stack area must be loaded into register 15. To calculate the starting address of the stack, a constant of 128 must be added to the number of stack locations needed by the interrupts. If this were done with the result of the example above, the starting address would be:

30 + 128 = 158

so 158 would be loaded into register 15 at the start of the program. This constant of 128 protects the dedicated area of memory from being overwritten.

If more than one program is to be using the processor at the same time, as for instance with a multi-tasking real time monitor system, then the programmer must decide whether to have one stack area which will be used by all the programs or whether each program will have a dedicated area within it which can be used as a stack area. If the former is chosen, then the starting address must be calculated by adding the number of locations needed by each program to the constant of 128.

e.g. three programs need respectively A, B and C number of locations, then starting address will be:

A + B + C + 128 = N

where N represents the total obtained by adding the four numbers.

With actual programs A, B, C and N would be real numbers and the value which N represents in the above example would be loaded into register 15 as the starting address of the stack. If the second method is chosen, there is no need to add the constant because each program will have its own area; it would, however, be wise to add a few locations to the number needed as an overflow area, and included in this area should be a routine to tell the operator that the stack is full.

A typical example of how the system operates is given below.

Operation

When an interrupt occurs, it is processed by the automatic stack handling routine. Part of the routine is done by hardware and part by software. These routines follow the sequence shown in the flowchart in figure 1.

Hardware Routine

When an interrupt line becomes active, the hardware routine stops the running program when the current instruction has been completed.

Then:

1 The contents of the PSW are stored in the stack location indicated by the contents of register 15 (stack pointer).
2 The contents of register 15 are decremented, then the contents of the P-register are stored in the stack location which is now indicated by the contents of register 15.
3 The contents of register 15 are again decremented and the Inhibit Interrupt (GF-register) bit is set.
4 An indirect branch to memory address 32, which contains the start address of the software routine, is made.

Software Routine

The software routine first stores the contents of the required registers and the mask register etc., into the stack; then it tests the state of the interrupt lines to find out which are active and which one has the highest priority. Each interrupt level will refer the software routine to a specific memory address which in turn contains the starting address of the routine which will service the routine.

If other interrupts are expected, the Enable Interrupt (GF-register) bit is set, then the routine which services the interrupt is started; in this routine an instruction must be included to reset the interrupting signal.

If during this routine one of the expected interrupts occurs, then:

1 The routine processing the current interrupt is itself interrupted.
2 The hardware routine starts the whole sequence again; storing the relevant information about the interrupted routine into the stack and branching to the software routine which then starts to process the new interrupt.
3 When processing of the new interrupt has been completed (provided that no new interrupt has occurred), the relevant information about the interrupted routine for the first interrupt is reloaded from the stack and the routine is continued from where it was interrupted.

When the software routine has completed the interrupt process, the old contents of the registers and mask register etc., are reloaded from the stack and a return instruction, referring to register 15, is made.

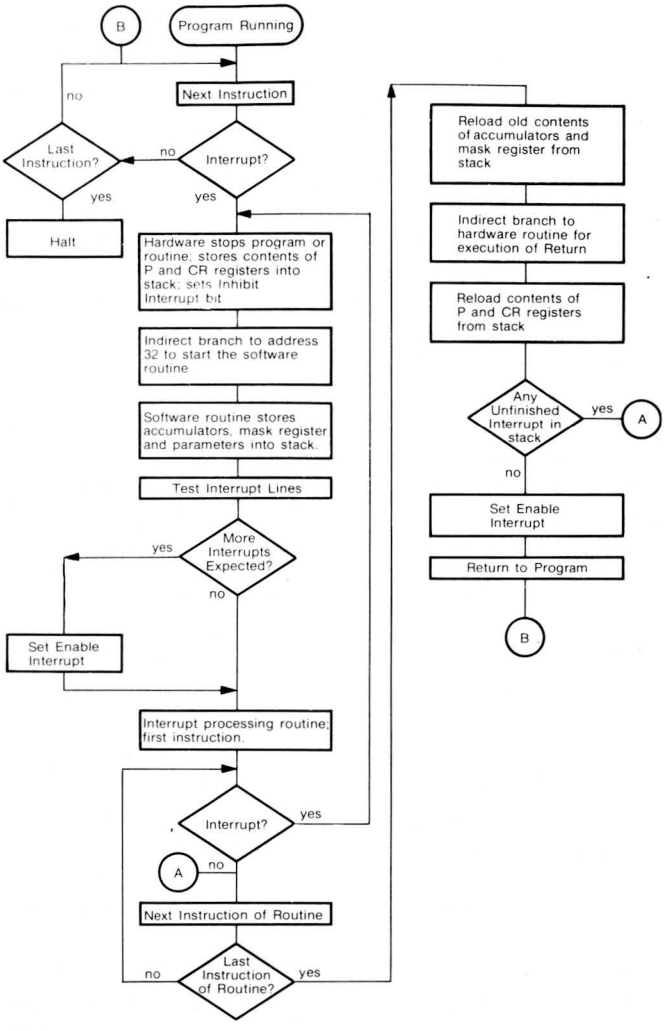


Figure 1 Stack Handling Operation

During the execution of the Return instruction:

1 The contents of register 15 are incremented, then the contents of the stack location indicated by register 15 are loaded into the P-register.
2 The contents of register 15 are again incremented, then the contents of the stack location indicated by register 15 are loaded into the PSW.
3 Finally the Enable Interrupt bit (GF-register) is set.

When the last interrupt has been serviced, the interrupted program is resumed.

Basic Executive Monitor

A Monitor is a system program used to load, supervise and execute user programs.
The Basic Executive Monitor is such a program intended for use on the P855 and P860 with a minimum configuration of 4k words of memory and an operator's typewriter. It is of modular structure, which means that the user can order those modules of the Monitor which he will need for his system, depending on his requirements and the types of periphery used.
Basically, the monitor is a one-level executive system, consisting of two parts:

- the loader, which, having first been loaded into the highest part of memory by bootstrap, is used to load the executive part of the system and after that the user program;
- the executive part, i.e. the actual Monitor, which handles the interrupt signals, controls I/O operations for typewriter and fast punched tape equipment and is able to execute some functions required by the user program through Monitor Requests.

If desired by the user, this Monitor can be expanded to include some additional Monitor Requests, basic operator communication and debug Routines.

Although the available interrupt lines apply only to hardware interrupts, it is possible to do a kind of multi-tasking within this system. This is done by so-called 'scheduled labels' which allow the user to run his main program, while, for example, an I/O operation is taking place. At completion of the I/O operation the system interrupts the main program to give control to such a 'scheduled label' routine. At the exit of this routine control returns to the main program if no other 'labels' or routines are scheduled.

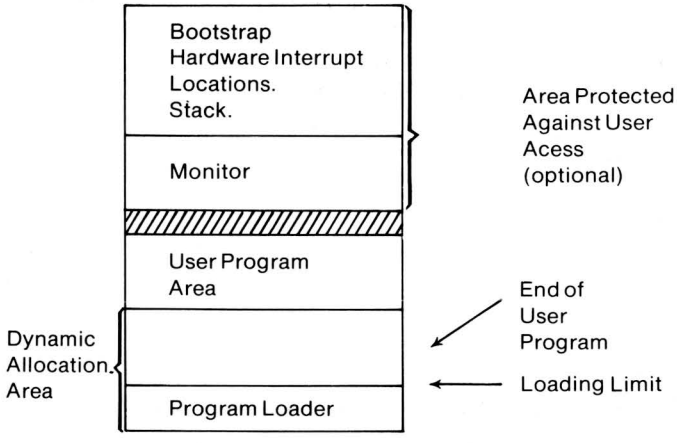
CONCEPTS AND RULES

Organization

Memory Organization
The memory is, basically, divided in two parts:

- The system area (from address 000 to 0800 for minimum configurations), containing the bootstrap, interrupt locations, stack area and monitor. This area can, on option, be protected against user access.
- the user area, which, at loading time, contains the Program Loader (previously loaded by bootstrap). This area is located at the end of the memory. The Loader area can be used by the user program by means of dynamic memory allocation, i.e. through Get Buffer and Release Buffer monitor requests.

At execution time, the layout of the memory is then as follows:



Interrupt System
The standard internal and external interrupts, such as tape reader, tape punch, typewriter, monitor request, memory protect, are handled by the system. These signals can be connected either to different level lines or to the common line which is covered by the interrupt mask register, on which the signals can be masked. This can be done by instructions which provide for setting or resetting one or more bits in the mask register.

Stack
The stack is a group of locations which are used to save the Program Status Word (PSW) and necessary registers and parameters, when an interrupt occurs.
The address of the first free location of the stack can be obtained at any moment by reading the contents of the A15 register; this address has been calculated and defined at SYSGEN and is loaded by the monitor (see Programming Conventions). The A15 register is protected and if an attempt is made to modify it in 'user mode' an interrupt occurs. There is also an interrupt when the stack (A15 register) reaches memory location 128_{10}.

File Codes
file codes are used for distinguishing devices and files. Some of these file codes are standard. These are the following (in hexadecimal):

01 = Standard Source Input
02 = Standard Listing Output
03 = Standard Punch Output
04 = Standard Object Input
05 = Standard Operator Keyboard (Input + Output)
0E = Standard Library Input
0F = Standard Program Input.

Scheduled Labels

The user can make a request for certain functions through Monitor Requests. Such a Request consists of loading necessary parameters in A7 and/or A8 registers, giving an LKM instruction and a DATA directive followed by a number. If this number is negative, this indicates that the user is scheduling a label upon completion of a request. This label identifies a user routine to which control is given then. In this request the user can for example, analyze the results of a Monitor Request.

For example:

```
LDK   A7,PARAM1
LDKL  A8, PARAM2
LKM
DATA  -1
DATA LABEL
```

Having loaded the A7 and A8 registers with the necessary parameters, a Monitor Request is given with the DATA directive containing the two's complement of the number identifying a specific Monitor Request, in this case the two's complement of 1, i.e. -1. This indicates an I/O Request. Because DATA contains the two's complement of this number, this means that a label is scheduled following the request: DATA LABEL. Upon completion of this I/O request control is given to the program part attached to LABEL. Thus, the main program is interrupted, its registers are saved and the user routine attached to LABEL is started with these values of the registers. When the scheduled label routine gives an EXIT Monitor Request, control is returned to the main program.

Such a scheduled label routine cannot be interrupted by another scheduled label routine. They are recorded in a list and when one routine exits, control is given to the next one according to 'last-in-first-out' method. At any time during execution, up to seven scheduled labels may be recorded in that list. The program may, of course, contain more.

Use of Interrupt System

User Interrupt Programs

Any user interrupt program running in enable mode, i.e. able to receive interrupts, must begin by saving the first 8 registers into the stack area, pointed to by the A15 register. When the program is interrupted, control is given to the dispatcher, a part of the monitor which checks the priority of the interrupts and transfers control to the interrupt with the highest priority. This dispatcher always assumes that the old contents of the first 8 registers can be found in the stack. At the end of the user interrupt program these 8 registers must be restored before a return is made to the A15 register.

When a program (interrupt or not) is running in enable mode, no Call Function instruction must be specified which affects the A15 register.

Interrupt Mask

The interrupt mask is a software mask written by the user to enable or inhibit interrupts on the 16-signal common line (see also Hardware Part), by masking or not masking the corresponding signals.

It is recommended to write this mask before writing the WIM instruction. If any of the standard interrupt lines are connected to this common line, the monitor will take care of the processing of these interrupts on signal lines (bits) 0 through 7 without masking these lines. The other free lines (bits 8 through 15) must then be processed by the user by means of a half mask (right character only).

Dynamic Memory Allocation

In order to provide the user with the possibility of making use of the memory area behind the user program, where the program loader is also allocated (see memory layout), two requests are available:

- Get Buffer, by means of which the user can reserve a buffer for his own use.
- Release Buffer, by means of which he can release the area created by the previous request.

This memory allocation is done in blocks of which the length is specified by the user. Each block is preceded by a control block containing links and other parameters. The user must be very careful not to destroy these.

OPERATION

Processing without operator communication

After loading the bootstrap, the program loader tape is loaded at the far end of memory (the high value addresses). The program loader, having received control from the bootstrap, first checks if the bootstrap has been loaded correctly by reading the checksum. The loader occupies about 350 words. If the bootstrap was not loaded correctly, the following message is output:

```
BOOT  CK  ERROR
```

If it was correct, the loader asks for the system tape by printing out:

```
SYSTEM TAPE ON READER, PLEASE
```

It then loads the tape from hexdecimal address 0040 to 0800 (for minimum configuration; this may vary, depending on the system configuration). Then it asks for the user tape and halts:

```
USER TAPE ON READER, PLEASE
```

At this point, register A9 contains the beginning address of the program user area and register A1 contains the value 1, indicating that the program will run in user mode. If the program is to run in master mode, the user may reset bit 15 of the A1 register. If he wants to change the loading base address, he may alter the contents of the A9 register.
The user program may then be loaded by depressing the START button. Loading is terminated with a Halt at a specific address. When loading has been completed, the system is ready for execution of the user program. To initiate this the operator must press the START button again.

Processing with Operator Communication
As described in the previous paragraph, the program loader is loaded by bootstrap and receives control.
If everything has worked correctly, the system is loaded and upon completion of this procedure control is given to the idle task. At that point, the operator can interfere through the control panel. Then the operator can push the INT button, causing the system to print M: Now the operator can load the user program by typing LD. When loading has been completed, the operator can give another control panel interrupt and type ST. Now program execution is started.

Monitor Requests
The user program can make requests to the Monitor for specific functions. This is done by first giving an LKM (Link to Monitor) instruction and then a DATA directive with a number as operand. This number indicates specifically the function requested from the Monitor. Basically, three Monitor Requests can be made:

- I/O Requests = LKM
  DATA 1
- Wait Request = LKM
  DATA 2
- Exit Request = LKM
  DATA 3

On option, the basic monitor can be expanded with a module providing for two additional requests, to be used for dynamic memory allocation:

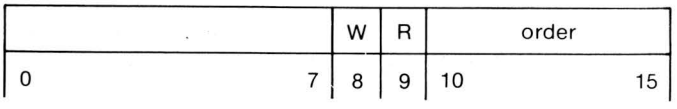
- Get Buffer Request: LKM
  DATA 4
- Release Buffer Request: LKM
  DATA 5

Preceding these requests, certain parameters may need to be loaded in the A7 and or A8 registers.
Following each request, the system gives the result in the A7 register. In the following paragraphs the function and calling sequence of each request is described.

I/O Requests
Calling Sequence:

```
LDK   A7, CODE
LDKL  A8, ECBADR
LKM
DATA 1
```

where CODE must be specified as follows:



ECBADR is the address of the Event Control Block, containing the parameters for the requested I/O function.

Use: Through this request the user can ask the system to start a certain I/O operation on a peripheral device. The parameters loaded in the A7 registers have the following meaning:

Bits 8 and 9 specify the mode of operation:
W = 1: the requesting program wants to wait for the completion of the I/O operation requested. Only after completion of the requested function, the return to the calling program will take place.
W = 0: a return to the calling program will be made as soon as the request has been sent to the channel. The program will give a Wait request later on for synchronization.
R = 1: The program itself will process any abnormal condition concerning the requested operation. The system will return the Hardware Status in ECB word 4.
R = 0: Any abnormal conditions will be processed in the system. The Software Status is returned in ECB word 4.
ORDER specifies which I/O function is requested, by giving one of the following combinations of 2 hexadecimal digits.
01: Basic Read.
05: Basic Write.
For Basic (Binary) I/O requests the system does not provide for character checking or data conversion, only for control command initialization and end of operation signals.
02: Standard (Object) Read.
06: Standard Write.
Standard (ASCII or Object) I/O requests provide, by means of standard conversions, for special features such as error control characters, conversion from external code to internal ASCII and vice versa. There is a checksum and characters are stored seven by seven bits, two characters to a word.
07: Object Write (4×4×4×4 tape format).
08: Object Write (8×8 tape format).
Object I/O requests provide, by means of standard conversions, for special features such as error control characters, checksum and data conversion from external 4×4×4×4 or 8×8 tape format to internal 16-bit format.

14: Skip to next EOS mask.
16: Skip to next EOF mark.
22: Write EOF mark.
26: Write EOS mark.
30: Get information about file code.

For 02, 07 and 08 at least 8k words of memory are required.

For each of these request orders, the following specific information applies to the various peripheral devices:

Basic Read (X'01'):

Operator's typewriter: All characters are entered on 7 bits until the requested length is reached.

ASR tape reader: Same as for the keyboard. The reader stops one character after an X.off code is read.

High speed tape reader: All characters are entered on 8 bits, without checking or special features, until the requested length is reached.

Card reader: All the words are entered and stored in Hollerith code on twelve bits (4 to 15). In each word the column image is right justified. The words are stored until the requested length is reached. The length is given in words.

Basic Write (X'05'):

Operator's typewriter: All characters are output without checking or special features. This order can be used to print something and have the answer on the same line.

ASR tape punch: Same as for printed output.

Line printer: All characters are output without checking. There is no control character.

High speed tape punch: All characters are output without checking or special features.

Standard Read (X'02'):

Operator's typewriter: ASCII characters are entered on 7 bits, with the following special features:
- the special characters, coded from X'0' to X'1F' (hexadecimal), are ignored.
- code X'7F' (Rub-Out or Delete character) is ignored.
- code X'5F' (←) can be used to delete the preceding character. If more ← are used consecutively, an equal number of preceding characters will be deleted.
- code X'5E' (↑) is used to delete the line preceding it, up to the next carriage return.
- code X'OD' (carriage return) indicates end of block. It is the last character to be entered. It is not transmitted to the user's buffer.
- code X'5C' (\ ) is used as a tabulation symbol (see ECB-word5).

If the address of the tabulation table is zero, or if the number of tackets is zero, or if the storage address is greater than the last tacket, the code X'20' is stored in the buffer. In other cases, X'5C' is not stored and replaced by spaces, as indicated by the tackets in the tabulation table.

ASR tape reader: For ASCII characters, the same features apply as for the keyboard: the code for carriage return must be preceded by the code X-off.
For object code in 4×4×4×4 tape format, the first character identifies the object format. It must range from X'18' to X'1F' and is converted to a number from X'0' to X'7' and stored on one character. The second character contains the word-length of the input block, excluding the first word and the checksum. Each punched row (4 bits)entered after this identifier is stored on one half-character up to the checksum. When the checksum has been read, input is stopped.The 8×8 tape format cannot be read on the ASR tape reader.To start the reader,an X-on code is sent by the system before entering the characters.

High-speed tape reader: Same as for ASR tape reader. In addition: for object code in 8×8 tape format, the first character, identifying the object code format, must have one of the following values: X'10', X'1' to X'4' or X'15' to X'17'. It is converted to a number from X'0' to X'7'. Each punched row (8 bits) entered after this identifier is stored on one character up to the checksum. The second character is the length of the block, in words, excluding the first word and the checksum.

Card reader: All words are read in Hollerith code, on 12 bits, converted and stored in ASCII code, on 7 bits, until the requested length is reached. Words which are not in Hollerith are converted into the ASCII code for X'20' and a 'data fault' status is returned in the software status (ECB word 4) (bit 13 is 1). There is no special code. However, EOS and EOF marks are detected (bits 14 and 15 in software status).

Standard Write (X'06'):

Operator's typewriter: All characters, except X'0' to X'1F' (special code characters) are output without checking. At the end of a line, a carriage return and line feed are output.

ASR tape punch: Same features as for keyboard. At the end of a line, the following character sequence is output:

LF -X-off- CR - R.out

High-speed tape punch: Same as for ASR tape punch with the following exception: the first character is a control code: if it equals X'30' or X'31', it is output as Line Feed; if it is different, it is not output.

Line Printer: All characters are output without checking, except for the control code. It must be stored in the word preceding the buffer address. This control code may have three values:

\+ X'2B': print the line without advancing the paper (superposition).
0 → X'30': advance two lines before printing.
1 → X'31': skip to top of page before printing.

All other control codes are used as normally: advance one line and print.
At the end of the buffer, after the requested length, one word must follow to be used by the system for a print code. If the requested length is more than one line, the system puts a print code after the maximum length and the buffer will be printed on two or more lines.

Object Write (4×4×4×4 tape format: X'07'):

ASR tape punch: The first character is output on one row, converted from X'0' - X'7' to X'18' - X'1F'. Each following character is output on two rows; to avoid special (ASCII) code each row is converted. The second character contains the length of the block in characters, excluding the first character. At the end an 8-bit checksum is performed and punched, followed by an X-off code.

Hig-speed tape punch: Same as for ASR tape punch, except that the second character contains the length in words.

Object Write (8×8 tape format: X'08'):

High-speed tape punch: The standard object code is output in 8×8 format, where the first character is a format character and is output on one row row, converted as follows:

X'0' → X'10'.
X'1' to X'4' → X'11' to X'14'
X'5' to X'7' → X'15' to X'17'.

The second character contains the length in words, excluding the first word. An 8-bit checksum is performed and punched.

Write EOF mark (X'22'):

For operator's typewriter and ASR tape punch an end-of-file mark is output as follows: :EOF LF-X-off-CR-R.out

High-speed tape punch: Same as for ASR.

Line printer: An End-Of-Segment is output: :EOS

Write EOS mark (X'26'):

For *operator's typewriter* and *ASR tape punch*, an end-of-segment mark is output as follows: :EOS LF-X-off-CR-R.out

High-speed tape punch: Same as for ASR.

Line printer: An End-Of-Segment is output: :EOS

Read up to End-Of-Segment (X'14'):

High-speed tape reader: The tape is read until an :EOS statement has been read.

Card reader: The cards are read until an :EOS statement is read.

Read up to End-Of-File (X'16'):

High-speed tape reader: The tape is read until an :EOF statement has been read.

Card reader: The cards are read until an :EOF statement has been read.

Return information about a file code (X'30'):

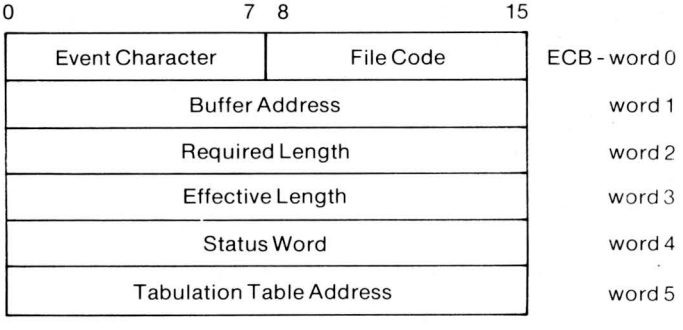
By means of this order it is possible to find out the assignment of a file code. The information will be returned in the Event Control Block:

ECB - word 0: Event Character + File Code.
ECB - word 1: Device Name (2 ASCII characters):
TY = operator's typewriter.
TR = ASR tape reader.
TP = ASR tape punch.

ECB - word 2: maximum record size (72 characters available as output).
ECB - word 3: left character: unused.
right character: device address.
ECB - word 4: Status = 0.

If the file code in ECB - word 0 is set to zero, the other words of the ECB will also contain zeros.

The *Event Control Block* (of which the address must have been loaded in the A7 register) has the following format:



where:

word 0: event character: the monitor stores all the information in the bits of this character:
bit 0 = 1: end of operation has occurred for this ECB
bit 1: unused.
bit 2 and 3: not significant for I/O.
bit 4 = 1: file code has not been assigned.
bit 5 = 1: abnormal end of operation.
bit 6 = 1: end-of-segment has been read.
bit 7 = 1: end-of-file has been read.
word 1: address of the user buffer.
word 2: requested length to be read or written (in words or characters).
The first character is always the character given by the buffer address.
word 3: effective length which has been transmitted (in words or characters). This information is stored by the monitor upon completion of the I/O operation.

word 4: Status word, stored by the monitor upon completion of the requested I/O operation.

- For Basic orders, this word will be filled with the hardware status by the control unit. However, if the monitor detects an error in the calling sequence, bit 0 will be set to 1 and the other bits will contain the software status.
- For the other orders the software status will be returned:

  bit 0 = 0: the operation has been successfully completed.
  bit 10 = 1: beginning of tape encountered.
  bit 11 = 1: end of input medium.
  bit 12 = 1: requested length is incorrect.
  bit 13 = 1: illegal character code.
  bit 14 = 1: an EOS mark has been read.
  bit 15 = 1: an EOF mark has been read.

  When the operation was not successfully completed, bit 0 is set to 1 and bit 1 is set to 0 (retry also was not possible). In this case bits 2 to 15 give the hardware status.

  When the monitor has detected an error in the calling sequence, bits 0 and 1 are both set to 1 and bits 11 to 15 have the following significance:
  bit 15 = 1: illegal file code has been used.
  bit 14 = 1: device is attached to another program.
  bit 13 = 1: ECB address is illegal.
  bit 12 = 1: buffer size or address is illegal.
  bit 11 = 1: function is unknown or not compatible with the device.

word 5: tabulation table address for the punched tape equipment.

This tabulation table has the following format:

Number of Tackets	First Tacket
Second Tacket	Third Tacket

etc.

The tackets have an absolute position in the Line. Characters up to the following tacket are filled with blanks.

Example:

3	10
20	30

Input Line: LABEL\OPER\ OPERAND\
COMMENT
Line in Buffer:
LABEL␣␣␣␣OPER␣␣␣␣␣␣OPERAND␣␣␣
1 10 20
COMMENT
30

At completion of input, the buffer is filled with spaces, but the returned length is the length effectively entered and stored, including the spaces replacing the tabulation codes (\ ). Upon completion of one of these requests, the system responds as follows:

A7 = 0: request completed.
= 1: the corresponding processor is not in memory.
= 2: the requested function cannot be processed.

Wait for an Event

Calling Sequence:

```
LDKL  A8, ECBADR
LKM
DATA  2
```

where ECBADR gives the address of the Event Control Block (see I/O requests).
The first character of the ECB is the event character. If the first bit of this character is set to 1, the event has been completed.

Use:

By means of this request the user can stop his program temporarily to wait for an event (e.g. end of an I/O operation).
This must be an event performed by other processors or by 'user interrupt' routines.
Note: A 'scheduled label' routine may **not** contain a Wait request.

Upon completion of the request, the system responds as follows:
A7 = 0: request completed.
= -1: the corresponsing processor is not in memory.
= -2: the requested function cannot be processed.

Exit

Calling Sequence:

```
LKM
DATA 3
```

Use:

This request is used to indicate the end of a user program or 'scheduled label' routine. Control is returned to the dispatcher which determines whether the request comes from the main sequence or from a 'scheduled label' routine. In the first case, the run is terminated, control goes to the initialization routine which reinitializes the table and buffer area and then stops, ready to accept a new run or the loading of a new program. In the second case, control is given to another 'scheduled label', if there is one present, otherwise control is returned to the main sequence.

The following two requests can be used only on option, if this module is included in the monitor. They are used for dynamic memory allocation:

Get Buffer

Calling sequence:

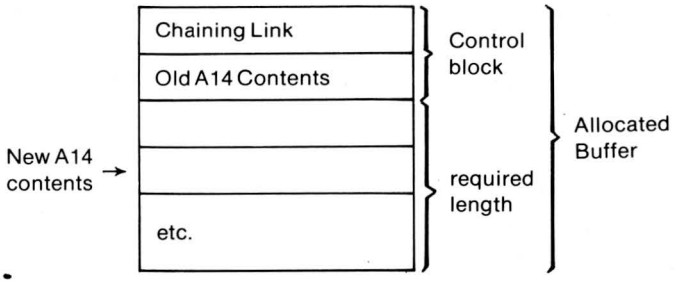
```
LDK   A7, LENGTH
LKM
DATA  4
```

where LENGTH is the length, in characters, which is to be allocated to the buffer area.

Use:

By means of this request, the user can allocate a memory area for temporary use, in the dynamic memory allocation area, behind the user program area.

When the allocation is made, a control block is created by the system at the beginning of the allocated area. This block will contain a chaining link and the old contents of the A14 register:



The user must be very careful not to destroy this control block.

Upon completion of this request, the system responds as follows:

A7 = 0: the buffer is allocated
= 1: there is no memory space available.

A14 contains the address of the second word of the buffer. (See above). The old A14 contents are saved. The A14 register gives the address of the **fourth** word of the allocated buffer, to enable the user to make a Call Function instruction with the A14 register as soon as the buffer is allocated without having to update the A14 register first.

Release Buffer

Calling Sequence

```
LDK   A7, LENGTH
LDK   A14, BLOCAD
LKM
DATA 5
```

where:

LENGTH is the length of the buffer, in characters.

BLOCAD is the beginning address of the buffer as given in A14 after a Get Buffer request.

Use:

By means of this request, the memory space reserved by a Get Buffer request can be released. The old contents of A14 is restored and the block is put in the system free block list.

Upon completion of the request, the system responds as follows:

A7 = 0: the buffer is released.
= 1: parameter error, or area has been destroyed.

A14: previously saved contents.

OPERATOR COMMUNICATION

On option, the monitor can be expanded to include modules for handling the following Operator Control Messages.

These control messages are entered via the typewriter. This is done by first pressing the INT button on the CPU control panel. The system then types out M: and the operator can type a control message. Such a message consists of 2 characters, identifying the message,followed by a space,possibly followed by one of the parameters. These parameters are separated by spaces.

- Numerical values used must be hexadecimal values.
- If a message contains an error, the system types out ER. The operator may then press the INT button and type the correct message.

The following messages can be used:

Load a Program

syntax: LD␣[M]

use: This message is used to load a program into memory. M, if present, indicates that the program is in master mode.

Start a Program

syntax: ST

use: This message is given to start the execution of a user program, after it has been loaded into memory.

Hexadecimal Dump

syntax: DM␣BEG␣END

use: This message causes a memory dump in hexadecimal from the first address (BEG) to the second address (END) specified. BEG consists of up to 4 hexadecimal characters and specifies the beginning address of the area which is to be dumped. END consists of up to 4 hexadecimal characters and specifies the ending address of the area which is to be dumped.

Halt Dump

syntax: HD

use: If this message is given, a dump which is being output can be stopped. This can be very useful if the dump is taking place on the typewriter, as this is a very slow device.

Write into Memory

syntax: WM␣COAD␣VAL1␣[VAL2␣....VALn]

use: This message can be used to correct one or more memory locations. COAD is the address of this first location in memory which is to be corrected. It must contain up to 4 hexadecimal characters. VAL1, VAL2.... VALn are values which are to be entered in the memory locations starting from COAD, i.e. VAL1 is put in location COAD. VAL2 is put in location COAD+2, etc.

Example:

WM␣4FE␣44F␣3FE4

The value 44F is placed in memory location 4FE.

The value 3FE4 is placed in memory location 500.

Retry an I/O operation

syntax: RY␣DA

use: When the monitor has typed out an error message following an I/O operation, i.e. PU␣DNXX␣status [␣RY|RD], requesting operator intervention, the operator may type the RY message to retry that same I/O operation, after he has taken any necessary steps. DA is the device where the operation has to be retried (2 hexadecimal characters).

If the operation succeeds now, control is returned to the user with the status in the ECB. If he still does not succeed, a new error message may be typed out by the monitor, if there is another possibility for a retry.

Release Device

syntax: RD␣DA

use: When the monitor has typed out an error message following an I/O operation, i.e. PU␣DNXX␣status [␣RY|RD], and requests operator intervention, the operator can type the RY message if he wants to release the operation on the device which resulted in the error message.

DA is the physical device address (2 hexadecimal characters).

Control is returned to the user with the status in the ECB.

Error Handling

A complete list of error messages will be supplied in a later edition.

System Messages

A list of system messages will be supplied in a later edition.

Basic Real Time Monitor

The Basic Real Time Monitor is an executive program which can handle more than one program concurrently. This real time possibility is provided by several features:

- the system includes up to 47 hardware interrupt levels, 2 monitor levels and up to 14 software priority levels, to which user tasks may be connected;
- the memory can be divided into a foreground area and a background area;
- programs can be activated by one another.

In the background area of memory, programs can be assembled, compiled, link-edited, etc. and executed through control messages entered on the operator's typewriter.
The foreground area can be divided into partitions of at least 1k words each, to be used by different user tasks, under monitor control.
In addition, this monitor allows for:

- connecting user programs to hardware interrupt levels;
- activating programs under control of the real time clock or a timer;
- sharing a software level between several programs (time-slicing);
- reentrant subroutines;
- dynamic loading of programs into a special partition where they can be executed with any software priority.

The Basic Real Time Monitor is punched- tape oriented, but it can be expanded to be able to handle other peripheral units as well, disc excepted. The user can, at system generation time, define his own system, choose the right modules for his Monitor, link them and merge them into one single module.
The minimum configuration on which a Basic Real Time Monitor can be used, must consist of:

- operator's typewriter with punched tape equipment
- 8k words of memory.

PRIORITY STRUCTURE

INTERRUPT SYSTEM

In Part I, Hardware, the basics of the interrupt system have already been explained.
When a program is running on a P855 or P860, its priority level is held in a 6-bit register, the PL (priority level) -register. This register can hold any number from 0 to 63. There are 64 levels which can generate an interrupt. Of these, 0 has the highest, 63 the lowest priority.

The levels 0 to 47 are all hardware interrupt lines.
The levels 48 and 49 are reserved for the system (monitor).
The levels 50 to 62 are software levels, i.e. can be assigned to user tasks.
The level 63 is reserved for programs which are executed in the background area of memory.

Interrupt Levels
The levels 0 to 47 are attached to hardware interrupt lines. These can be attached to handle two kinds of interrupts: internal and external interrupts.

Internal interrupts
Standard internal interrupts are:

- control panel
- illegal code interrupt
- stack overflow.

Others depend on the user's options. For example, they may include:

- memory protect
- power failure/automatic restart
- real time clock.

These internal interrupts can be attached to the hardware interrupt level lines as desired.

External interrupts
External interrupts are signals from I/O devices requesting exchange of data or status information. They are usually generated by device control units, each control unit having an individual interrupt line.

Every interrupt level program must be written in master mode and use only the registers A1 through A8. When an interrupt occurs, the program must save the contents of register A1 through A8 of the interrupted program in the stack. Only after this has been done, the interrupt routine can make use of these registers.
To be able to connect an interrupt routine to a hardware level of a line (0 to 47), the location associated to the interrupt line must first be filled with the starting address of the interrupt routine by means of the directive AORG (see Autocode).
In case a device is connected to the single interrupt line which can handle up to 16 signals (see Hardware Part), the associated location which is used by the interrupt handler depends on the connection of the single interrupt line.
As always, programs attached to one of the levels 0 through 47 can only be interrupted by higher priority interrupts, i.e. an interrupt routine connected to an interrupt level can only be interrupted by another hardware interrupt of a higher priority, not by software level interrupts.

Software Priority Levels
The software priority levels 50 to 63 are attached to user tasks and permit multiprogramming between programs.
To each level, a task is associated. Each task consists of all the programs running on that level. Such a task is started by a special monitor program, the dispatcher. The dispatcher divides the

central processor time between the programs attached to the priority levels. It puts the level number (from 50 to 63) of the user task which is to be started into the PL-register. Each time the dispatcher has to change the running priority level, e.g. because of operator intervention, a monitor request, an I/O interrupt, it will select the task with the highest priority (lowest level number) and transfer control to that task.
Level 63 is always assigned to programs running in the background area of memory, if used.
Any hardware interrupt level necessarily takes priority over the software levels.

Note: The levels 48 and 49 cannot be assigned by the user, for they are used by the system.

STACK

The stack is the area of memory behind the bootstrap, interrupt locations and monitor and is used for saving PSW, instruction counter (AO) and any necessary registers and parameters when a program is interrupted. The Program Status Word and the instruction counter are saved by hardware, the other registers and parameters must be saved by software, by the interrupt routine.
The beginning address of the stack area must be defined at SYSGEN and is loaded into the A15 register by the monitor. This register is always protected. In the section Programming Conventions of this Part, it is explained, how this starting address is calculated.
The stack area is filled towards the dedicated area of memory. When the stack reaches location 128_{10}, a 'stack overflow' interrupt is generated.

MEMORY ALLOCATION

Basically, the memory area remaining after the bootstrap, interrupt locations, monitor and stack, is divided into two parts: the Foreground Area and the Background Area.

FOREGROUND AREA

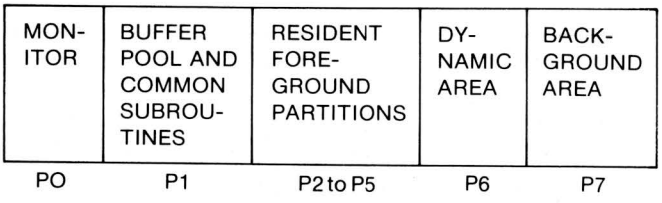
The foreground area is reserved for programs making use of the real time facilities of the monitor, i.e. programs attached to one of the levels from 50 to 62.
Thus, the monitor can supervise the execution of up to 13 user tasks in the foreground area. As only one activity at a time can use the central processor, these tasks are attached to priority levels. If, while one program is running, the monitor is called by a program of higher priority, the running program is interrupted and resumed only after the higher priority program has been attended to.
The interrupt routines in system mode also run in the foreground area.

BACKGROUND AREA

This area is reserved for programs which do not make use of the real time facilities, i.e. the processors (assembler, Fortran compiler, Debug, Update, Text Editor) or programs to which the user has assigned the lowest possible priority.
The level attached to programs running in this area is 63.
This level is handled like any other software level by the monitor.
Background programs can later be attached to foreground levels.

PARTITIONS

The memory can be divided into as many as 8 partitions:



P0 is the dedicated memory area, containing bootstrap, interrupt locations, monitor and stack.
P1 is a partition which can be used to reserve space for user buffers and subroutines used by various tasks (reentrant subroutines).
P2 to P5 are resident foreground partitions, containing the resident user programs making use of the real time facilities.
P6 is a dynamic area, where non-resident foreground programs can be loaded and executed. For example, large programs which rarely have to be executed can use this area.
P7 is the background area, for programs of the lowest priority (level 63).

Each partition must consist of at least 1k words of memory. If a larger partition is needed, it can be expanded with blocks of 1k words. The user does not have to define all the partitions mentioned above, i.e. he can work without Foreground, or without Background or with only one foreground partition, and so on.

All the user programs loaded into the same partition are allowed to work within the boundaries of that partition during a run. All have the same memory protection mask. To allow communication between programs in different partitions, this memory protect is a protection against writing only: any program can read data which are located in any other partition.

Buffer Pool and Common Subroutines Partition
This area can be used to allocate buffer space dynamically by means of the monitor request 'Get Buffer'. This space can be released by the monitor request 'Release Buffer'.
Reentrant subroutines, i.e. subroutines which can be called by various programs, must also use the monitor request 'Get Buffer' to obtain memory space in this partition.

Dynamic Area
By means of the monitor request 'Activate', the user can ask the monitor to load and execute a program in this area. The request is queued, if necessary, and handled on a first-in-first-out basis. The program will be loaded when it becomes the first

one in the queue and, when its execution is finished, it will be erased by the program loaded after it.

PROGRAMS

TASKS

A task consists of one or more programs using central processor time on the same priority level. These programs have been attached to a priority level before they are activated. This connection can be made by means of an operator control message or a monitor request. Programs required for a real time process may be used by one or more tasks. They are loaded into memory as separate programs, the connections between programs used by any one task being supplied through monitor requests in the individual programs. Central processor time is shared between various tasks, and between programs of the same task by time-slicing ('Switch inside a software level' monitor request).

A program comes under control of the monitor and remains so from the moment when all the information about it has been recorded into the Program Control Table (after loading) until it is terminated. During this time, the program passes through various states, as recorded in the Status Word of the Program Control Table.

- inactive: the program has been connected to a level, but it has not been called yet.
- active: the program has been called, but is not yet terminated.
- wait for execution: the program is ready to use central processor time when it receives priority from the dispatcher.
- wait for an event: the program gives up control voluntarily. It has requested an external event, e.g. I/O and waits for that event to occur.

When a program is in Pause state, it is waiting to be started by the operator.

PROGRAMS

The monitor considers a program as a unit to be loaded into memory. Such a program must be relocatable and may not make any external references.
Before it can be started by the dispatcher, a program must first have been connected to a priority level. Then it can be activated in one of the following ways:

- by an interrupt (interrupt level programs)
- by operator control message ST (software level programs)
- by monitor request 'Activate' (software level programs).

There are two types of user programs: interrupt routines and software level programs.

Interrupt routines
These programs are called through an interrupt and run in master mode, on a hardware interrupt level (0 to 47).
Interrupt routines must use the registers A1 to A8 and provide for saving their old contents in the stack.
Interrupt routines are connected to their priority levels as follows:
Each interrupt is associated with a memory location, connected to a hardware line, and containing the start address of the routine. This address must be loaded into this location by the user through an AORG directive.
An example of an interrupt routine is a routine simulating an optional instruction, e.g. MUL, DADD, which is not provided for in a particular configuration (illegal code interrupt).
Interrupt programs must be resident in memory.

Software Level Programs
These programs must be connected to a level from 50 to 63 and are activated through a monitor request or operator control message, after which the dispatcher starts them according to their priority.
The monitor keeps a Program Control Table (PCT) containing all information about software level programs which have been or will be loaded into memory. This table has the following layout:

word 0	PRO-
word 1	GRAM
word 2	NAME
word 3	START ADDRESS
word 4	SAVE ADDRESS
word 5	STATUS
word 6	ECB ADDRESS
word 7	ECB ADDRESS
word 8	PCT ADDRESS
word 9	SCHED. LAB. ADDRESS
word 10	CHAINING LINK
word 11	CHAINING POINTER
word 12	ECB ADDRESS

where:

- word 0, 1, 2 contain the program name.
- word 3 contains the start address of the program.
  If bit 15 = 0: the program runs in master mode.
  If bit 15 = 1: the program runs in user mode.
- word 4 contains the address of a save area of 16 words which has to be reserved in front of each program to save the contents of PSW, A0 (instruction counter) and register A1 through A14.
  If bit 15 = 0: this is a main program.
  If bit 15 = 1: this is a scheduled label.

- word 5 contains the program status.
- word 6 contains the address of the Event Control Block, if the program performs a Wait monitor request.
- word 7 contains the address of the ECB, if the program has activated another program and is waiting for its EXIT.
- word 8 contains the address of the Program Control Table of the activated program.
- word 9 contains the address of the first 'scheduled label' routine in this program.
- word 10 contains a chaining link to a block which, in turn, contains parameters of programs in the queue, waiting for activation.
- word 11 contains the address of the buffer reserved by the Get Buffer monitor request.
- word 12 contains the ECB address for which a Scheduled Label sequence may wait.

These software level programs must be resident in memory and cannot be reentrant.

Reentrant Subroutines

Reentrant subroutines must make use of the monitor request 'get Buffer' to obtain memory space. This memory space must then be reserved in the partition behind the dedicated memory area. This partition is reserved for user buffers and reentrant subroutines. All the reentrant subroutines must be loaded into this area at Monitor loading time. It is not possible to add new reentrant subroutines dynamically.
A program can call a reentrant subroutine by using the instruction CF A14, ROUTINE NAME. If this subroutine itself wants to call another reentrant subroutine, it must reserve 2 extra words in the buffer which it has created through the monitor request 'Get Buffer'.
In reentrant subroutines, scheduled labels (see below) may not be used.

Scheduled labels

The user can make a request for certain functions through Monitor Requests. Such a Request consists of loading necessary parameters in A7 and/or A8 registers, giving an LKM instruction and a DATA directive followed by a number. If this number is negative, this indicates that the user is scheduling a label upon completion of a request. This label identifies a user routine, to which control is given then. In this request the user can for example, analyze the results of a Monitor Request.

For example:

```
LDK A7 PARAM1
LDKL A8, PARAM2
LKM
DATA -1
DATA LABEL
```

Having loaded the A7 and A8 registers with the necessary parameters, a Monitor Request is given with the DATA directive containing the two's complement of the number identifying a specific Monitor Request, in this case the two's complement of 1, i.e. -1. This indicates an I/O Request.
Because DATA contains the two's complement of this number, this means that a label is scheduled following the request: DATA LABEL. Upon completion of this I/O request control is given to the program part attached to LABEL. Thus, the main program is interrupted, its registers are saved and the user routine attached to LABEL is started with these values of the registers. When the scheduled label routine gives an EXIT Monitor Request, control is returned to the main program.

Such a scheduled label routine cannot be interrupted by another scheduled label routine. They are recorded in a list and when one routine exits, control is given to the next one according to 'last-in-first-out' method. At any time during execution, up to seven scheduled labels may be recorded in that list. The program may, of course, contain more.

In each program which may use this 'scheduled label' facility, the user must reserve 24 words at the beginning of the program, as follows:

```
      IDENT
      RES     24
SA    -
      -
      coding
      -
      END     SA
```

FILE CODES

File codes are assigned to the various devices and input/output used, to provide for distinguishing and addressing them.
Each file code consists of 2 hexadecimal digits, of which the following combinations are standard codes.

X'01': Standard Source Input.
X'02': Standard Listing Output.
X'03': Standard Punch Output.
X'04': Standard Object Input.
X'05': Operator's Typewriter (Input and Output).
X'0E': Standard Library Input.
X'0F': Standard Program Input.

REAL TIME CLOCK - TIMERS

Syntax	Meaning	Page
AB␣[PRNAME]	Abort a program	23
AS␣FC,DNXX,[F]	Assign a file code	20
CL␣PRNAME,L	Connect a program to a level	22
CT␣PRNAME,NTIM,[NC],[PR]	Connect a program to a timer	21
DL␣PRNAME,L	Disconnect a program from a level	22
DM␣BEG,END	Memory dump	23
DP␣[P1]/[P2],[P3],[P4],[P5]/[P6]/[P7]	Define partitions	20
DT␣PRNAME,NTIM	Disconnect a program from a timer	21
HD	Halt dump	23
LD␣[RA],[NP],[L],[M]	Load a program	21
LL␣PRNAME,[<fcode>],[RA],[NP],[L],[M]	Load a program from a library	21
MC␣DA,<software order>	Manual device control	23
PS␣[PRNAME]	Pause	22
RD␣DA	Release device	23
RP␣NP,[D]	Release programs	20
RS␣[PRNAME],[NEWA7]	Restart a program	22
RY␣DA	Retry an I/O operation	22
SC␣HH,MM,SS	Set clock	20
SD␣DD,MM,YY	Set date	21
ST␣[PRNAME]	Start a program	22
WM␣COAD,VAL1␣[VAL2␣.....VALn]	Write into memory	23

OPERATOR CONTROL MESSAGES

There are a number of control messages which the operator can enter via the typewriter. To do so, he must first press the INT button on the control panel. The system then types out M: and the operator can type a control message. Such a message consists of 2 characters, identifying the message, followed by a space, possibly followed by one or more parameters. These parameters are separated by commas. Some of these parameters are optional. When a parameter at the end of a message is not used, the last parameter may not be followed by a comma, e.g.

CC PAR1

When a parameter which is not located at the end of a message remains unused, it must always be replaced by a comma, e.g.

CC Par1, (Par2), Par3 or CC Par1,,Par3

Note:

- Every numerical value used in an operator control message must be a hexadecimal value.
- If a message contains an error, the system types out a message ER followed by a number identifying the error. The operator then may press the INT button and type in a correct message.

INITIALIZATION

Define Partition

syntax: DP␣[P1]/[P2],[P3],[P4],[P5]/[P6]/[P7]

use: This message is used to define the number of memory paritions and their lengths. Note, that programs in one partition cannot modify programs in another partition.

From 1 to 7 partitions can be defined:

P1: number of pages of 1k words for the common subroutine and buffer pool partition.

P2 to P5: number of pages of 1k words for resident foreground partitions.

P6: number of pages of 1k words for the dynamic foreground partition.

P7: number of pages for the background partition.

This message can be accepted only before loading any programs.

Examples:

DP 3/2, 1/4/2

The memory will be partitioned as follows:

P1	P2	P3	P4	P5
3k	2k	1k	4k	2k

DP 3/2//6

This message will define the following partitions:

P1	P2	P3
3k	2k	6k

(no dynamic foreground partition)

DP 3/3, 2/4/

This results in the following partitions:

P1	P2	P3	P4
3k	3k	2k	4k

(no background partition)

DP ///5

This results in:

P1
5k

(background partition only)

Relase programs

syntax: RP␣NP,[D]

use: This message causes all programs loaded in partition NP to be released.

NP is a number from 1 to 7 indicating the partition number.

D if set, specifies that the preceding partition is to be extended, i.e. partition NP is to be added onto partition NP - 1.

If a program, connected to a certain level, belongs to the partition released, it is automatically disconnected from that level. This message must not be used while a program is running in that partition.

Example:

RP 3,D

This message specifies that all programs of partition 3 are to be released; the partition is destroyed and its memory space is added onto partition 2.

Assign a file code

syntax: AS␣FC,DNXX,[F]

use: By means of this message a new file code can be created or the assignment of a previously created file code can be modified.

The parameters have the following meaning:

FC : file code to be assigned, 2 hexadecimal characters between 01 and 7F.

DN: device name, 2 ASCII characters.

XX: physical device address, as defined at system generation time.

F : if specified, this file code is reserved specially for the foreground.

The following device names can be used:

TY : operator's typewriter
PR : punched tape reader
PP : tape punch
LP : line printer
CR : card reader
CP : card punch
MT: magnetic tape
MC: magnetic tape cassette
NO: no device; an operation on this file will have no effect.

Example:

AS 08,LP02,F

File code 08 is assigned to the line printer with physical address 2 and it will be used only by foreground programs.

Set clock

syntax: SC␣HH,MM,SS

use: This message is given to initialize the clock.

HH indicates the hour (a value from 0 to 23)
MM indicates the minute (a value from 0 to 59)
SS indicates the second (a value from 0 to 59)

The value typed in by the operator will be automatically incremented by the monitor through real time clock interrupts. When it reaches 00 hours, 00 minutes, 00 seconds, a message will be sent to the operator to update the date (if that option is present).

Example:

SC 3,45,2

The clock is initialized with the time 3 hours, 45 minutes, 2 seconds.

Set date

syntax: SD␣DD,MM,YY

use: This message is used to specify the date. It will not be updated by the monitor (see Set Clock message).
DD indicates the day (a value from 1 to 31)
MM indicates the month (a value from 1 to 12)
YY indicates the year (a numeric value of 2 characters)

Example:
SD 21,6,71
The date is specified as June 21, 1971.

Connect a program to a timer

syntax: CT␣PRNAME,NTIM,[NC],[PR]

use: By means of this message the program PRNAME can be connected to
- the timer with the number NTIM, to be reactivated every PR cycles of that timer; or
- the absolute time HH MM SS, to be reactivated every PR cycles of the timer with the number NTIM.

PRNAME is the name of the program which is to be connected.
NTIM is the number of the timer (0 is assumed to be the real time clock).
NC is the number of cycles of the timer before the first activation (default value is 0) or, if the time of activation is to be given in absolute form, this has to be specified here as follows: HH␣MM␣SS
PR is the pulse rate, i.e. the number of cycles of the timer between two activations. Default value is 0.

A program can only be activated if it has been connected to a level first.

Examples:
CT PROG,1,3,4
The program named PROG is connected to timer number 1. The number of cycles before it is activated for the first time is 3 and it will be reactivated every 4 cycles of the timer.

CT PROG,1,04 12 15,3
The program named PROG is connected to the absolute time 4 hours, 12 minutes, 15 seconds and will be reactivated every 3 cycles of time number 1.

Disconnect a program from a timer

syntax: DT␣PRNAME,NTIM

use: This message disconnects the program named PRNAME from timer NTIM.
PRNAME is the name of the program to be disconnected.
NTIM is the number of the timer from which that program is to be disconnected.

Example:
DT PROG,3
The program PROG is disconnected from timer number 3.

LOADING

Load a program

syntax: LD␣[RA],[NP],[L],[M]

use: This message causes a program to be loaded in the partition specified. It is loaded from the standard object input device and connected to a level L (if specified).
NP is the number of the partition into which the program is to be loaded. If not specified, the program will be loaded into the background area. In this case, the parameters L and M are irrelevant and need not be specified.
RA displacement value, relative to the beginning address of the partition, in hexadecimal. If NP is specified, the program is loaded RA words after the beginning of the partition. If RA is not specified, the program is loaded behind the last program in partition NP.
M if specified, indicates that the program to be loaded is written in master mode. If not specified. the program is written in user mode.
L specifies the software level to which the program is connected (two hexadecimal characters from 32 to 3E).

Every time a program is loaded, the program identification, its loading address and its length are printed out on the typewriter. If a program is not connected to a level, a message is output on the typewriter:
PRNAME NOT CONNECTED.

Examples:
LD
The program is loaded in the background area.

LD ,2
The program is loaded in partition 2 behind the program loaded last.

Load a program from a library

syntax: LL␣PRNAME,[<file code>],[RA],[NP],[L],
[M]

use: By means of this message a program can be loaded from a library, e.g. a cassette tape.
PRNAME is the name of the program to be loaded (up to 6 characters, the first one of which must be a letter).
file code is a number of 2 hexadecimal digits specifying the logical number of the file which contains the library.
NP, RA, M and L have the same meaning as in the LD message.

Connect a program to a software level

syntax: CL␣PRNAME,L

use: This message is used to connect a program to a software priority level.
PRNAME is the name of the program which is to be connected (up to 6 characters, the first one of which must be a letter).
L specifies 2 hexadecimal characters (from 32 to 3E), indicating one of the software priority levels 50 to 62, to which the program is to be connected. A background program is automatically connected to level 63.

The software level (from 50 to 62) will start with
- the operator message ST; or
- a monitor request 'Activate';

Example:
CL PROG,34
The program named PROG is connected to level 52 (hexadecimal 34).

Disconnect a program from a software level

syntax: DL␣PRNAME,L

use: This message causes the program specified to be disconnected from its software level.
PRNAME is the name of the program to be disconnected.
L is the software level number (2 hexadecimal characters) from which the program is to be disconnected.

If the program was connected to a timer, it is disconnected from that.
This request for disconnection is accepted only if the program specified has performed its exit first (in particular I/O operations).

Example:
DL PROG,34
The program named PROG is disconnected from software level 52, to which it had been previously connected.

EXECUTION

Start a program

syntax: ST␣[PRNAME]

use: This message activates the program named PRNAME. If no name is specified, it is assumed to be the background program.
This message is valid only if the program has previously been connected to a software level. The activated program will be handled by the dispatcher and start with its own priority.
It is not possible to activate an interrupt level with this message.

Example:
ST PROG
The program with the name PROG is started.

Pause

syntax: PS␣[PRNAME]

use: This message causes a temporary halt of the program PRNAME, running at the software level connected to it. If no name is specified, the background program is stopped.
To restart the task, the operator must type RS.

Example:
PS PROG
The program PROG is put in pause state.

Restart a program

syntax: RS␣[PRNAME],[NEWA7]

use: This message causes a program, stopped temporarily by a PS message, to be restarted.
PRNAME is the name of the program which is to be restarted. If no name is specified, this is assumed to be the background program.
NEWA7 is an operator's answer which can be loaded in register A7 only in case a program has been stopped by monitor request.

Example:
RS PROG,81E
The program PROG, which has previously been stopped by a Pause request, is restarted and will find the value X'081E' in the A7 register.

Retry an I/O operation

syntax: RY␣DA

use: When the monitor has typed out an error message following an I/O operation, requesting operator intervention, the operator may type this message to retry that same I/O operation, after he has taken any necessary steps. (See Error Handling).
DA is the device address where the operation has to be retried (2 hexadecimal characters).
If the operation succeeds now, control is returned to the user with the status in the DECB. If it still does not succeed, a new error message may be typed out by the monitor if there is another possibility for a retry.

Example:
RY 02
Retry the last I/O operation on the device with physical address 02.

Release Device

syntax: RD␣DA

use: When the monitor has typed out an error message after an I/O operation and requests operator intervention, the operator can type this message if he wants to release the operation on the device which resulted in the error message. (See Error Handling).
DA is the physical device address (2 hexadecimal characters).
Control is returned to the user with the error status in the DECB.

Example:
RD 02
Release the last I/O operation on the device with physical address 02.

Manual device control

syntax: MC␣DA,<software order>

use: This message can be given if the operator wants to do a manual operation on a device.
DA is the physical device address (2 hexadecimal characters).
<software order> : to be defined.

Memory Dump

syntax: DM␣BEG,END

use: This message causes a memory dump from the first address (BEG) to the second address specified (END). This is done on the print file, usually the line printer.
BEG is the beginning address of the dump (up to 4 hexadecimal characters).
END is the ending address of the dump (up to 4 hexadecimal characters).

Example:
DM 4F,3FE
The memory will be dumped from hexadecimal address 4F to 3FE.

Halt Dump

syntax: HD

use: If this message is given, a dump which is being output can be stopped. This can be very useful if the dump is taking place on the typewriter, as this is a very slow device.

Write into memory

syntax: WM␣COAD,VAL1␣[VAL2␣.....VALn]

use: This message can be used to correct one or more memory locations.
COAD is the first location which is to be corrected (hexadecimal address of up to 4 characters).
VAL1, VAL2, ...VALn are values which are to be entered in the memory locations starting from COAD, i.e.
VAL1 is put in location COAD
VAL2 is put in location COAD + 2, etc.

Example:
WM 4FE,44F 3FE4
The value 44F is placed in memory location 4FE.
The value 3FE4 is placed in memory location 500.

Abort a program

syntax: AB␣[PRNAME]

use: This message definitely stops a program which is running at a given level.
PRNAME is the name of the program which is to be aborted. If no name is specified, the background program is aborted.
The devices connected to this program are detached and the buffers are deallocated.

Example:
AB PROG
The program named PROG is aborted.

MONITOR REQUESTS

The user program can make requests to the Monitor for specific functions. This is done by first giving an LKM (Link to Monitor) instruction and the DATA directive with a number as operand. This number indicates specifically the function requested from the Monitor.

Preceding these requests, certain parameters may need to be loaded in the A7 and/or A8 registers.
Following each request, the system gives the result in the A7 register. In the following paragraphs the function and calling sequence of each request is described.

It is possible to include Scheduled Labels in a monitor request (see General Concepts).

I/O Requests
Calling Sequence:

```
LDK    A7,CODE
LDKL   A8,ECBADR
LKM
DATA   1
```

where CODE must be specified as follows:

		W	R	ORDER	
0	7	8	9	10	15

ECBADR is the address of the Event Control Block, containing the parameters for the requested I/O function.

Use:

Through this request the user can ask the system to start a certain I/O operation on a peripheral device. The parameters loaded in the A7 registers have the following meaning:

Bits 8 and 9 specify the mode of operation:

W = 1: the requesting program wants to wait for the completion of the I/O operation requested. Only after completion of the requested function, the return to the calling program will take place.

W = 0: a return to the calling program will be made as soon as the request has been sent to the channel. The program will give a Wait request later on for synchronization.

R = 1: The program itself will process any abnormal condition concerning the requested operation. The system will return the Hardware Status in ECB word 4.

R = 0: Any abnormal conditions will be processed by the system. The Software Status is returned in ECB word 4.

ORDER specifies which I/O function is requested, by giving one of the following combinations of 2 hexadecimal digits.

01: Basic Read.

05: Basic Write.
For Basic (Binary) I/O requests the sytem does not provide for character checking or data conversion, only for control command initialization and end of operation signals.

02: Standard Read.

06: Standard Write.
Standard (ASCII) I/O requests provide, by means of standard conversions, for special features such as error control characters, conversion from external code to internal ASCII and vice versa. There is a checksum and characters are stored seven by seven bits, two characters to a word.

07: Object Write (4×4×4×4 tape format)

08: Object Write (8×8 tape format)
Object I/O requests provide, by means of standard conversions, for special features such as error control characters, checksum and data conversion form external 4×4×4×4 or 8×8 tape format to internal 16-bit format.

14: Skip to next EOS mask.

16: Skip to next EOF mark.

22: Write EOF mark.

26: Write EOS mark.

30: Get information about file code.

For 02, 07 and 08 at least 8k words of memory are required.

Fore each of these request orders, the following specific information applies to the various peripheral devices:

Basic Read (X'01'):

Operator's typewriter: All characters are entered on 7 bits until the requested length is reached.

ASR tape reader: Same as for the keyboard. The reader stops one character after an X.off code is read.

High speed tape reader: All characters are entered on 8 bits, without checking or special features, until the requested length is reached.

Card reader: All the words are entered and stored in Hollerith code on twelve bits (4 to 15). In each word the column image is right justified. The words are stored until the requested length is reached. The length is given in words.

Basic Write (X'05'):

Operator's typewriter: All characters are output without checking or special features. This order can be used to print something and have the answer on the same line.

ASR tape punch: Same as for printed output.

Line printer: All characters are output without checking. There is no control character.

High speed tape punch: All characters are output without checking or special features.

Standard Read (X'02'):

Operator's typewriter: ASCII characters are entered on 7 bits, with the following special features:

- the special characters, coded from X'0' to X-1F' (hexadecimal), are ignored.
- code X'7F' (Rub-Out or Delete character) is ignored.
- code X'5F' (←) can be used or delete the preceding character. If more ← are used consecutively, an equal number of preceding characters will be deleted.
- code X'5E' (↑) is used to delete the line preceding it, up to the next carriage return.
- code X'0D' (carriage return) indicates end of block. It is the last character to be entered. It is not transmitted to the user's buffer.
- code X'5C' (\) is used as a tabulation symbol (see ECB-word 5). If the address of the tabulation table is zero, or if the number of tackets is zero, or if the storage address is greater than the last tacket, the code X'20' is stored in the buffer. In other cases, X'5C' is not stored and replaced by spaces, as indicated by the tackets in the tabulation table.

ASR tape reader: For ASCII characters, the same features apply as for the keyboard: the code for carriage return must be preceded by the code X-off.
For object code in 4×4×4×4 tape format, the first character identifies the object format. It must range from X'18' to X'1F' and is converted to a number from X'0' to X'7' and stored on one character. The second character contains the word-length of the input block, excluding the first word and the checksum. Each punched row (4 bits) entered after this identifier is stored on one half-character up to the checksum. When the checksum has been read, input is stopped. The 8×8 tape format cannot be read on the ASR tape reader. To start the reader, an X-on code is sent by the system before entering the characters.

High speed tape reader: Same as for ASR tape reader. In addition: for object code in 8×8 tape format, the first character, identifying the object code format, must have one of the following values: X'10', X'1' to X'4' or X'15' to X'17'. It is converted to a number from X'0' to X'7'. Each punched row (8 bits) entered after this identifier is stored on one character up to the checksum. The second character is the length of the block, in words, excluding the first word and the checksum.

Card reader: All words are read in Hollerith code, on 12 bits, converted and stored in ASCII code, on 7 bits, until the requested length is reached. Words which are not in Hollerith are converted into the ASCII code for X'20' and a 'data fault' status is returned in the software status (ECB word 4 : bit 13 is 1). There is no special code. However, EOS and EOF marks are detected (bits 14 and 15 in software status).

Standard Write (X'06'):

Operator's typewriter: All characters, except X'0' to X'1F' (special code characters) are output without checking. At the end of a line, a carriage return and line feed are output.

ASR tape punch: Same as for ASR tape punch with the following exception: the first character is a control code: if it equals X'30' or X'31', it is output as Line Feed; if it is different it is not output.

Line printer: All characters are output without checking, except for the control code. It must be stored in the word preceding the buffer address. This control code may have three values:
+ → X'2B': print the line without advancing the paper (superposition).
0 → X-30': advance two lines before printing.
1 → X-31': skip to top of page before printing.
All other control codes are used as normally: advance one line and print.
At the end of the buffer, after the requested length, one word must follow to be used by the used for a print code.
If the requested length is more than one line, the system puts a print code after the maximum length and the buffer will be printed on two or more lines.

Object Write (4×4×4×4 tape format: X'07'):

ASR tape punch: The first character is output on one row, converted from X'0' - X'7' to X'18' - X'1F'. Each following character is output on two rows; to avoid special (ASCII) code each row is converted. The second character contains the length of the block in characters excluding the first character. At the end an 8-bit checksum is performed and punched, followed by an X-off code.

High speed tape punch: Same as for ASR tape punch, except that the second character contains the length in words.

Object Write (8×8 tape format: X'08'):

High speed tape punch: The standard object code is output in 8×8 format, where the first character is a format character and is output on one row, converted as follows:
X'0' → X'10'.
X'1' to X'4' → X'11' to X'14'.
X'5' to X'7' → X'15' to X'17'.
The second character contains the length in words, excluding the first word. An 8-bit checksum is performed and punched.

Write EOF mark (X'22'):

For *operator's typewriter* and *ASR tape punch* an end-of-file mark is output as follows: :EOF LF-X-OFF-CR-R-out.

High speed tape punch: Same as for ASR.

Line printer: An End-Of-File is output: :EOF

Write EOS mark (X'26'):

For *operator's typewriter* and *ASR tape punch*, an end-of-segment mark is output as follows: :EOS LF-X-off-CR-R-out

High speed tape punch: Same as for ASR.

Line printer: An End-Of-Segment is output: :EOS

Read up to End-Of-Segment (X'14'):

High speed tape reader: The tape is read until an :EOS statement has been read.

Card reader: the cards are read until an :EOS statement is read.

Read up to End-Of-File (X'16'):

High speed tape reader: the tape is read until an :EOF statement has been read.

Card reader: the cards are read until an :EOF statement has been read.

Return information about a file code (X'30'):
By means of this order it is possible to find out the assignment of a file code. The information will be returned in the Event Control Block:

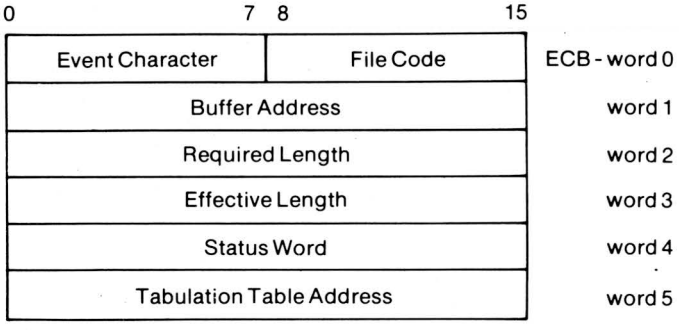
ECB - word 0 : Event Character + File Code.
ECB - word 1 : Device Name (2 ASCII characters):
TY = operator's typewriter (listing).

TR = ASR tape reader.
TP = ASR tape punch.

ECB - word 2 : maximum record size (72 characters available as output).
ECB - word 3 : left character: unused.
right character: device address.
ECB - word 4: Status = 0.

If the file code in ECB - word 0 is set to zero, the other words of the ECB will also contain zeros.

The *Event Control Block* (of which the address must have been loaded in the A7 register) has the following format:



where:

word 0: event character: the monitor stores all the information in the bits of this character:
bit 0 = 1: end of operation has occurred for this ECB.
bit 1 : unused.
bit 2 and 3: not significant for I/O.
bit 4 = 1: file code has not been assigned.
bit 5 = 1: abnormal end of operation.
bit 6 = 1: end-of-segment has been read.
bit 7 = 1: end-of-file has been read.

word 1: address of the user buffer.

word 2: requested length to be read or written (in words or characters).
The first character is always the character given by the buffer address.

word 3: effective length which has been transmitted (in words or characters). This information is stored by the monitor upon completion of the I/O operation.

word 4: Status word, stored by the monitor upon completion of the requested I/O operation.
- For Basic orders, this word will be filled with the hardware status by the control unit. However, if the monitor detects an error in the calling sequence, bit 0 will be set to 1 and the other bits will contain the software status.
- For the other orders the software status will be returned:
bit 0: the operation has been successfully completed.
bit 10 = 1: beginning of tape encountered.
bit 11 = 1: end of input medium.
bit 12 = 1: requested length is incorrect.
bit 13 = 1: illegal character code.
bit 14 = 1: an EOS mark has been read.
bit 15 = 1: an EOF mark has been read.
When the operation was not successfully completed, bit 0 is set to 1 and bit 1 is set to 0 (retry also was not possible). In this case bits 2 to 15 give the hardware status.
When the monitor has detected an error in the calling sequence, bits 0 and 1 are both set to 1 and bits 11 to 15 have the following significance:
bit 15 = 1: illegal file code has been used.
bit 14 = 1: device is attached to another program.
bit 13 = 1: ECB address is illegal.
bit 12 = 1: buffer size or address is illegal.
bit 11 = 1: function is unknown or not compatible with the device.

word 5: tabulation table address for the punched tape equipment.
This tabulation table has the following format:

Number of Tackets	First Tacket
Second Tacket	Third Tacket

etc.

The tackets have an absolute position in the Line. Characters up to the following tacket are filled with blanks.

Example:

3	10
20	30

Input Line: LABEL\OPER\OPERAND\
COMMENT
Line in Buffer:
LABEL␣␣␣␣OPER␣␣␣␣␣␣OPERAND␣␣␣
1 10 20
COMMENT
30
At completion of input, the buffer is filled with spaces, but the returned length is the length effectively entered and stored, including the spaces replacing the tabulation codes (\).
Upon completion of one of these requests, the system responds as follows:
A7 = 0 : request completed.
= 1 : the corresponding processor is not in memory.
= 2 : the requested function cannot be processed.

Wait for an event

format:

```
LDKL   A8,ECBADR
LKM
DATA   2
```

where ECBADR gives the address of the Event Control Block (see I/O requests). The first character of the ECB is the event

character. If the first bit of this character is set to 1, the event has been completed.

use:

This request causes a program to stop and wait for the completion of an event which has to take place in another program (user or system). If the event has occurred, the dispatcher returns control to the requesting program. If the event has not occurred the program is put in wait state, to be restarted when the event has occurred.

Note: A 'scheduled label' routine may **not** contain a Wait request.

Upon completion of the request, the system responds as follows:

Exit

format:

```
LKM
DATA 3
```

use:

This request is used to specify the end of a program. The program exit is effected after completion of all I/O operations and after all labels have been scheduled.

The program is put in inactive state, if no other program has entered a wait request for this one. It remains connected to its priority level, however. If this is a level which can be shared, a switch is implicit in the exit.

Get buffer

format:

```
LDK   A7,LENGTH
LKM
DATA  4
```

LENGTH is a constant specifying the requested buffer length in characters.

use:

This request asks the system to reserve memory space in the pool of buffers defined at generation time in the first memory partition.

The system responses are in:

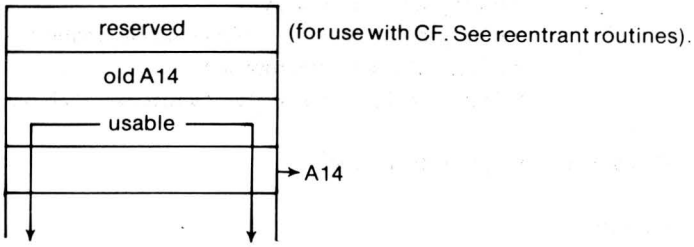
A14: the address of the second word of the reserved buffer. The user contents of A14 is saved by the system and placed in the buffer. This is restored when the user releases the buffer.

A7: = 0: the memory space is reserved for the user in the buffer pool.

= 1: there is not enough memory space for the user in the buffer pool.

= 2: the buffer pool is destroyed and a message is sent to the operator.

The structure of the reserved memory space is as follows:



Release buffer

format:

```
LDK   A7,LENGTH
LDKL  A14,BUFADR
LKM
DATA  5
```

LENGTH is a constant, specifying the length of the memory space to be released, in characters.

BUFADR points to the second available word in the buffer, as given in response to the Get Buffer request.

use:

This request is used to release the memory space reserved previously in the buffer pool by means of a Get Buffer request. The A14 register is restored with the value it contained before this Get Buffer request was made.

The system response is as follows:

A7 = 0: the memory space is released.

A7 > 0: it is impossible to release this memory area because:

- the address is not correct; or
- the length specified is not correct.

Connect a program to a software level

format:

```
LDK   A7,NUMBER
LDKL  A8,PRNAME
LKM
DATA  8
```

NUMBER is the number of the software level to which the program is to be connected.

PRNAME points to a 3-word block containing the name of the program.

use:

A program which may be running at any level, can make a request for another program to be connected to a software level by means of this request.

The system responds as follows:

A7 = 0: the connection has been accomplished.

A7 ≠ 0: it is impossible to make the connection, because:
- the program does not exist;
- the program is not compatible with the requested level, i.e. it is an interrupt level;
- the program has already been connected to a level.

Disconnect a program from a level

format:

```
LDK    A7,NUMBER
LDKL   A8,PRNAME
LKM
DATA   9
```

NUMBER is the number of the software level to which the program has been connected.
PRNAME points to a 3-word block containing the name of the program.

use:
By means of this request, the program PRNAME is disconnected from the level indicated in register A7. If it was connected to a timer, it is disconnected from that.
The system responds as follows:
A7 = 0: the program has been disconnected.
A7 ≠ 0: it is impossible to disconnect the program, because:
- the program was not connected to this level (result: a NO-operation);
- the program is busy.

Connect a program to a timer

format:

```
LDK    A7,PARAM
LDKL   A8,PRNAME
LKM
DATA   10
```

PARAM specifies the necessary parameters in one word as follows:

NC		NTIM		PR	
0	7	8	9	10	15

where: NC is the number of cycles of the timer before the first activation (a number from 0 to 255).
NTIM is the number of the timer (0 is assumed to be the real time clock).
PR is the requested pulse rate, i.e. the number of cycles of the timer between two activations (a number from 0 to 63).
PRNAME points to a 3-word block containing the name of the program.

use:
By means of this request, a program running at a software level can be connected to a timer, according to the parameters given in register A7.
At generation time it is possible to define several software timers whose pulse rates are multiples of the basic interval time, i.e. the real time clock cycle time, i.e. the time between two clock interrupts.
The program must have been connected to a software level, otherwise it can not be started by the dispatcher.
The system responds as follows:
A7 = 0: the connection has been accomplished.
A7 ≠ 0: connection is impossible, because:
- the timer specified has not been defined;
- the program has already been connected to a timer.

Disconnect a program from a timer

format:

```
LDK    A7,NUMTIM
LDKL   A8,PRNAME
LKM
DATA   11
```

NUMTIM is a constant specifying the number of the timer.
PRNAME points to a 3-word block containing the name of the program.

use:
By means of this request a program can be disconnected from a timer specified in register A7. If the program had also been connected to a software level, that connection remains.
The system responds as follows:
A7 = 0: the program has been disconnected.
A7 ≠ 0: it is impossible to disconnect the program, because:
- the timer does not exist;
- the program does not exist.

Activate a program

format:

```
LDKL   A7,PRNAME
LDKL   A8,BLOCK
LKM
DATA   12
```

PRNAME points to a 3-word block containing the name of the program to be activated.
BLOCK points to a 2-word block of the following format:

reserved for the system	(ECB)
parameter block address	

The first word of this block is updated by the system at the exit of the activated program. If the calling program has requested a wait for the activated program, this word must be considered as its Event Control Block (ECB): see the Wait request.
The second word contains the address of a parameter block. If the activated program is in the same partition as the calling program, the activated program can write in this block without problems, because there is no memory protection. If, however, the activated program is not in the same partition, the calling program must, before activating this one, send a Get buffer request to build its parameter block in the buffer pool. In both these cases register A14 will contain the address of the parameter block when the activated program is started.

use:

This request can be made by a program running at any level, to activate a software level program. Depending on their priority levels, both the calling and activated program may be processed concurrently.
If the activated program is busy, the request is recorded in a stack and will be processed in due time.
If the activated program must run in the dynamic foreground area, the loader can be activated to load the program.
The system responds as follows:
A7 = 0: the request has been processed.
A7 ≠ 0: it is impossible to process this request, because the program has not been connected to a level.

Switch inside a software level

format:

```
LDK   A7,LEVEL
LKM
DATA  13
```

LEVEL is a constant specifying the number of the level in which the switch is to be made.
If LEVEL is specified as 0, the level to be switched is equal to the requesting level plus one.

use:

By means of this request it is possible to have time-slicing inside a software level by halting execution of the program running on that level (A7) and giving control to the next program on the same level.

Attach a device to a program

format:

```
LDKL  A7,DEVBLK
LKM
DATA  14
```

DEVBLK points to a 2-word block of the following format:

DN	
	XX

where DN is the device name (2 ASCII characters).
XX is the physical device address (2 hexadecimal characters).

use:

By means of this request a program running at any software level can reserve the use of the device specified to itself; in that case no other program can perform any I/O operations on this device.
The system responds as follows:
A7 = 0: the device has been attached to this program.
If the device does not exist, or if it is impossible to attach it, the program is aborted.
If the device has already been attached to another program, the requesting program is put in wait state until the device specified has been detached (see following request).

Detach a device from a program

format:

```
LDKL  A7,DEVBLK
LKM
DATA  15
```

DEVBLK points to a 2-word block of the following format:

DN	
	XX

where DN is the device name (2 ASCII characters).
XX is the physical device address (2 hexadecimal characters).

use:

By means of this request a device, which has been previously attached to a program through an Attach request, is detached from that program.
To release the device as quickly as possible, this request must be made as soon as the program concerned no longer requires the use of this device.
The system responds as follows:
A7 = 0: the device has been detached.
A7 ≠ 0: it is impossible to detach the device because:
- it is busy with an I/O operation for this program;
- it is attached to a different program;
- the device specified does not exist.

Pause

format:

LDKL A7,MESBLK
LKM
DATA 16

MESBLK points to a block containing a message which is to be printed out on the typewriter. The 1st character must contain the length in characters of the message block.

use:

This request causes a temporary halt of the running program. It is put in wait state and, if specified, a message is printed out on the typewriter. The program can be restarted only by an operator control message RS. When the program is restarted, it may be given an additional parameter in register A7 (see Restart operator control message).

Get Time

format:

LDKL A7,TIMBLK
LKM
DATA 17

TIMBLK points to a 6-word block which will be returned by the monitor upon receiving this request. Its format is as follows:

DAY
MONTH
YEAR
HOUR
MINUTE
SECOND

The values are given in ASCII.

use:

Upon receipt of this request the monitor will print out the date and time.

ERROR HANDLING

When one of the following errors occurs, a message will be printed out on the operator's typewriter:

- an error in an operator control message
- an error during the loading of a program
- an error on a peripheral unit.

Error in an operator control message

The message, printed out in case of such an error, will have the following format:

ER␣<code>

where <code> is represented by 2 hexadecimal digits, with the following meaning:

01: control message is erroneous
02: syntactical error in control message.
03: logical error: one of the parameters in the control message refers to a non-existing item.

To correct the control message, the operator must push the INT button on the control panel. M: is printed out and the operator can type the correct message.

Program Loading errors

The message, printed out in case of such an error, will have the following format:

ER␣<code>

The meaning of <code> will be defined later.

Error on a peripheral unit

In the case of such an error, the system asks for operator intervention on a peripheral device by printing out the message:

PU␣, DNXX, <hardware status> [, RY|RD]

where

DN is the name of the device on which the error occurred.
XX is the device address.
<hardware status> is the status of the device as stored in the Event Control Block by the I/O routine.
RY and RD, if printed, indicate that the operator may retry (control message RY) or release (control message RD) the last operation on this device.

Program abortion

In this case, a message is printed out as follows:

AB␣PRNAME␣LEVEL␣CODE␣ADDRESS␣STATUS

where

PRNAME is the name of the aborted program.
LEVEL is the priority level attached to the program.
CODE will be defined later.
ADDRESS is the address where the abort occurred.
STATUS is the status of the Program Control Table.

A program can be aborted for one of the following reasons:

- illegal operation code.
- memory protect error.
- illegal monitor request (LKM).

If the aborted program was running on a software priority level, the monitor takes the following actions:

- the program is disconnected from its priority level.
- if the program was connected to a timer, it is disconnected.
- if the program was attached to a device, it is detached from it.
- any buffers, reserved by the monitor request 'Get Buffer', are released.

If the aborted program was an interrupt level program, a halt instruction is executed.

Assembly

Programs for the P855 and P860 are written in a low-level symbolic language, in which each instruction corresponds to a single instruction in machine language, possibly extended by a data word.
The process of converting the program from the low-level language (Autocode) to machine language is called assembly, and the program which performs the conversion is an assembler.

There is a Basic Assembler which can be used on any P855 or P860 with 4k or more of memory. If the user has a memory of 8k or more he can use the Extended Assembler, which provides a few additional facilities. The instructions input to this assembler are the same as those for the Basic Assembler, but there are a number of additional directives which are useful in various circumstances.
The details will be described in the following section.

HARDWARE LANGUAGE

The following description of the formats of machine instructions should facilitate understanding of the two Autocodes and the reading of hexadecimal dumps.
There are two instruction formats. Format 1 instructions perform a number of operations by reference to two of the sixteen registers available for user-access; one of these registers may point to a data-item either in the word following the instruction or elsewhere in memory because it is possible to use it as an index-register. Format 0 instructions use one of the first eight registers, and include constant handling, input/output, shift and miscellaneous instructions.

Format 1 instructions:

1	Op. Code	R1	mode	R2	I/s
1	4	4	2	4	1

Bit 0: instruction format (1).
Bits 1 to 4: operation code.
Bits 5 to 8: the number of one of the sixteen registers, expressed as follows:
if the 16 registers are divided into two groups, Group 0 (registers 0 to 7) and Group 1 (registers 8 to 15), then bit 8 shows the group number and bits 5 to 7 show the number of a register within that group; i.e. if bit 8 = 0, bits 5 to 7 indicate registers 0 to 7; if bit 8 = 1, bits 5 to 7 indicate registers 8 to 15.
In branch instructions, however, bits 5 to 7 indicate a condition and bit 8 is not used.
Bits 9 and 10: addressing mode code; this specifies - direct or indirect addressing, - whether the word following the instruction, or another memory word, has to be taken into account, -register-to-register instructions.
Bits 11 to 14: the number of one of the sixteen registers, expressed in the same way as in bits 5 to 8.
Bit 15: a load/store indicator; used in certain instructions to indicate that the result of the operation is to be placed in the register shown by bits 5 to 8 (bit 15 = 0; also the default value) or in a memory word (bit 15 = 1).

Format 1 instructions may be extended by a data word containing an address or a constant value.

Format 0 instructions:

0	Op. Code	R3	
1	4	3	8

Bit 0: instruction format (0)
Bits 1 to 4: operation code
Bits 5 to 7: the number of one of the first eight registers (i.e. Group 0 only); or a condition indicator in a Branch instruction.
Bits 8 to 15: the contents of this field varies according to the type of instruction. It may be an 8-bit positive constant (short constant instruction), an even displacement value (relative branch instructions), an indication of the type of shift required (shift instructions), device address (I/O instructions) or a fixed parameter (certain miscellaneous instructions).

Types of instruction
These two hardware formats can represent a large number of different types of operation.

Format 1
- Register to register instructions; data may be transferred between the registers specified in the R1 and R2 field, or between the register specified in R1 field and a memory address held in the register specified by the R2 field. Load and store, add and subtract, logical and various other operations can be performed.
- Long constant instructions; these are actually a subset of the above-mentioned group, although this is not apparent in the Basic language. The constant is located in the word following the instruction, being referenced by the R2 field which contains zeroes (thus indicating the P-register, which always holds the address of the next instruction or data-word in normal sequence). Most of the same operations as above can be performed.
- Memory reference instructions; data may be transferred between the register specified in the R1 field and a memory word whose address is given in the word after the instruction. This address may be direct or indirect, and can be modified in either case by the contents of an index register specified in the R2 field. All operations which can be done by register to register instructions, and a few more, are possible.

In many of the Format 1 instructions the programmer can specify whether the result is to be placed in the R1 register (L/S bit set to 0) or in the memory word indicated by the instruction (L/S bit set to 1).

Format 0:

- Short constant instructions; a value held in the second half of the instruction is operated upon (add, subtract, logical operations etc.) and relocated in the register specified by the R3 field.
- Shift instructions: the contents of the register indicated in the R3 field are shifted one position in a specified direction.
- I/O and miscellaneous instructions; this group includes a variety of operations connected with input/output and interrupt handling.

Branch instructions are included in both formats. The R1 or R3 field is replaced by a condition indicator, while the address for the branch may be absolute, symbolic or relative.

Addressing modes

Addressing Indicator 4k Autocode		MD field value	R2 field value	Effective operand address	Meaning
(R in mnemonic)	→	00	0 to 15	R2	Second operand is located in register indicated by R2 field.
(K in mnemonic)	→	01	= 0	(P)	Long constant instruction (the word following the instruction is the operand, a 16-bit constant)
(R in mnemonic followed by ⋆)	→	01	≠ 0	(R2)	Address in register: the effective address is held in the register indicated by the R2 field.
-	→	10	= 0	(q)	Direct Address in next word.
Register expression after address expression	→	10	≠ 0	(q)+(R2)	Indexed address; address in next word is modified by the contents of the register specified in R2 field.
⋆ after mnemonic	→	11	= 0	[(q)]	Indirect address: the next word contains the address of the address of a memory word to be operated upon.
⋆ after mnemonic, register expression after address expression	→	11	≠ 0	[(q)+(R2)]	Indirect indexed address: address in next word is modified by the contents of the register specified in R2 field, and the resulting address is indirect.

Key: P: instruction counter (P-register)
q: contents of P (i.e. word after instruction)
R2: bits 11 to 14 of instruction word
( ): contents of
[ ]: indirect addressing.

BASIC AUTOCODE

Basic Autocode is an easily comprehensible symbolic language which enables all information pertaining to one instruction to be included in a single statement line, although this information may sometimes occupy two words in memory. The user does not need to know the format of hardware instructions in detail other than for reading dumps; the syntactical layout given in the instruction set should be a sufficient guideline.

A number of directives are used to guide the Assembly process, and the DATAdirective to generate data within the program. These directives are described in the chapter on directives; any differences in their use by the different versions of the Assembler are indicated there.

Character set

The following characters may be used to write programs in the Basic Autocode:
Letters: A to Z inclusive.
Digits: 0 to 9 inclusive.
Special characters:
+ | − | ⋆ | = | ' | , | ␣ (space) | / | (|) | . | :

Syntax notation

A number of special symbols are used, in this chapter and in the instruction set, to define the syntax of instructions and their components:

⌈ ⌉ the syntactic item(s) between these brackets may be omitted: ⌈L⌉ means that the parameter L need not be specified.

⌊ ⌋ one of the syntactic items between these brackets must be specified: ⌊⋆|<symbol>⌋ means that either ⋆ or a <symbol> must be specified.

| or

< > these brackets contain a syntactical item which must be replaced by the required parameter.

::= is composed of.

These symbols must not be used when writing actual programs.

Format of source statements

<source statements>::=⌊<instruction>|<directive>
|⋆<comment **line**>⌋

A source statement may be either an instruction, or a directive or an independent comment line. The format of directives is described in the chapter on directives.

<instruction>::=⌈<label>⌉␣<operation code>␣⌈<operand>⌉
␣⌈<comments>⌉

The Assembler interprets each statement line using a free form scan, requiring the fields of the instruction to be separated by one or more spaces (max. 10). Consequently, spaces must not be used within the fields, other than in a character constant or the comment field.

If there is an identifier it must start with the first character in the line. If the first character is a space this indicates (or will be assumed to indicate) that there is no identifier field; if the first character is an asterisk (⋆) the line is an independent comment line.

If there are more than 10 spaces after the operation code field, any character following the statement line will be interpreted as belonging to the comment field.

Every statement line on punched tape must be terminated with Xoff-CR-LF.

Label field

<label>::=<identifier>

where:

<identifier>::=<letter>⌈:|<letters>|<digits>⌉

An identifier may be composed of up to 6 characters; the first of these must be a letter, while the rest can be any combination of letters and digits.

Examples: AB2
K
WORD3A

An identifier containing more than 6 characters will be truncated to 6 characters by the Assembler. If the resulting identifier coincides with another, a double definition error will occur.

Identifiers are used to label instructions or data words so that other instructions can refer to them. The Assembler assigns an address value to each identifier specified in the identifier field of an instruction, and fills in this value when the identifier is referred to in the operand of another instruction.
The value of an identifier is normally regarded as relocatable. However, it may acquire an absolute value if

- it is equated to an absolute value by an EQU directive; or
- it appears in an absolute program (defined by an AORG directive) and is not equated by an EQU directive to an identifier previously defined as relocatable.

Operation code

<operation code>::=<mnemonic>⌈S|(CND)|L⌉⌈⋆⌉

An operation code is represented by the mnemonic of an instruction, which may have one or two other indicators attached to it (Format 1 instructions only). These indicators can be:

- S, allowed after the mnemonic of certain register to register and memory reference instructions; indicates that the result of the operation must be stored in a memory word (LS bit set to 1);
- (CND), allowed after the mnemonic of a conditional branch instruction; specifies the condition under which a branch is to be performed; the following conditions may be specified:

CR Contents / Instruction type	= 0	= 1	= 2	= 3
Arithmetic	Z Zero	P Positive	N Negative	O Overflow
Compare	E Equal to	G Greater than	L Less than	-
I/O	A Accepted	R Refused	-	U Address Unknown
CR contents	≠ 0	≠ 1	≠ 2	≠ 3
Arithmetic	NZ Not zero	NP Not positive	NN Not negative	-
Compare	NE Not equal	NG Not greater	NL Not less	-
I/O	NA Not accepted	NR Not refused	-	NU Address not Unknown

Note: Only on the extended version of the assembler is it possible to use these letters; on the basic version, only numeric notation can be used.

- L, allowed after the instruction mnemonic of a constant instruction; specifies that the constant occupies 16 bits, i.e. that the instruction must be assembled as a Format 1 'long' instruction.
- ⋆, in a register-to-register or memory reference instruction, specifies indirect addressing mode (see Hardware Language).

Examples:

```
␣LDR␣A4, A5␣LOAD A5 CONTENTS INTO
⋆A4 REGISTER.
␣CFI␣A3, E:MULT␣CALL E:MULT MODULE
⋆(INDIRECT).
␣AB(P)␣SUB4␣BRANCH TO SUB4 CONTENTS IF
⋆POSITIVE.
␣ADS␣A7, ADDEND␣ADD A7 CONTENTS TO
⋆ADDEND CONTENTS AND STORE RESULT IN
⋆ADDEND.
␣SUKL␣A1, X'OBCF' SUBTRACT X'OBCF' FROM A1
⋆CONTENTS.
␣SLA␣A4␣SHIFT A4 LEFT ARITHMETICALLY ONE
⋆POSITION.
␣OTR␣A2, 0, 02␣OUTPUT A2 CONTENTS TO
⋆DEVICE 02.
```

Operand expressions

A large number of expressions can be used in the operand field, the one selected depending on the type of instruction.

Register to Register instructions

With the exception of branch instructions, this type of instruction will contain two register expressions in its operand, written as follows:

- Register 0 (instruction counter or P-register):P;
- Register 1 to 14: A1, A2, A3.....A14;
- Register 15 (stack pointer): A15.

As registers 0 and 15 have a special purpose in addition to being general-purpose registers, users must take great care not to alter their contents unintentionally.
The register expressions must be separated by a comma.

The first expression relates to the R1 field, the second to the R2 field. The register specified by the R2 field may be used to hold the address of the data item to be operated upon, rather than the data item itself; this is a form of indirect addressing, and must be indicated by attaching an asterisk to the instruction mnemonic.

Examples:

```
␣LDR␣A1, P␣LOAD P-REGISTER CONTENTS INTO
⋆A1 REGISTER
␣ANR⋆␣A5, A6␣PERFORM LOGICAL AND BETWEEN
⋆CONTENTS OF A5 AND MEMORY WORD
⋆ADDRESSED BY A6.
␣SUR␣A2, A3␣SUBTRACT CONT. OF MEMORY
⋆WORD ADDRESSED BY A3 FROM A2 AND STORE
⋆RESULT IN MEMORY WORD.
```

Memory Reference Instructions

The operand of a Memory Reference instruction contains the internal number of a register to be used in the operation (corresponding to the R1 field) and the address expression, which may consist of a number of terms. It may also include the number of another register, whose contents are used to modify the memory address indicated.

Address expression may contain any of the following terms, or a combination of them:

- *: this is a predefined expression representing the current value of the location counter, a software counter maintained by the Assembler to hold the address of the current instruction; this counter is incremented by 2 or 4 or as many times as the number of words produced by the instruction, after each instruction (depending on the length of the instruction) to simplify the loading procedure, as the bit-positions of the P-register correspond to bits 0 to 14 of the other registers, bit 15 being ignored.
- a symbol: this has the same form as an identifier, and is used to refer to an instruction or data word which has the same identifier in its identifier field; the Assembler will convert the symbol to a relative address occupying bits 0 to 14 of the word following the instruction;
- a displacement value can be attached to ⋆ or <symbol> to indicate a word which is not labelled by an identifier; it should be noted that, as both address values and the location counter advance by increments of 2, the displacement value must be twice the even number of characters to be advanced or retraced, i.e. a value of 2 for each single-word instruction or data-word and 4 for each two-word instruction.

Examples:

```
␣LD␣A2 , BOX␣LOAD CONTENTS OF BOX INTO A2.
␣ST␣A2 ,⋆+4␣STORE A2 CONTENTS IN WORD AFTER
⋆THIS (TWO-WORD) INSTRUCTION..
␣ANS␣A9, BOX +2␣PERFORM LOGICAL AND ON
⋆CONTENTS OF A9 AND WORD AFTER BOX, STORE
⋆RESULT IN BOX +2.
␣C2⋆␣ITEM␣TWO'S COMPLEMENT CONTENTS OF
⋆WORD WHOSE ADDRESS IS HELD IN ITEM.
```

Register expressions are written in the same way as in register-to-register instructions. The register corresponding to the R1 field is used in the same way as in register to register instructions, if present. The register corresponding to the R2 field is used to modify the address expression in the operand, particularly when it is desired to address consecutive words or characters of a data-block or buffer on consecutive runs through an instruction sequence.

The value held in the register should be:

- for word addressing instructions, initially an even number, incremented by 2 for each advance of one word (cf. displacement values);
- for character addressing instructions, an odd or even number, incremented by 1 for each consecutive character; bit 15 of the word following the instruction (which contains an address value) is set to one if the (index)register contains an odd number and to 0 if it contains an even number; if bit 15 is set to 1, the right hand character of a data word is referred to; if to zero, the left hand character.

CONSTANT INSTRUCTIONS

A variety of constants may be used in either long or short format instructions. However, certain of the operations available cannot be performed by short format instructions.

The operand of a long constant instruction contains a register expression (referring to any of the 16 registers) and one of several types of constant:

- decimal constant; this may be a digit or an integer in the range 32767 to -32768.
- hexadecimal constant; this is considered as a logical value in the range 0 to 65535, and is assembled on 16 bits (bit 0 is not a sign bit); it is written in the form
  X'<hexadecimal integer>' or
  /<hexadecimal integer>
  where <hexadecimal integer> may be a digit or integer in the hexadecimal system of numbering, which uses, in addition to the digits 0 to 9, the letters A to F to represent the numbers 10 to 15 (highest possible value: X'FFFF' or /FFFF).
- character constant; by means of this constant the user can specify one or two characters of the character set on page 35; the Assembler will generate the internal ASCII configuration required; the constant is written in the form:
  '<character>[<character>]'
  The quote mark (') may not be used as a character in this type of constant.

If the constant is one character long, it is right-justified; the left character becomes zero then: X'F' becomes 0F.

In all instructions using a long constant in the operand, an L must be attached to the mnemonic.

examples:

```
␣ADKL␣A12, 3000␣ADD 3000 TO CONTENTS OF A12.
␣ANKL␣A6, X'FF00␣PERFORM LOGICAL AND OF A6
⋆CONTENTS AND X'FF00'
␣CWK␣A2, '␣␣'␣COMPARE A2 WITH 2 BLANKS
```

The operand of a short constant instruction contains a register expression (referring to one of the first eight registers) and one of the above-mentioned constants, with the following restrictions:

- decimal constants: the range 0 to +255 (2^8–1) can be represented (i.e., no negative values);
- hexadecimal constants: the range 0 to 255 can be represented highest value: X'FF');
- character constants: a single character can be specified ('<character>').

Examples:

␣ADK␣A2, 124␣ADD 124 TO CONTENTS OF A2.
␣ORK␣A4, X'0F'␣PERFORM LOGICAL OR OF A4
⋆CONTENTS AND X'0F'.

Branch instructions
Memory reference instructions, register-to-register instructions and constant instructions all include certain forms of branch operation, enabling control to be given to an instruction outside normal sequence.

In branch instructions the register expression representing the R1 or R3 field is not used; instead, a condition indicator is attached to the mnemonic. The instruction to which control is to pass can be indicated in various ways:

- by means of a symbolic address expression (memory reference instructions or constant instructions);
- by an absolute address held in a register (indirect addressing in register-to-register instruction);
- by using a constant to indicate an absolute memory address (short constant instructions);
- by means of a displacement value added to, or subtracted from, the instruction counter value (RB and RF instructions only); this displacement value is computed by the Assembler from an address expression used in the operand, and may not exceed 127 words.

If the condition indicated is fulfilled, control passes to the operand address; otherwise, the next instruction in normal sequence is carried out. If no condition is specified the branch is automatically carried out.

Shift instructions
The operand of a shift instruction simply contains an expression referring to one of the registers 1 to 7, whose contents are to be shifted by one position in the direction and manner specified by the mnemonic. The P-register must never be specified in a shift instruction mnemonic.

Examples

␣SAL␣A1␣SHIFT A1 LEFT, ARITHMETICALLY.
␣SRC␣A2␣PERFORM RIGHT CIRCULAR SHIFT
⋆ON A2.

I/O and miscellaneous instructions
In this group of instructions a variety of expressions - register, function code, device address - can be used; or in certain cases, no operand is present.

Comment Field
Comments are, of course, only for the programmer's benefit, and will be included in the Assembly listing but not in the generated object program.

A comment can be continued onto the next line, or an independent comment statement introduced, by putting an asterisk (⋆) in the first column of the programming paper or by starting a line with more than 10 blanks.

Examples:

PNCHON␣CIO␣A4, 1, 4␣THIS COMMAND SWITCHES
⋆ON THE TAPE PUNCH (ADDRESS 04); A4 CAN BE USED
⋆TO CONVEY CERTAIN INFORMATION TO THE CPU.

ASSEMBLY DIRECTIVES

The directives used to control the Assembly process and to generate data on the P855 and P860 may be divided into two groups, according to the Assembler which is being used:

Group 1:
DATA, EQU, IDENT, END, RES, AORG, RORG, ENTRY, EXTRN, these may be used with both Assemblers;

Group 2:
EJECT, LIST, NLIST, STAB, COMN, FORM, XFORM, GEN, IFF, IFT, XIF; these can be used only with the extended assembler.
Directives may be used for various purposes: to generate data, to control the assembly process, to give information to the Linkage Editor if required.

Group 1 directives

DATA (Data generation)
The DATA directive enables the user to generate data conveniently within his program. It has the following syntax:

[<identifier>]␣DATA␣<data expression>

The meaning of <data expression> is:
the data expression may be a decimal or hexadecimal constant or an address expression as described previously.
Character strings are also possible; a character string consists of a string of from 1 to 32 characters selected from the Assembler character set, placed between quote marks, there may be more than two data expressions in the operand. A quote mark within a character string is represented by 2 quote marks: 'IT"S'.

In the case of the character string a series of words will be generated in memory, one word for every two consecutive characters; if the number of characters is odd, the rightmost character of the last word will be generated as a space.

In other cases, one word will be generated for each data expression. However, one single DATA directive may not generate more than 16 memory words.
The DATA directive can be used to generate the bit-configuration of any I/O or miscellaneous instruction wherever it is required in a program.

Examples:

```
␣LDKL␣A1,-64 *LOAD -64 INTO A1 REGISTER
SM␣DATA␣72 WORD SM CONTAINS DECIMAL
*VALUE OF 72
␣LD␣A4,SM LOAD CONTENTS OF WORD SM INTO
*A4 REGISTER

PRINT␣DATA␣TR04␣IS␣OFF␣LINE'␣CHARACTER
*STRING GENERATED.
␣LD*␣A9, LABEL␣LOAD CONTENTS OF BOX
*(INDIRECT ADDRESS) INTO A9 LABEL␣DATA␣BOX.
```

EQU (Equate symbol to a value)
The EQU directive is used to equate the symbol in the identifier field to a value in the operand field which may be either absolute or relocatable. It is written as follows:

<identifier>␣EQU<predefined expression>

The EQU directive can be used to generate mnemonics to represent the bit-configuration of I/O and miscellaneous instructions. A mnemonic thus defined can then be used whenever the bit configuration which it represents is required in the program

Examples:

```
CT␣EQU␣/41C4␣*GENERATE CIO START. CT CAN
*NOW BE USED ANYWHERE IN THE PROGRAM TO
* REPRESENT THE VALUE /41C4.
MASK␣EQU␣X'FFFF'␣VALUE X'FFFF' GENERATED
*AT ADDRESS MASK.
```

IDENT (Program Identification)
The IDENT directive specifies the name to be given to the object program for identification purposes. It is written as follows:

␣IDENT␣<program name>

The <program name> has the form of a symbol or identifier as defined in the Assembly Language in which the program itself is written. This name may be defined and used in the remainder of the program or in other programs, or specified for the purposes of selective loading or updating (see chapters on Linkage Editor, Executive and Update Package). The <program name> must be followed by at least 10 blanks. The IDENT statement must always be present and must be the first statement in a source program.

END (End of Assembly)
The END directive is the last statement of a source program. It is written as follows:

[<label>]␣END␣[<predefined expression>][,<symbol>]

The first operand expression, if present, gives the address of the first instruction to be performed in the program.

The second operand expression is not valid as an input to the Basic Assembler. With the extended Assembler, it gives a name to the internal symbol table of the generated object program (for debugging purposes), and causes the table to be included in the object program, if a STAB directive has previously been successfully assembled. (see STAB)
The END directive causes the Assembler to terminate the assembly, i.e. to perform the following functions:

- verify that all forward references are defined;
- verify that all entry points are defined
- terminate the assembly listing and object program output;
- give the generated object module the start address indicated by the first operand expression, if any. At loading time this start address will be included in the chaining table built up by the program loader; this parameter is therefore essential if several programs are to be loaded at the same time;
- if the second operand expression is present, and the STAB directive has successfully been assembled include the internal symbol table in the generated object program and give it the name specified by the parameter. The Assembler defines this symbol as an entry point and assigns to it the value taken by the location counter before processing the END directive; care must be taken to avoid a real entry point having the same name as the symbol table;
- read the next record of the source stream; if it is not an End of File mark, initialize the assembly process and assemble the next source module; otherwise, halt.

The identifier on the left of the END directive is given a relocatable value equal to the length of the relocatable program section of the generated object program (0 if the program is entirely absolute). The length includes the length of the optional internal symbol table, if this was specified in the END directive.

RES (Reserve Storage Area)
The RES directive is used to reserve a number of memory **words** for use by the program. It is written as follows:

[<identifier>]␣RES␣<predefined absolute value>

The first value in the operand field is a number of words to be added to the location counter; this number may be positive, negative or zero, but must not cause the location counter to take a negative value.

If the first operand parameter is positive the RES directive reserves the number of words specified, by adding twice this number to the value of the location counter (as explained earlier, the location counter always proceeds by increments of 2). The identifier, if present, is given the address of the first word of this area.

If the first operand parameter is negative the location counter is reset to a lower value, by adding twice this (negative) number to the location counter. The identifier, if present, is equated to the previous value of the location counter.

If the first operand parameter is zero, the location counter is not updated; if there is an identifier, the statement is equivalent to:

<identifier>␣EQU␣*

AORG (Assign Absolute Program Origin)
The AORG directive is used to assign an absolute value (**always even)** to the Location Counter. This is useful mainly for presetting data in interrupt locations or in other standard locations. It is written as follows:

␣AORG␣<predefined absolute value>

The <predefined absolute value> will normally be an absolute address represented by a constant. From the time AORG is given and until a RORG directive is given (see next section), the location counter will progress in the same way as if it were relative; i.e. by increments of 2 or 4, depending on the length of the instruction.

All identifiers in an absolute program section are given absolute values, unless they are equated to a predefined relative value by an EQU directive.

A short instruction in an absolute program section cannot refer to an address in a relocatable program section. A Memory Reference or long constant instruction, or a DATA directive, must be used for this purpose.

RORG (Assign Relative rogram Origin)
The RORG directive initializes the location counter with a relocatable value (always even)

␣RORG␣[<predefined relocatable value>]

It is used mainly to switch from an absolute program section to a relocatable program section.

If RORG has no operand, the location counter is given the highest relative value it has previously taken. This value is equal to the length of the relocatable program section at the time this directive is assembled.

ENTRY (Define entry point names)
The ENTRY directive specifies that one or several of the identifiers in the source program will be defined as entry points in the generated object module. It is written as follows:

␣ENTRY␣<entry point name>[,<entry point name>, ... <entry point name>]

If an identifier is listed after the ENTRY directive, this means that it is referred to in the operand of an instruction in a completely different program. The program which makes the reference cannot be executed in full unless it is linked with the program to which it refers, to form one complete program.
This Linkage process is performed by a Linkage Editor.

All ENTRY directives must be grouped together immediately after the IDENT directive.

Example:

␣ENTRY␣PROG2, CONVRT, CHECK.

EXTRN (Define external reference names)
The EXTRN directive specifies that one or several of the identifiers of the source module will be defined as external references in the generated object module. It is written as follows:

␣EXTRN␣<external name>[,<external name>, ..., <external name>]

If an identifier is listed after the EXTRN directive, this identifier is used in the operand of an instruction in the program to refer to an instruction or a symbol defined in another program with the same identifier in its label field. The program cannot then be performed unless it is linked to the program to which it refers, as no address values can be given for the external identifiers. This linkage process is performed by a Linkage Editor.

A symbol which is defined as an external name must not be used in the identifier field of an instruction in the same program. It may only be used in DATA statements or in the expression of a constant or address on 16 bits. If it is used in an address expression, the remainder of this expression, if any, must be absolute: i.e. never <symbol> ± <symbol>, but <symbol> ± <integer>.

The EXTRN directives must be grouped together after the last ENTRY directive, if any, or after the IDENT directive if there is no ENTRY directive.

Example:

␣EXTRN␣ASCON, PRINT4, MASK2

Group 2 Directives (Extended Assembler only)

STAB (Define Internal Symbol Table)
The STAB directive allows the user to declare that one or several of the internal symbols will be included by the Assembler at the end of the relocatable section of the object program, for debugging purposes. It is written as follows:

␣STAB␣[<internal symbol>][,<internal symbol>, ...,<internal symbol>]

They will be ineffective if the END directive does not contain a second operand parameter indicating the name of the internal symbol table.

All the symbols listed after STAB must have been defined previously in the current program. No external or entry point names may be specified.

If STAB has no opernand, all the internal symbols of the source program will be included in the object program.

Example:

␣STAB␣CHECK, REG4, SWITCH

NLIST (Suspend listing output)
The NLIST directive causes the assembly listing to be suspended, from the point where the NLIST directive is given until either the END statement is reached or a LIST directive is given. It is written as follows

␣NLIST␣

Lines which contain errors will continue to be printed out during this phase

LIST (Resume listing output)

The LIST directive causes the Assembler to resume the listing output after it has been suspended by an NLIST directive. It is written as follows:

␣LIST␣

EJECT (Continue listing on new page)
The EJECT directive causes the remainder of the current page of lineprinter paper to be left blank, and the listing output to be immediately continued at the top of the next page. It is written as follows:

␣EJECT␣

Conditional Assembly directives (IFT, IFF, XIF)
The directive IFT or IFF is used in combination with the directive XIF to indicate that a bloc of instructions is to be assembled only if a certain condition is fulfilled. The IFT and IFF directives are written as follows:

␣[IFT|IFF]␣<predefined absolute expression>=<predefined absolute expression>

and XIF as follows:

␣XIF␣

IFT (If true) enables the assembly of all source lines following IFT, up to the next XIF directive, if the condition expressed in the operand field of IFT is satisfied.
If the condition is not fulfilled, the lines up to XIF are not assembled.

IFF (If false) has the reverse function: the source lines following IFF, up to the next XIF directive, are assembled if the condition expressed in the operand field is **not** satisfied.

Assembly of the END and IDENT directives can never be disabled.

Example:

```
␣IFF␣* = X0800␣WHEN LOCATION COUNTER
* REACHES VALUE X0800, CURTAIL PROGRAM
  -
  -
long
  -
  -
instruction sequence
  -
  -
  -
␣XIF␣
* PROGRAM ALREADY 1k IN LENGTH, REST
* ABANDONED.
␣END␣FIRST.
```

COMN (Declare Common block)
The COMN directive facilitates communication between programs written in Assembly language and FORTRAN. It is written as follows:

[<label>]␣COMN␣<common field definition list>

where:
<common field definition list>::=<common field definition>, [,common field definition list]

where:
<common field definition>::=<common field name>[(<common field length>)]
or <identifier>(<integer>)

If the parameter <common field length> is omitted, it is assumed by the Assembler to be 1. The field length must be given in words. Any COMN directives present must be placed after the EXTRN directives, if present; otherwise after the ENTRY or IDENT directive, whichever is used last. So: ENTRY-EXTRN-COMN.

By using a COMN directive the programmer can define one or more common blocks, which can be used in an Autocode program to transmit data to a FORTRAN program. Each common block may be divided into a number of sub-fields of varying length, each having a symbolic name which can be referred to directly, but only in the module in which they are declared.

Example:

A␣COMN␣FVAL1 (3), FVAL2 (3), INTGV (10)

This defines a labelled common named A having the length 3+3+10 = 16 words.

A is defined as an external reference and common block name. Either the common block name itself or the subfield names may be referred to in the same program. The subfield names are then considered as equivalent to:

<common block name> + <absolute displacement>

For example,
LD␣A1,FVAL2 is equivalent to LD␣A1,A+3

and
ST␣A2, INTVG + 9 is equivalent to ST␣A2, A+15

At link-edit or link-load time the Linkage Editor reserves a space for this common block at the end of the first relocatable program section that refers to it.

In addition to the 'labelled commons, a single 'blank common block can be allocated. This is defined by using COMN directives without an identifier in the identifier field; it can be referred to only by using the subfield names defined in the operand field.

Example:

␣COMN␣VAL1 (3), VAL2 (4)
␣COMN␣VAL3 (9), VAL4 (10)

These directives define a blank common block of 3+4+9+10 =26 words. VAL2, for instance, may be used in symbolic expressions and is equivalent to:

<blank common name> +3

The blank common name (assembled as ␣) is not used by the Assembler but has a meaning for the Linkage Editor, which allocates a space to the blank common block at the end of the link-load or link-edit run. This block will thus start at the end of the entire relocatable program section.

FORM (Format definition)
The FORM directive is used to define the format of a word, or a group of up to eight words, labelled by an identifier which can be used as an instruction mnemonic later in the program. The directive is written as follows:

[<identifier>]␣FORM␣<field definition>[,<field definition> ...., <field definition>][/<field number list>]

The field definition is written

<field length definition>[[=|:]<field value definitions>]

The field length definition states the number of bits to be allocated to a field of the word being generated, and can range from 1 to 16: alternatively it may consist of an expression which has been equated to a value in the range 1 to 16 by an EQU directive. If several fields are defined, the sum of the field lengths must be a multiple of 16. Not more than 8 consecutive words can be defined by a single FORM directive. A field may not overflow into the next word.

The field value definition can be used to place a value in the field to which it refers, if preceded by an equal-sign (=). If it is preceded by a colon (:) it indicates the address of a program word in relation to the first word of the expansion defined by FORM. The value definition itself may be an integer, a predefined expression of an external identifier. If there is no value definition attached to a particular field, this field will be filled with zeroes.

Examples:

VAL1 RES 1
INST FORM 16=X'8141', 16=VAL1

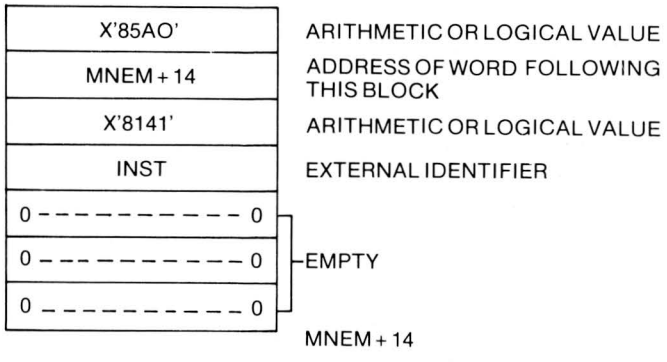
(The FORM directive generates two words containing, respectively, an absolute and a relocatable value).

WORD FORM 5,2=3, 1=1, 8

(The generated 16-bit word consists of four fields, two of which are left blank).

MNEM FORM 16=X'85A0', 16:14,16=X'8141',16=INST,16,16,16

(The following seven words are generated:



Thus the parameter 16:14 indicates a word address seven words from the beginning of the expansion defined by FORM).

An identifier which has been defined by a FORM directive can be used as an instruction mnemonic, with its operand field containing values to be placed in any non-predefined fields. Thus, in the second example given above:

```
WORD FORM 5, 2=3, 1=1, 8
  -
  -
  -
WORD 0, 24      .....
```

0 is placed in the first field, 24 in the fourth; the other two fields are already defined.

If it is desired to put the values of the operand field of the pseudo-mnemonic in an order different from that of the non-predefined fields they are to occupy, or if the user wishes to alter the values held by any of the predefined fields, a field number list must be used in the FORM directive.

Each field to be generated has a number, starting with 0 for the first field, 1 for the second field, (N-1) for the Nth. field (however, field numbers may not exceed 15). The field number list is introduced by a slash (/) and placed after the last field definition of the FORM directive; the field numbers are separated by commas.
All the fields specified in the field definition list must also be specified in the field number list. A field number is represented as a decimal integer. If a field number list has been used after a FORM directive, the operand expressions following the

pseudo-mnemonic will occupy the fields specified in the field number list in the given order. In this way, the contents of predefined fields may be altered while blank fields may be left blank.

Example:

```
WORD  FORM 5,2=3, 1=1, 8/2, 1, 3
WORD  PAE1, PAE2, PAE3
```

The field number list has the following effect:

- field 0 is filled with zeroes;
- field 1 contains PAE2;
- field 2 contains PAE1;
- field 3 contains PAE3;

The operand expressions following a pseudo-mnemonic are positional parameters; if one is omitted (other than the rightmost one), its position must be indicated by a comma.

If a FORM defined pseudo-mnemonic is identical with a standard instruction mnemonic, the pseudo-mnemonic has priority.

XFORM (extension of FORM directive)
The XFORM directive can be used to define a number of pseudo-mnemonics which have the same format as an existing pseudo-mnemonic defined by FORM, but may differ as regards the contents of certain fields. XFORM is written as follows:

<identifier>␣XFORM␣<FORM-defined pseudo-mnemonic>,<field list>

The field list is a series of field definitions, giving the format of the new pseudo-mnemonic and the contents of its fields. No field number list can be used.

The field length definitions must be the same as those of the FORM directive referred to, and appear in the same order. All non-predefined fields must be of the same nature, size and disposition as in the FORM directive.

Example:

```
LD    FORM 1=1, OPC=0, R1,1=1,I,X, LS=0, M/4,2,7,5
AD    XFORM LD, 1=1,OPC=2, R1,1=1, I,X, LS=0,M
AD.S  XFORM LD,1=1, OPC=2,R1,1=1, I,X, LS=1,M
```

GEN (Generation directive)
The GEN directive can be used to extend the symbol table of the Assembler, so that the Assembler will recognize and assemble a number of non-standard symbols in any program in which they are used. Thus a particularly useful pseudo-mnemonic or system macro (e.g. multiple shift operation) can be defined once for all times, instead of having to be generated by a FORM directive in every program where it is used.

The GEN directive is written as follows:

␣GEN␣[A]

A, if present, indicates that the program is output in absolute form.

Only one GEN directive may be used in a source program, and it must appear before the END directive. The source program may not contain any of the directives ENTRY, COMN, RES, AORG, RORG, STAB or DATA, any relocatable symbols, machine instructions or pseudo-mnemonics defined by FORM **and used in the operation code field**. GEN must be used very carefully, for the system performs no check on it.
The Assembler symbol table consists essentially of entries which define standard symbols (such as P, A1.....A15), mnemonics of all directives, standard external references and standard FORM-defined mnemonics. This table can be extended by writing a source program consisting mainly of EQU, FORM and XFORM directives (which generate the desired values) with a GEN directive before END; this program is assembled and can then be either linked to the Assembler and output (by the Linkage Editor) or simply loaded into memory immediately behind the Assembler.

By using GEN, FORM and XFORM, it is possible to define a specialized language for a specific application, such as process control, message switching etc. For example, as a single FORM directive may generate up to 8 words, a complex sequence of introductions may be reduced to a short sequence of predefined pseudo-mnemonics; this is especially useful for calling simulation routines. However, if there is an error in an EQU, FORM or XFORM directive, the GEN directive will be ineffective.

PROGRAMMING EXAMPLE

See following pages.

Problem EXAMPLE Programmer A. V.D. LINDEN Date 11-11-1971 Page 1 of 3

LABEL	OPERATION	OPERAND		IDENTIFICATION
*THIS	IS A PR	ØGRAM TØ TRANSFER 20 CHARACTERS FRØM AN I/Ø DEVICE		
*(WITH	ADDRES	S=1) INTØ A BUFFER, ADD THEM TØGETHER IN PAIRS AND STØRE		
*THE	RESULT	IN THE SAME BUFFER.		
	IDENT	PRØGEX		
S	EQU	1		
H	EQU	0		
ASR	EQU	1		
BUF	RES	10	RESERVE BUFFER	
CØUN1	DATA	-20	INITIALIZE CØUNTERS	
CØUN2	DATA	-10		
START	LDK	A2,0	REGISTER A2 IS USED AS INDEX REGISTER FØR BUFFER	
*DURING	INPUT.			
	LDK	A5,1	LØAD 1 FØR INPUT BEFØRE CIØ START.	
TSTST	TST	A1,1	TEST DEVICE STATUS.	
	ABL(3)	DEVUN	BRANCH IF DEVICE UNKNØWN.	
	SRC 1	A1		
	ABL(2)	TSTST	BRANCH BACK IF DEVICE IS BUSY.	
STASR	CIØ	A5,S,ASR	START ASR.	
INPUT	INR	A3,0,ASR	INPUT ØNE CHARACTER.	
	ABL(1)	SENST	IF NØT ACCEPTED, GET STATUS WØRD.	
	SC	A3,BUF,A2	STØRE CHARACTER IN BUFFER.	
	IM	CØUN1	INCREMENT CØUNTER1.	
	ABL(0)	STDEV	BRANCH IF ZERØ.	
	ADK	A2,1	INCREMENT INDEX REGISTER.	

Problem EXAMPLE Programmer A. V.D. LINDEN Date 11-11-1971 Page 2 of 3

LABEL	OPERATION	OPERAND	IDENTIFICATION
	ABL(7)	INPUT INPUT NEXT CHARACTER.	
STDEV	CIØ	A5,4,ASR HALT I/Ø.	
	ABL(7)	INPUT BACK TØ INPUT.	
SENST	SST	A6,1 CØMMAND ACCEPTED? IF NØT BRANCH TO INPUT	
*TEST	CØNTENT	6 ØF STATUS WØRD IN A6.	
	SRC1	A6	
	ABL(0)	ADDIT INPUT CØMPLETE, GØ TØ NEXT ØPERATIØN.	
	ABL(2)	NØTØP BIT15=1, DEVICE NØT ØPERABLE.	
	SRC1	A6	
	ABL(2)	THERR BIT14=1, THRØUGHPUT ERRØR.	
	SRC1	A6	
	ABL(2)	DATAF BIT13=1, DATA FAULT.	
	SRC1	A6	
	ABL(2)	INCLE BIT12=1, INCØRRECT LENGTH.	
THERR	LDK	A4,1	
	ABL(7)	HALT	
INCLE	LDK	A4,2	
	ABL(7)	HALT	
NØTØP	LDK	A4,3	
	ABL(7)	HALT	
DATAF	LDK	A4,4	
	ABL(7)	HALT	
DEVUN	LDK	A4,5	
	ABL(7)	HALT	
ADDIT	LDK	A1,0 SET INDEX REGISTER FØR BUFFER.	

Problem EXAMPLE Programmer A. V.D. LINDEN Date 11-11-1971 Page 3 of 3

LABEL	OPERATION	OPERAND	IDENTIFICATION
	LDKL	A3,/00FF LØGICAL CØNSTANT INTØ A3.	
SUM	LC	A2,BUF,A1 LOAD LEFT CHARACTER INTØ A2.	
	ANS	A3,BUF,A1 CLEAR BITS 0 TØ 7 ØF BUFFER CURRENT WØRD.	
	ADS	A2,BUF,A1 ADD RIGHT CHARACTER TØ LEFT ØNE AND STØRE	
*INTØ	BUFFER.		
	ADK	A1,2 INCREMENT INDEX REGISTER A1	
	IM	CØUN2 INCREMENT CØUNTER2.	
	ABL(2)	SUM LØØP ØN SUM UNTIL CØUNT2=0.	
HALT	HLT		
	END	START	
*WHEN	THE TABLE IS CØMPLETE (I.E. CØUN2=0) ITS CØNTENTS CAN BE CØNVERTED		
*TØ DECIMAL	NØTATIØN AND EXTERNAL ASCII FØRMAT, USING A UTILITY PACKAGE,		
*AND THEN ØUT	PUT ØN THE ØPERATØR'S TYPEWRITER.		

ASSEMBLER

The Basic Assembler produces a program in object code (which may include a number of unsatisfied external references), an assembly listing and error messages. It allows the user, in certain cases, to correct errors detected during the actual assembly. The source program must be input on one device and the object program output on a different device. The assembly listing, error and control messages will be typed on another device, usually the operators typewriter.

There are two versions of the Assembler:

- a stand alone version which requires at least 4k of memory. This version does not include the following features:
  - no simultaneous I/O (no double buffering)
  - group 2 directives (see Assembly Language)
  - some instructions are not optional
  - condition mnemonics
  - register mnemonics
  - internal symbol table output.
- a version running under monitor control. This version requires at least 8k of memory, accepts the full assembly language and does simultaneous I/O operations. It is called the extended version.

Initialization and Input

The stand-alone version of the assembler is initialized as follows:
The Assembler is in absolute code and can be loaded by the bootstrap loader. It is started by pressing the START button on the control panel.
After the Assembler has been loaded, but before it is started, the source program should be put on the desired peripheral and all devices to be used during the Assembly should be switched on.

Input Files

The type of source input file is specified in the option statement. All source statements are followed by CR.LF. or LF.CR. The end of an input file is indicated by an End-Of-File mark.
The input file may contain deletion codes:

- a vertical arrow (↑) causes all characters of the input line to be ignored by the input routine, up to the next CR.LF., inclusive.
- a string of n left arrows (←) causes an equal number of characters to the left of the arrow to be ignored by the input routine.
- the 'delete character (8 punches) is always ignored by the input routine.

Initialization

The assembler, after it has been started, types out the message:

A:

on the operators typewriter. This is a request for the assembly options to be specified.
If the operator wants the standard option, i.e.

- source input: high speed punched tape reader or card reader
- no error recovery from the typewriter
- listing on the typewriter or line printer
- object code output: high speed tape punch,

he types in:

.CR.LF.

If he wants any other options, he must type in an option statement, followed by CR.LF. The option statement has the following format:

[<hexadecimal constant>][R][N][Q]

where:

<hexadecimal constant> consists of four digits with the following meanings:

- first digit: source input file code.
- second digit: is not taken into account.
- third digit: listing output file code. May never be 0.
- fourth digit: object code output file code; 0 if no object code output is desired.

R must be specified if error recovery via the typewriter is wanted.

N must be used to indicate that the output listing must be suspended.
However, even in this case, erroneous lines will still be listed, together with the IDENT and END statements and the symbol table.

Q is used to specify that the object code must be produced in 4×4×4×4 tape format.

The default value for the option statement is 1023.

If the option statement is erroneous, e.g. tape punch specified as the input device, the assembler again types out:

A:

and the operator may type in a new option statement.

(For **systems with ASR 33 as the only peripheral device**, the option statement always is 6057Q. Moreover, source lines read from the ASR tape reader must end with CR.Xoff.LF.).
After the option statement has been typed in, followed by CR.LF., the assembler starts assembling the source program.
If, for any reason, the operator wants the exit of the assembler, he can, after the assembler's message A: type in EOF.

Processing

The assembler normally processes all modules of the input stream, from the first IDENT statement to the last END directive, or until an End-Of-File mark is encountered in the input stream. It reads the source programs line by line, checks each one for errors, prints out the listings with error messages, if applicable, and produces an object module for each source module offered to it. In certain cases it will stop and type out a message to allow the operator to correct an error discovered during the assembly. Finally, the assembler outputs an End-Of-File mark on the object code output medium to indicate that the assembly has been completed.

If N is specified in the option statement, no listing will be produced. However, any erroneous source lines with error messages, the IDENT and END statements and the symbol table will always be output on the operators typewriter.
If 0 is specified for the object code output file code, no object module will be produced.

Having output the EOF mark and finished assembly, the assembler types out A: on the operators typewriter.
If the operator wants to stop assembling, he must type in EOF. If he wants another assembly to be performed, he must type in a new option statement.

An EOS mark in the input stream causes the assembler to stop. This can be very useful at points where the operator has to change tapes. Having done this, he must type in CR.LF. for assembly to continue.

Output
There are four kinds of assembler output:

- programs in object code; all modules are separated by an EOS mark, and the last one is followed by an EOF mark.
- assembly listings.
- assembly error messages.
- control messages and operator error messages.

Assembly listings
The assembly listing lines contain the following information, from left ro right:

- an error flag, if an assembly error was detected in the source statement.
- the source line number, in decimal. Recovered source lines are not numbered.
- the value of the location counter, in hexadecimal. For all directives, except RES, this field remains blank.
- the hexadecimal representation of the generated code.
- a flag indicating if the generated code is relocatable (R), or contains an external reference to be solved at link-load time (X), or a forward reference (F).
- the source line image.

The assembler outputs as many listing lines as code words produced, but the lines following the first one are not numbered. They only contain the location counter value, the representation of the generated code and, if required, the flag indicating relocatable, external or forward.

The following feature applies on the **extended version** of the assembler: If a **symbol table** name for the entry points has been specified in the operand of an END directive and a STAB directive has been specified, a part (the symbols specified in the operand of the STAB directive) or the whole symbol table is output as code clusters, word by word (relocatable or absolute). The end of the table is indicated by a zero word. The entry point name has the value of the location counter before the processing of the END directive.
This table may then be used by the Debugging Package.

After the assembler has processed the END directive, this symbol table is also printed out at the end of the listing. The table is headed SYMBOL TABLE.
Each symbol is printed, together with its value and type:
The value is given as a four-digit hexadecimal number.
The type is given as a letter:
A if the symbol is absolute,
R if the symbol is relocatable,
X if it is an external reference.
If a symbol is undefined, its value is given as **.

The Symbol Table is followed by the message

UND.ENT.␣␣␣␣␣

if an entry point name has remained undefined; and/or

UND.LAB.␣␣␣␣␣
if a label has remained undefined.
Then the message

ASS.ERR.␣␣␣␣␣

is printed, giving the number of assembly errors in the current module.
In the specifications for the rightmost 8-bit expression in forward references two kinds of error may occur:

- overdisplacement
- not an absolute value.

In this case the following message is printed:

FOR.OVR. XXXX

where XXXX gives the address of the word where the error occurred.

Assembly Errors
When an assembly error is detected, no listing line is printed.
First an * is output, the second character is the error code. This is followed by the sequence number, the hexadecimal representation of the location counter value, the code representation of the HLT instruction and the source image of the erroneous statement.
The second line also begins with an *, but the sequence number is replaced by *****; then the location counter value is given, followed by the code representation of RB *-2 instruction, where * indicates the location of the character where the error was detected.
Such error lines are always printed, even when the option statement specifies that listing must be suspended.
Fatal errors are printed as follows:
The first character of the line is an *. It is followed by 5 more asterisks. Then the error code is given (one letter), plus the sequence number and the source image of the corresponding statement line. Only one listing line is printed.

Control Messages
These messages are always typed out. They are used to supply information on the handling of the assembler, not about the current assembly run.

Error handling

When an error is detected, this results in the following actions:

- if, in the assembly option statement, no error recovery has been specified, no attempt is made to retrieve this error: the error counter is incremented and the next statement is assembled.
- if error recovery has been specified in the option statement, the assembler types out R:
  and waits for input. This may be
  .CR.LF. if the operator does not want ro recover this error.

This makes the error definite and the following two instructions are produced in the object code: HLT and RB*-2 (two words).
If the operator wishes to correct the error he types in a new source line, followed by CR.LF.

Some errors are handled in a different manner:

- illegal use of forward references (overdisplacement, illegal relocation) is not mentioned until at the end of the assembly (see Assembly Listings).
- END missing: a standard END statement is assumed, i.e. without label, start address and symbol table name. Then the assembly of the current statement is resumed (either an IDENT or an EOF mark).
- IDENT missing: any following statements are read and bypassed until a following IDENT statement or an EOF mark is encountered.
- core overflow: the current assembly is stopped, a standard END statement is assembled, the error flag is ignored and the relocatable length is set to 0. Any following statements are bypassed up to the next IDENT or EOF mark. Thus, the current module can be bypassed at link-edit/link-load time without having to do any punched tape manipulation.

Error and Control Messages

A complete error message directory is given in an Appendix at the end of this chapter.
Below are listed the operator messages, typed out for the operator by the system:

A: the operator can type in: an option statement, followed by CR.LF. or: CR.LF.
R: indicates that there is an error in a source statement. The operator can type in a new source statement, followed by CR.LF. If he does not want ro recover the error, he can type in .CR.LF.
EOS: an End-Of-Segment mark has been read. To restart the assembler, the operator must type in CR.LF.
EOF: an End-Of-File mark has been read. To restart the assembler, the operator must type in CR.LF.

APPENDIX 1 ERROR MESSAGE DIRECTORY

Note: a parameter is that part of the operand located between either a blank and one or two comma's, or a comma and a blank.

Code	*Meaning*	*Description*
⋆I	Illegal identifier	The first character of a name must be a letter
⋆C	Illegal constant	– constant overflow – a hexadecimal constant written beginning by X' must be ended by a '.
⋆X	Illegal expression	– more than two symbols – more than three terms – an external reference and a forward reference specified in the same expression – more than one external reference specified in the same expression – an external reference – an external reference preceded by a minus sign – no term following a plus sign or a minus sign – a forward reference or an external reference specified in a requested predefined expression
⋆R	Illegal relocation	– predefined absolute expression instead of a predefined relocatable expression or conversely – too many relocatable symbols are added or subtracted from each other – the expression is equivalent to the subtraction of a relocatable part from an absolute part – if an external reference is specified the displacement must be absolute – an instruction code operation defined by an EQU directive must be absolute
⋆L	Illegal label	– double definition as symbol name, or entry point name or external reference name – no label is allowed – a label is requested
⋆P	Illegal parameter	– too many parameters specified in the operand of an instruction or FORM defined pseudo instruction or directive – not enough parameters – a parameter of STAB directive must not be an entry point name or a common name or a forward symbol – the operand of a DATA directive must not produce more than 16 code words
⋆O	Overdisplacement	– overdisplacement of a parameter's value
⋆E	Not an even address	– the specified start address is not even – the specified AORG/RORG operand is not even
⋆M	Unknown mnemonic	– unknown mnemonic – unknown condition mnemonic – a mnemonic defined by EQU must be absolute – a mnemonic must be predefined
⋆S	Illegal statement	– the ENTRY/EXTRN/COMN directive is no longer accepted – this directive requests no operand – this directive requests an operand – this character is invalid – the indirect addressing is invalid – the condition specification is invalid – no operation code following a label – no) corresponding to a (– the operand value of a RES directive makes the location counter value negative – less than ten blanks following progname in IDENT – an operation code defined by EQU must not have an operand

Code	*Meaning*	*Description*
*F	Illegal FORM/XFORM	– an XFORM declared symbol must be linked to a FORM defined pseudo whose name is the first parameter of the operand – more than 16 fields specified – negative field length – the length of this field is not coherent with a 16 bit word (this field is too long) – no displacement allowed when the predefinition concerns an external reference name – the :- predefinition is only allowed for a 16 bit field – invalid predefined value of a field (over-displacement)/ negative value for a field of less than 16 bits) – the lay out of the current word of a XFORM declaration is not the same as the corresponding word of the linked FORM symbol – the predefinitions of the fields of the current word of a XFORM declaration are not the same as the corresponding word of the linked FORM symbol – more than 8 words described by a FORM declaration – more than the number of words described by the linked FORM symbol than described by a XFORM declaration – this field number specified in the syntax definition line is invalid – twice the same field specified in the syntax definition line – all the described fields must be specified in a Syntax Definition line
******O	Core overflow	– fatal error: too many symbols or too many forward references used
******E	End missing	– fatal error: End missing
******I	Ident missing	– fatal error: Ident missing

APPENDIX 2 LIST OF PREDEFINED SYMBOLS

Names are given to the registers and predefined values are associated to each register name. The Assembler obtains the internal code of each register from its corresponding predefined value.

Names	*Meaning*	*Predefined value*	*Internal value*
P	instruction counter	0	0
A1	register 1	1	2
A2	register 2	2	4
A3	register 3	3	6
A4	register 4	4	8
A5	register 5	5	10
A6	register 6	6	12
A7	register 7	7	14
A8	register 8	8	1
A9	register 9	9	3
A10	register 10	10	5
A11	register 11	11	7
A12	register 12	12	9
A13	register 13	13	11
A14	register 14	14	13
A15	register 15	15	15

P, A1, ...A15 can only be used to call the registers.
If they are used for other purposes the error message ⋆L (double definition) is output.

Linkage Editor

The Linkage Editor is used to join together a number of programs which make external references to one another. The product of this linkage is another, larger program, which may be loaded into memory during the linkage process (only if all external references are satisfied: Link-Load), or produced on an output medium for storage, further linkage or loading by the program loader (Link-edit).

It is possible to use the Linkage Editor in two ways, one of which must be specified before starting the processor:

LINK EDIT MODE

In this mode the Linkage Editor joins a number of object programs output by either the Assembler, the Fortran Compiler, the Linkage Editor itself or object libraries, into one larger object program, which may still contain external references, and outputs it onto a suitable medium.
If all external references are satisfied the program may be loaded by the program loader and executed; if not, it may be stored in an object module library or input to a further linkage process.

LINK LOAD MODE

In this mode the Linkage Editor loads a number of object programs and object libraries into memory where linkage will take place.
The user must ensure that all external references can be satisfied. If there is insufficient space in memory for all the object programs to be loaded the Linkage Editor will stop processing and print an error message.

CONFIGURATION REQUIREMENTS

The minimum configuration required to use the Linkage Editor is:
4k memory
operator's typewriter
high speed punched tape reader
high speed tape punch (only for link edit operations).

Due to his configuration the user must take the following limitations into account.
The memory size determines the maximum size of a program which has to be generated in memory and its maximum number of entry points in a linkage operation. Moreover it determines the maximum number of unsatisfied external references as well as the maximum number of entry points during a link edit process.

In a 4k configuration the maximum number of unsatisfied external references during the processing of an input module is 25. The maximum number of common blocks during a run in such a configuration is 20.

In case of core overflow the user may try to group several object programs into a single object program and enter the latter for a further link edit operation. In this way he can reduce the number of entry points in a given link edit operation.

Input to the Linkage Editor

The input to the Linkage Editor is twofold and consists of:

- options and control messages
- object programs and object program libraries.

OPTIONS

The options of the Linkage Editor allow the user to specify:

- the functions he wishes to have performed (link edit or link load)
- the peripheral units he wishes to use as input and output media.

The functions and peripheral units may be specified in the following option control message:

⌈[E|L⌈:<hexadecimal number>⌉,<name>⌈,4|8⌉⌉](CR.LF.)

where:

E = link edit mode
L = link load mode
:<hexadecimal number> = In link edit mode the hexadecimal consists of three hexa digits indicating the file codes of respectively
the object code output device (first digit)
the listing output device (map) (second digit)
the object module input device (third digit).
If either output device is not required the file code is replaced by a zero.
In link load mode the specifications are the same as for link edit mode except that the object output file code is not used and must **not** be replaced by a zero.
,<name> = the name to be given to the generated object module. (Meaningful only in a link edit operation).
,4 = the object program must be punched in 4-4-4-4 format (link edit only).
,8 = the object program must be punched in 8-8 format. (link edit only).

If no options are specified, link load from standard input file and standard listing file will be assumed.

OBJECT PROGRAMS AND OBJECT PROGRAM LIBRARIES

Object programs which may be input to the Linkage Editor are generated by either the Assemblers, the Fortran Compiler or by the Linkage Editor itself. Their format is as described in the chapter Object Language.
They are presented to the Linkage Editor either as object programs separated by an End Of File mark or as a series of object programs separated by an End Of Segment mark. The last program must be terminated by an EOF.

Each time the Linkage Editor encounters an EOF mark it prints on the operator's typewriter L: and waits for a new control message from the operator.

COMMONS

Labelled commons

Labelled commons are defined as external references in each module in which it is referred to. Any reference to a labelled common is handled as an external reference.
Each time a module of a segment refers to a labelled common for which space has already been reserved no new space will be reserved for this common.
If several object programs refer to the same labelled common the labelled common length must be the same.
Space for a labelled common is allocated behind the first module referring to it.

Blank commons

Blank commons are defined as an external reference in each module which refers to it. Any reference to the blank common is handled as an external reference.
The allocation of the blank common is given when the T control message is introduced. At that time the blank common is allocated at the end of the relocatable program section of the generated object program.
The length of the area reserved for the blank common is equal to the highest blank common length in an object module.

Inhibit Common allocation

By using the X control message it is possible to inhibit the allocation of some specific commons. All references to these commons remain external references in the generated object program.

Example:
Suppose we have labelled commons A and B and one blank common. The allocation is inhibited as follows:
X␣A,␣,B or
X␣␣,A,B

MEMORY LAYOUT

The figures below show the memory layout for the link edit function and the link load function.

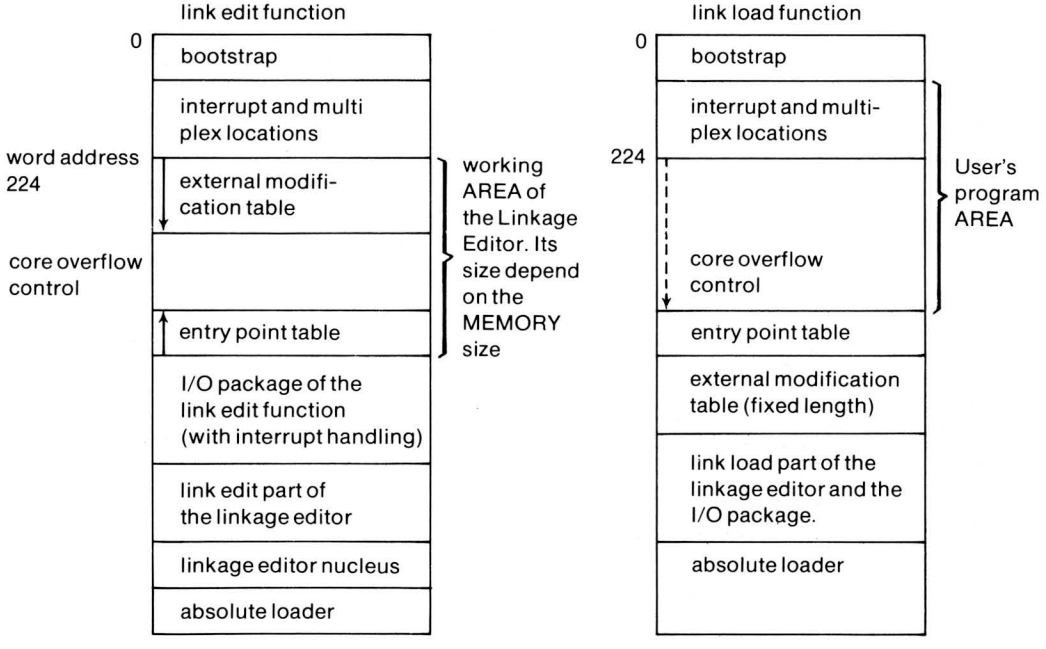
A number of physical limitations should be noted:

- the maximum number of common blocks in a link edit or link load operation is 20.
- the maximum number of unsatisfied external references, in a 4k configuration, is 50. (inside one module).

CONTROL MESSAGE FORMAT

The actual syntax of a control message is explained in the description of the control message. <address> and <symbol> appear in almost all control messages and are explained here:

<address>:: = string of up to 4 hexa digits. It is always a word address (even numbers).
<symbol>:: = string of up to 6 characters, the first one of which must be a letter.



Twelve operator control messages are provided to control the linkage process. Four of these can only be used in link edit mode. The use of the other eight is the same for both modes in most respects; the differences are explained in the control message.

DEFINE ENTRY POINTS (link edit only)

E␣<symbol>[,<symbol list>] (CR LF)

This control message causes the Linkage Editor to declare a series of symbols as entry points in the generated object program. These messages must be the first given and they must be grouped.

DEFINE EXTERNAL REFERENCE NAMES (link edit only)

X␣<symbol>[,<symbol list>] (CR LF)

This message specifies to the Linkage Editor that the external references listed must be left unsatisfied.

The X control message may be used if the generated module is to be input to a further linkage process but it is chiefly useful if one or more of the input modules include references to standard library modules; the latter will normally be present in memory at execution time and should not also be included in the load module.

All X control messages, if present, must be grouped together and be either the first of the messages sent to the Linkage Editor, or the first following a series of E messages.

DEFINE RELATIVE BASE ADDRESS FOR RELOCATABLE PROGRAM SECTIONS (link edit only)

R␣[<address>] (CR LF)

This control message enables the user to define a relative base address for relocatable program sections. Any absolute program section will remain absolute.
The base address progresses from the value specified in this control message. When address is not specified the value is assumed to be zero. In either case this value is updated by addition of the length of the programs input to the Linkage Editor, each time such a program has been fully processed.
Relative addresses in code words are relocated in relation to the base address. All relocatable program sections processed following this control message will result in a relocatable program section being output by the Linkage Editor. This does not apply to absolute program sections and this control message ceases to operate until an A control message is given.

DEFINE AN ABSOLUTE BASE ADDRESS FOR RELOCATABLE PROGRAM SECTIONS

A␣<address> (CR LF)

This control message enables the user to specify an absolute base address for relocatable program sections. This is useful in link load mode and also when the user wants to produce a program in absolute core image format (link edit mode).

Relocatable addresses in code words are updated in accordance with the base address. As with the R message this base address is updated by addition of the length of the programs input to the Linkage Editor, each time such a program has been fully processed. The resulting program, whether output or loaded, will be in absolute format until an R message is given.

In link load mode any P or S message must be preceded by an A control message.

PROCESS INPUT FILE UP TO END OF FILE MARK

P (CR LF)

This message causes the Linkage Editor to process the whole input file up to the End Of File mark.
In link load mode it must be preceded by an A message.

SELECT A SPECIFIED OBJECT PROGRAM OF THE INPUT FILE

S␣<symbol> (CR LF)

This message causes the Linkage Editor to process only the input program which has the name specified in this control message. In link load mode this message must be preceded by an A message.

SATISFY EXTERNAL REFERENCES FROM AN OBJECT MODULE LIBRARY

L (CR LF)

This message causes the Linkage Editor to process only those modules of the input file whose entry points match undefined external references.
However, external references listed in a previous X control message will remain unsatisfied.
If a library is not in such an order as to enable all external references to be solved in one run, it may be necessary to present it to the Linkage Editor several times. The U control message can be used to determine whether this is necessary or not.

LIST THE NAMES OF ALL UNSATISFIED EXTERNAL REFERENCES

U (CR LF)

This message causes the Linkage Editor to list on the operator's typewriter the names of all undefined external references except for any which have been listed in a X control message. One external name is typed out per line.

CHANGE ASSIGNMENT OF PERIPHERAL EQUIPMENT

C␣<hexadecimal number> (CR LF)

This message enables the user to change the assignment of a peripheral, the hexadecimal number being used in the same way as in the option message. E.g. after E,<name> C␣32A meaning switch from standard input file to an input from file code A.

TERMINATE LINKAGE PROCESS (link edit mode)

T[&] (CR LF)

This message causes the Linkage Editor to complete the object program output and (on option) print out a listing.

If & is not present in this message an End Of File mark is generated after the last object program has been output. The Linkage Editor halts and may be restarted by pressing the START button on the console.

If & is present in this message an End Of Segment mark is generated after the last generated object program has been output. A new link edit operation is then initiated by typing in a new option message as an answer to the L: typed out on the operator's typewriter by the Linkage Editor.

When the T message has been introduced the Linkage Editor prints out three messages:

L=<length> (CR LF)

which gives the length of the generated object program as an even number of characters. This length is the length of the relocatable program section, including the length of the blank common, if applicable.
If the generated object program is entirely absolute the length is indicated as zero.

S=<start address> (CR LF)

which gives the starting address (relative or absolute) of the generated object program.

E=<ending address> (CR LF)

This message gives the highest absolute address in the absolute program section of the generated object program. If the entire generated object program is relocatable the address will be zero.

The Linkage Editor then outputs the END cluster of the generated program, a flag indicating whether any errors have been detected and a protection mask.

TERMINATE LINKAGE PROCESS (link load mode)

T[D] (CR LF)

By introducing this control message the Linkage Editor will terminate the link load operation.
If the D parameter is present it causes the entry point table of the Linkage Editor to be shifted from its original place to the address following the highest address of the user's program.

When this control message has been taken into account the Linkage Editor prints on the listing device the following two messages:

E=<ending address> (CR LF)

where ending address indicates the last address of the user's program; if necessary the blank common length and the entry point table length are also taken into account.

S=[<start address>|?] (CR LF)

where start address is the start address of the user's program.

After having typed out this control message the Linkage Editor halts. When the start address is known the user may start his program by pushing the START button, otherwise the user must start the program from the console.

PROCESSING

The Linkage Editor is loaded into memory and given control by the Absolute Loader. It then types out on the operator's typewriter:

L:

and waits for the operator to input the options as described in option control message.

If an erroneous parameter was given the Linkage Editor types out:

L?

and the operator must type in the correct option control message.

The Linkage Editor will now start a link edit or link load operation. The linkage process is controlled by the messages typed in by the operator.

Link edit operation
In case of a link edit operation the Linkage Editor types out in ASCII the name of the object program chosen in the option control message and next

L:

The operator may now introduce a control message. The Editor starts reading the input stream each time a P, S or L control message is given and processes the records of the input stream according to their type.
All entry points of the program are placed in the entry point table as well as the blank common name and its length when the name has not yet been encountered on the input file.
When a blank common name is met the common length is compared with the length previously defined.

The relative addresses of relocatable code clusters are changed into absolute addresses when an A control message has been introduced, or in a partly relocated relative address after an R control message.
The relocation is calculated by adding the relocation value specified in the A or R control message and the sum of the length of all object programs processed from the A or R control message.
The length is zero for entirely absolute programs.
The code words are relocated by addition of the relocation value.

When externals are specified the code words are changed according to those externals. Otherwise all information is kept in an external modification table until the externals are defined.

The externals listed in the X control message remain unsatisfied in the program and are taken into account when the external reference cluster is modified.

When the external modification table is filled the Linkage Editor tries to decrease the number of external modifications by satisfying external references each time an external name is defined as an entry point.
When this is possible an entry is made in an internal modification cluster and, if necessary, this cluster is output.
When this operation has not been successful the message 'TABLE OVERFLOW' is output and the processing stops.

Absolute code clusters are processed in the same way as relocatable code clusters except for the address of code clusters which is never modified.

When all records of the object program have been read the Linkage Editor processes the END cluster and reads the next record of the input file.

If it is not an End Of File the Editor will process the next object program on the input file. If it is an EOF the Linkage Editor types out:

L:

and waits for an operator control message.

Link load operation
The processing of programs in a link load operation is almost similar to that in link edit mode.
Relocatable code clusters are processed by loading sequentially into memory all code words of the cluster, starting at an absolute address calculated by addition of the currently processed object program length to the relative address specified in the cluster of the absolute base address. The latter is computed by adding to the absolute address specified in the previous A control message, the sum of the lengths of all the object programs processed since the last A control message (or since the beginning of the link load process when no A control message has been introduced).

The absolute code clusters are processed in the same way as the relocatable code clusters, except for the address of the code cluster which is never changed.

OUTPUT

The output of the Linkage Editor depends upon which option (link edit or link load) has been chosen.

Link edit output
The output of link edit operation is an object program punched on paper tape. This object program may be either:

- a self contained program (i.e. an object program with no external information). This program may serve as input to the resident loader or the Linkage Editor.
- and object program still containing external references. Such a program may only be input to the Linkage Editor.

Link load output
In a link load operation no object program is output as the program has been loaded directly into memory.

INFORMATION MESSAGES

The Linkage Editor outputs information messages for the operator when a specific action is required by him or to inform him about the linkage operation.

Message	*Meaning*
L:	Input a control message.
L?	The previous control message contained an error. Type in the correct message.
<symbol>␣<address>	The processing of an object program has been initiated. <symbol> = name of this object program.
S = [R␣]<address>	This message gives the start address of the generated program. R, if present, means that the start address <address> is a relative address in the relocatable program section of the generated program. If R is absent (link load) the start address is absolute.
L=<hexadecimal value>	This message gives the length of the relocatable program section of the generated object program. <value> is an even number and zero if the generated program is entirely absolute.
E = <address>	This message gives the highest address (absolute) in the generated program. It is zero when this program is entirely relocatable.

ERROR MESSAGES

Errors can be devided in:

- fatal errors
- non fatal errors

Fatal errors
When a fatal error occurs e.g. core overflow, the Linkage Editor stops processing and may not be restarted.

Non fatal errors
Some errors do not cause processing to stop.
An error message is output. An error in the input of object programs causes the Linkage Editor to type out an error message and it halts. The operator may have the last record read again after having pushed the START button.

Map
The map is printed by the Linkage Editor on the listing device after a T control message.

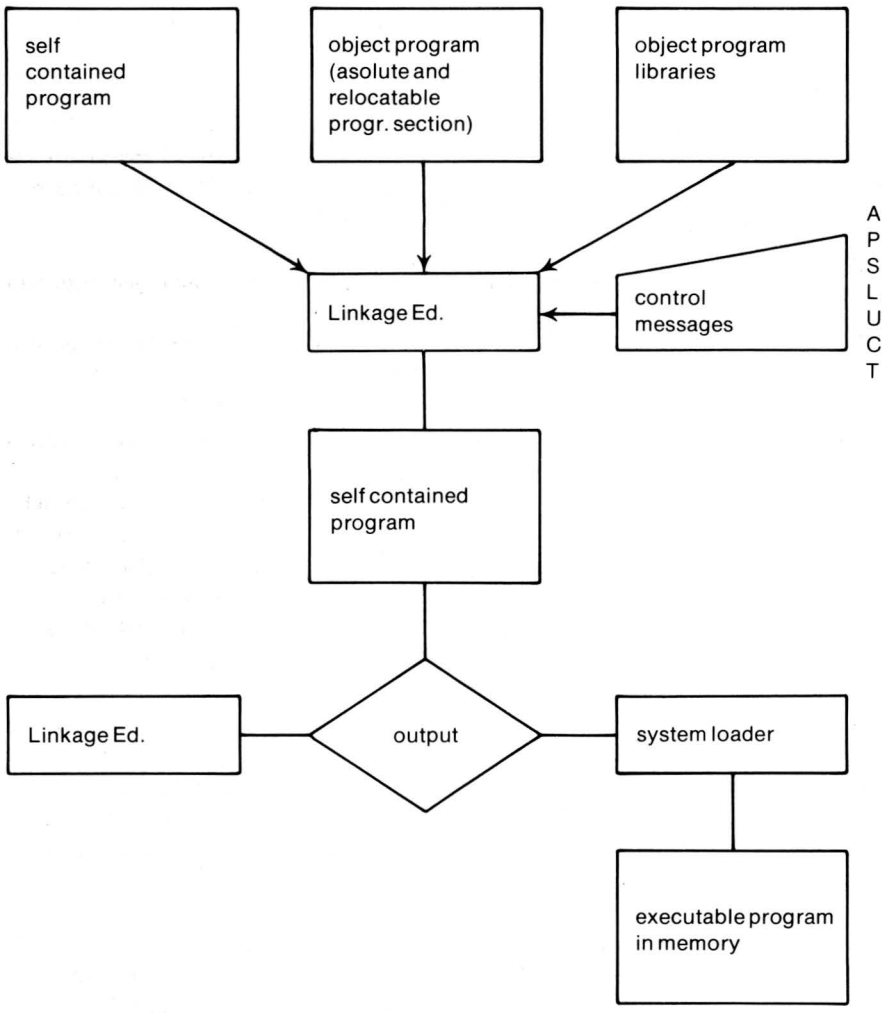
It gives a list of all entry point names together with their numerical equivalent if they are defined. In a link load operation all entry points listed have absolute values.
The format of the listed output is:

<symbol>␣[U|X|R␣<address>|␣␣<hexadecimal value>]

where:

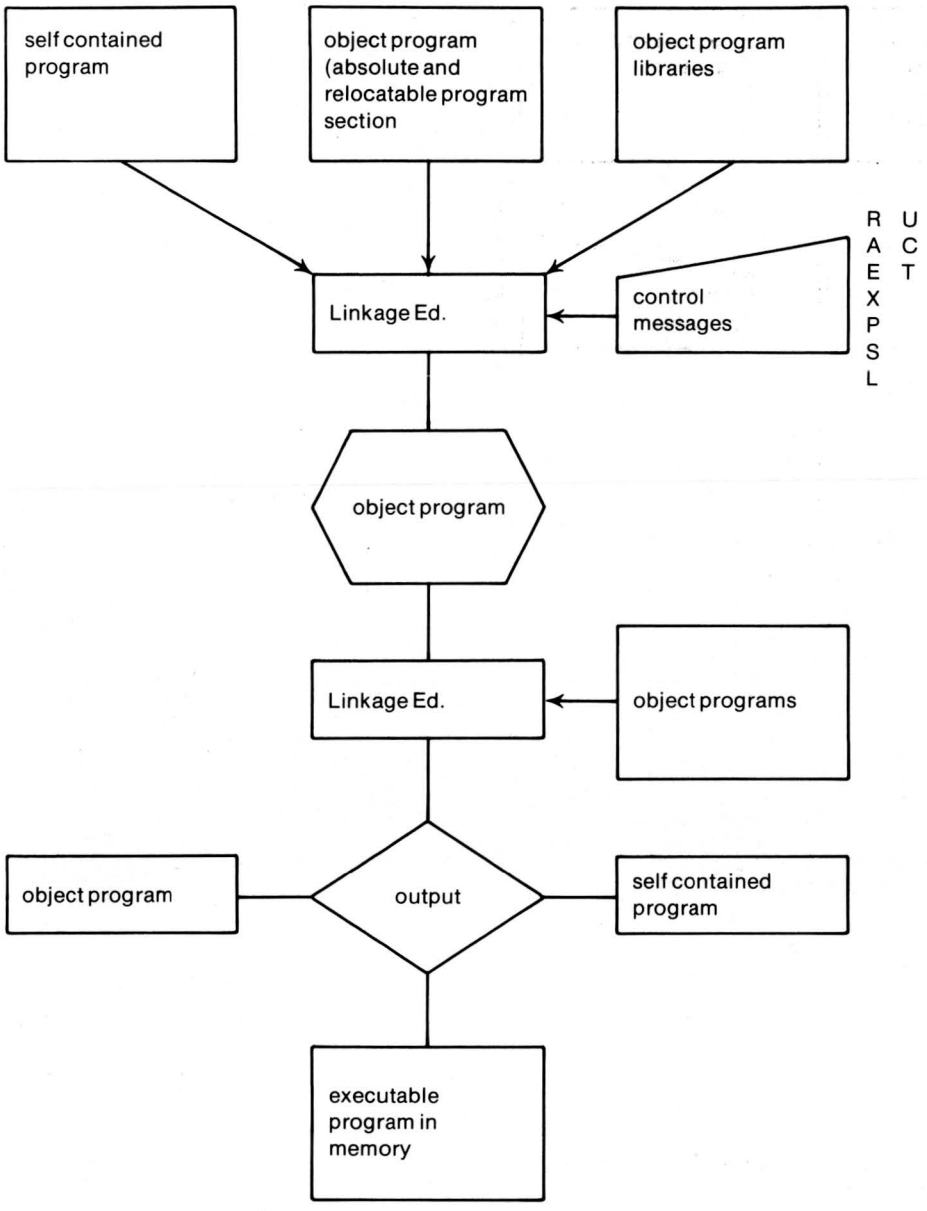
<symbol>	=	the name of an entry point or external reference.
U	=	undefined symbol
X	=	External symbol, i.e. symbol has been declared in an X control message (link edit only).
R	=	relocatable address. If R is present <address> is a relative address in the relocatable program section of the generated program. If R is absent, always in a link load operation, the <hexadecimal value> is the absolute value of <symbol>.

Examples:



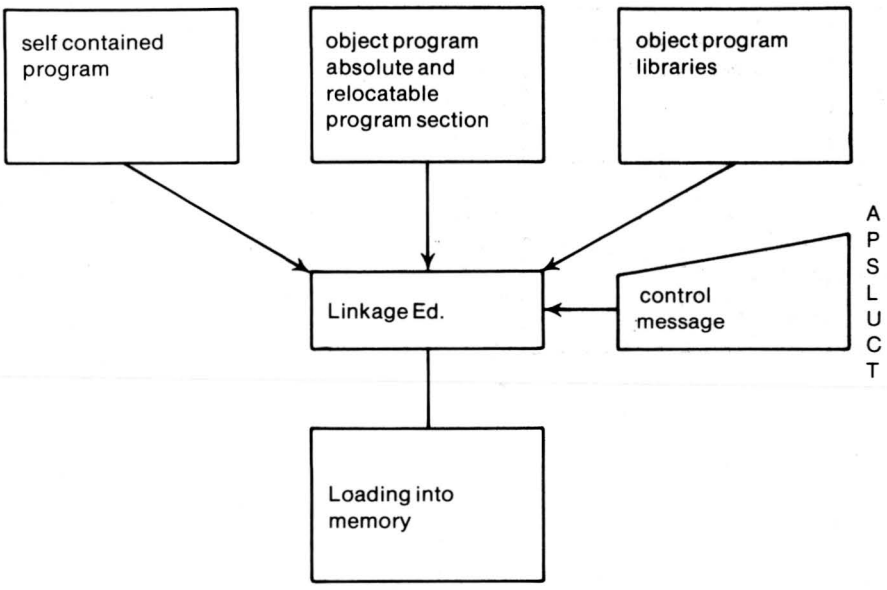
Example 1: Produce an absolute core image program on paper tape (link edit).

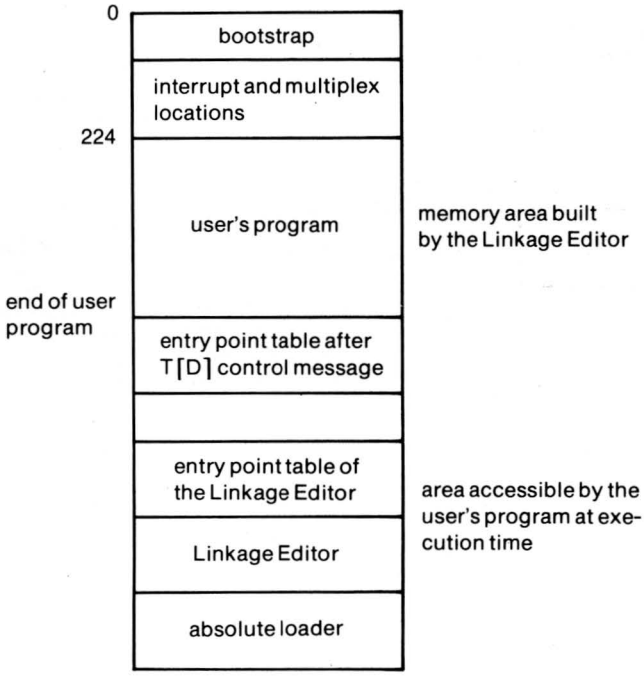
Example 2: Produce an object program on paper tape.



This kind of link edit operation may be immediately followed by a link load operation without having to reload the Linkage Editor. In that case the space occupied by the link edit part of the Linkage Editor, may be used.

Example 3: link load operation





P855/P860 UPDATING OF PROGRAMS

Two standard programs are available for the updating of user programs, TEXT EDITOR and the UPDATE PROCESSOR. The summary shows the main characteristics of both, while the detailed descriptions are given in the next two chapters.

Summary	**Text Editor**	**Update Processor**
Function:	Updating	Updating
Level:	Characters/lines	Lines/Modules (source) Modules (object)
Medium:	Punched tape	Any medium
Operation:	Stand Alone Monitor Control	Stand Alone Monitor Control
Minimum Configuration:	4k memory Operator's typewriter	4k memory Operator's typewriter Punched tape reader and punch; or cassette tape unit and either punched tape reader or punch.

Update Processor

The Update Processor allows the user to update any source program written in ASCII code as well as any P855/P860 Fortran source program by inserting and deleting lines and modules. Object programs may be updated by inserting and deleting modules.

The Processor consists of two modules. One module accepts and executes the control messages, introduced by the operator, which contain the correction parameters.
Once the control messages are taken into account it transfers the updated text line by line to the second module.
In those cases where object modules are to be updated the control messages contain the name(s) of those modules which have to be inserted or deleted.

The updated source program is accepted line by line by the second module whose function is to take care of the writing or punching the received source lines.
Object programs are copied after having been updated by the first module.

The first module, which makes it possible to change the program in memory, is also included in the Assemblers and in the Fortran P855/P860 Compiler.
This makes it possible to execute temporary corrections without changing the actual program on the data carrier.

MINIMUM CONFIGURATION

The Update Processor uses the following minimum configuration:

- 4k memory
- operator's typewriter
- high speed punched tape reader (or cassette tape unit)
- high speed punched tape punch (or cassette tape unit)

It is not allowed to use two cassette tape units so the user must choose between:

- high speed punched tape reader and cassette tape unit, or
- high speed tape punch and cassette tape unit.

FILE CODES

The input and output file codes are chosen in the option control message in which the operator may type in the (hexadecimal) file codes and he may choose the object program output (4-4-4-4 or 8-8).
Which file codes he will choose depends upon the user's configuration.

PROCESSING

The Update Processor consists of two parts. One part that takes care of updating the input modules or programs and a part that takes care of the output of the updated programs on the file codes specified in the option control message described under operator control messages.
Line numbers in the programs and in the control messages must be in ascending order.

The first part, which is also included in the Assemblers and the Fortran Compiler, must be initiated manually by filling the P-register with the processor's start address.

As soon as the processor has been given control it types out on the operator's typewriter:

U:

The operator may now type in the options as described in the option control message, followed by carriage return and line feed (CR.LF).
In case of an erroneous parameter in the option message the processor types out:

U?

thus requesting for a correct option input.

When the option has been accepted the user may introduce his control messages with the parameters he considers necessary. The control messages are described in the relevant section.
Each time the processor needs a new control message it types out on the operator's typewriter:

C:

and waits for a new control message.
The processor also stops reading when an erroneous parameter was entered with a control message, especially when the line number sequence was wrong. The line numbers must be in ascending order. The processor types out:

C?

The operator must now introduce the correct control message.
When an End Of File has been read on the original input file or in the source correction stream the processor types out on the operator's typewriter:

U:

The operator may answer in the following three ways.

.(CR.LF)

which means that the processor will type out EOF, thus terminating the Update process, or the user types in a new option control message

<4 hexa digits>[,4|,8]

and a new Update process may begin, or

; (CR.LF)

The processor prints an EOS mark and next

U:

The user may answer with

. (CR.LF) or

a new option control message.

OPERATOR CONTROL MESSAGES

The following control messages allow the updating of source and object programs. All introduced control messages are terminated with CR.LF.
The control messages for line corrections apply only to source modules as they are irrelevant to object modules.
Line numbers must be in ascending order.

Option control message

In those configurations, other than those having an operator's typewriter as their only I/O device the user must specify certain options.
The syntax of this control message is:

<input file code> <correction input file code> <listing output file code> <punch output file code> [,4|,8]

<input file code> is the file code from which the program to be updated is read.
<correction input file code> is the file code from which the corrections are read.
<listing output file code> is the file code on which the updated program is listed.
<punch output file code> is the file code on which the updated program is punched.
,4 = the program is an object program. The output must be in 4-4-4-4 format.
,8 = the program is an object program. The output must be in 8-8 format.
Remark: If one of the above listed file codes is not used the operator must type in 0 and the corresponding function is not performed.

List control message

L The names of all the modules until EOF are listed.

Copy control messages

. There are no corrections until EOF.
; There are no corrections until EOS. When this mark has been reached the operator may introduce the following control messages.

Correction control messages

M:<name> No corrections are necessary until the source module with name <name> is reached.
S:<name> All records up to the source module with name <name> are deleted.
S: All records are deleted until EOF.
D: <line number 1>
[␣<line number 2>] If both parameters are introduced all lines between <line number 1> inclusive and <line number 2> inclusive are deleted. <line number> is in decimal.
If only one parameter is used the line with that particular number is deleted.
D:<name> All the modules up to module with the name <name> are copied and this module is deleted.
I:<line number> The statements which will now be introduced are inserted on this line and following. The line numbers must be in ascending order.
<line number> is in decimal.
I:<name> The module with name <name> is searched on the correction file and it is inserted in the original program.
If <name> is omitted the user may insert a correction control message.

Text Editor

The TEXT EDITOR program allows the updating of punched tape input on a character or line level. It may be used either as a Stand Alone program or it may be used under monitor control. the initialization procedure is different and will be discussed below.
The TEXT EDITOR executes control messages containing the editing parameters typed in from the operator's typewriter. When all control messages have been processed the updated program is punched or written on the output device.

MINIMUM CONFIGURATION

The TEXT EDITOR uses the following minimum configuration:

- 4k memory
- operator's typewriter

FILE CODES

The file codes have been determined at SYSGEN time. When the TEXT EDITOR is used as a stand alone program the user may type in the file codes he wishes to use by answering the requests for file codes typed out on the operator's typewriter.

PROCESSING

The TEXT EDITOR is called as follows:
Stand Alone:

- Load the start address in the P-register
- Push the START button
- On the operator's typewriter are typed out the following requests:
  MAIN INPUT = (the operator must type in the input file code)
  AUXY INPUT = (type in auxiliary input file code)
  OUTPUT = (type in the output file code)
  LINES PER PAGE = (a page equals the buffer size or the number of lines until an EOF)

  TABULATOR = see Monitor requests
  T: (input of command string)

Monitor Control

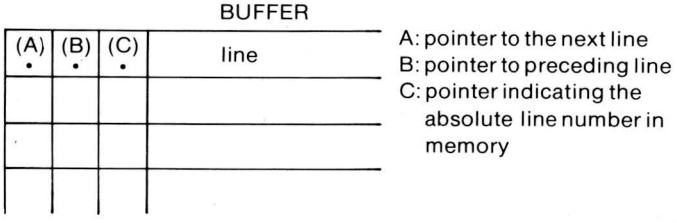
When the TEXT EDITOR is used under monitor control the load procedure is as follows:

- LD
- ST

Control is then given to the TEXT EDITOR and the same messages as described above are printed on the operator's typewriter.

The source text that needs updating is read from the paper tape reader. The processing starts by reading one page from the program introduced from the input file (one page ends when an EOF is encountered or when the edit buffer is filled).
The operator is now able to input control messages from the operator's typewriter which will modify characters and lines in the program page in the edit buffer.
When the page in the edit buffer has been edited a new page from the program is read into the buffer and the operator may edit this page. The contents (the previous page) is destroyed after having been output on the output file.
It is possible to assign a file code at any time during the processing.

A line in the buffer is preceded by three pointers. The first pointer indicates the next line in the buffer, the second pointer the preceding line and the third pointer indicates the absolute line number in memory. Apart from those three pointers there is a text pointer pointing to that character in a line or to the beginning of that line that will be edited. By means of control messages the user may set the text pointer where he wishes provided it is within the edit buffer.



OPERATOR CONTROL MESSAGES

Control messages for editing may be input separately or in a block.
In either case it is necessary to terminate the single control message or the block of control messages with a BEL character.
The TEXT EDITOR prints, after the execution of the control message(s) or when a new control message is asked for T: on the operator's typewriter.
Blanks, carriage return and line feed are ignored in the control message string. The carriage return and line feed may therefore be used to continue a message string on the next line.

It is also possible to use an auxiliary input file, which makes it possible to add information from a secondary file. Three control messages handle the auxiliary input.
Special characters in the control message(s) allow to correct mistakes and to print the number of characters or lines in the buffer.

Control message format

The control messages must have the following format:

[<integer>]<code>[<string 1>][<string 2>]

where:

<integer> = a signed integer in the range -999 to +999 (both inclusive) expressed as a decimal value or a hexadecimal value. In the latter case the hexa value must be preceded by a slash (/). In a hexa value only two hexa digits are taken into account.

<code> = a two letter mnemonic. They are explained under control message description.

<string> = a number of characters. When a character string is given as an operand the string must begin and and with a TAB character and the TEXT EDITOR prints it as a reverse slash (\). ex SS \string\

Special characters

In any editing control message string described below the user may include special characters which provide the user with additional information such as the number of characters or lines in the buffer, the deletion of previous characters etc.

Character	*Meaning*
(equal sign) =	When this character is met in a message string the number of characters present in the buffer is printed on the operator's typewriter.
(colon):	This character causes the number of lines present in the buffer to be printed on the operator's typewriter.
(period) .	The period character allows the printing of the line number in which the text pointer is set.
(horizontal arrow) ←	The last typed in character in a control message or string of control messages, except the BEL character, is deleted when it is immediately followed by the horizontal arrow character.
(vertical arrow) ↑	All characters belonging to the current unterminated control message string (always before the BEL character has been typed in) are deleted. When the user types in the BEL character followed by the vertical arrow the execution of the current message string is stopped. The text pointer is reset to the beginning of the buffer and editing may be restarted.
(End Of Tape) EOT	When this character is met on the input file the reading is stopped and the user may change the tape on the input file when the input file consist of more than one tape. He must type in a BEL character to start the reading of the new tape.

AUXILIARY INPUT

Apart from the input consisting of the program to be edited it is also possible to add source lines from an auxiliary input file

The following three control messages make this possible:

Message	*Meaning*
CI	CHANGE INPUT Through this control message the input is changed from the main input to the auxiliary input or vice versa. The text pointer remains unchanged.
<integer>␣CL	COPY n LINES Copy n lines from the auxiliary input file on the output file with listing on the operator's typewriter.
CM	ADD REMAINING PART Add the information present on the auxiliary file, from the text pointer on, to the output file and switch back to the main input.

INPUT/OUTPUT CONTROL MESSAGES

The following control messages make it possible to read a page from the program to be edited and/or to write or punch this edited page on the output file.

Message	*Meaning*
RP	READ A PAGE The next page is read into the edit buffer and the contents already present in the buffer is destroyed. The text pointer (TP) is moved to the address of the first character of the new page.
[<integer>␣]PU	PUNCH THE EDIT BUFFER <integer> = omitted; 0;>0 The entire edit buffer is output on the output file if the code is preceded buy a zero or when it is omitted. If the code is preceded by a positive integer (+n), only n lines starting from the text pointer are output.
PF	PUNCH FORM FEED CHARACTER Punch and add a form feed character when the buffer contents has been output. The text pointer remains unchanged.

Message	*Meaning*
[<integer>␣]LF	OUTPUT LEADER OR FORM FEED <integer> = omitted; 0;>0 If the integer is omitted or is zero, the text editor will print a form feed character on the output file . If the integer is positive (+n), the text editor will print n zero characters. The text pointer remains unchanged.
[<integer>␣]OR	OUTPUT BUFFER AND READ NEXT PAGE <integer> = >0 Depending upon the value of the integer (n+1) (PU + RP) is performed. The text pointer is placed before the new page in the buffer. The remainder of the old page is destroyed.
[<integer>␣]PR	PRINT BUFFER ONTO OPERATOR'S TYPEWRITER <integer> = omitted; 0; >0 If the integer is omitted or zero the entire buffer is printed onto the operator's typewriter. If the integer is positive (+n), n lines starting from the text pointer are printed. The text pointer remains unchanged.

MOVE TEXT POINTER CONTROL MESSAGES

The following control messages allow to move the text pointer to any place in the edit buffer.

Message	*Meaning*
MB	MOVE TEXT POINTER TO BEGIN BUFFER The text pointer is moved one place before the first character in the buffer.
ME	MOVE TEXT POINTER TO END BUFFER The text pointer is moved to one address behind the last character of the buffer.
<integer>␣MC	MOVE TEXT POINTER N CHARACTERS <integer> = -999≤n≤+999 Move the text pointer n character positions (forward or backward depending upon the positive or negative integer) from the current text pointer position, but never after or before the end or beginning of the edit buffer.
[<integer>␣]ML	MOVE TEXT POINTER TO BEGINNING OF LINE <integer> = omitted; 0; -999≤n≤+999 When the integer is omitted or zero the text pointer is moved to the beginning of the current line (i.e. backward past carriage return and then forward one character). When the integer n<0 the text pointer is moved backward n+1 carriage returns and then moved forward one character. When n>0 the text pointer is moved forward n carriage returns.
<integer>␣AL	MOVE TEXT POINTER TO BEGINNING ABSOLUTE LINE <integer> = the absolute line number This control message moves the text pointer to the beginning of the specified absolute line number.

TEXT MODIFICATION CONTROL MESSAGES

The following control messages allow the updating of characters and lines within the existing text.

Message	*Meaning*
<integer>␣DC	DELETE CHARACTER <integer> = >0 or <0 This control message deletes characters from the present position of the text pointer to the new position of the text pointer which is determined by a moving text pointer control message. If integer>0 the deletion is forward If integer<0 the deletion is backward
[<integer>␣]DL	DELETE LINES integer>0 or <0 or omitted When the integer is omitted the current line is omitted. When the integer is <0, n lines are deleted and the text pointer is moved backward n+1 carriage returns and next forward one character. When the integer is >0, n lines are deleted and the text pointer is moved forward n carriage returns.

Message	*Meaning*
[<integer>␣]SI␣ <string1>[␣ <string2>]	**INSERT CHARACTER STRING** <integer> = <0 or >0 or omitted If integer is omitted the character string, which must be terminated by TAB character, is inserted in the line indicated by the current position of the text pointer. After execution the text pointer is placed after the last character inserted. If the integer is specified the user can insert one character in the line indicated by the current position of the text pointer. The value of the integer is masked to seven bits, and then no operand in the control message.
SS␣<string>	SEARCH STRING string is character string The input string of characters searches forward from the current position of the text pointer on, for the first occurrence of the string in the program. When the string is found the text pointer is positioned after the last character of the string. The line is printed on the operator's typewriter and the user may either stop searching or search for the next occurrence of the string by answering the printed question mark (?) either with * (stop searching) or with & (continue). When the string is not found in the buffer the remaining pages of the program are read, one page at the time, until the string is found or until the input file is empty. The subsequent pages may be punched according to the chosen control message SP or SK (see following control messages).
SP␣<string>	SEARCH STRING AND PUNCH PAGE This control message allows to search for the string specified in this control message. When the string is not present in the buffer so many pages are read from the input file and next punched onto the output file until the string is found. When the string is not found a message is output on the operator's typewriter.
SK␣<string>	SEARCH STRING WITHOUT PUNCHING PAGE This control message performs the same functions as the SP control message except that the pages are not punched. The Editor performs the same functions as in the RP control message. When the search has been unsuccessful the text pointer is set to the beginning of the buffer.

Message	*Meaning*
<integer>␣SR ␣<string1> <string2>	STRING REPLACEMENT <integer> = >0 or <0 or omitted. <string1> is the character string that will be deleted (provided it is found in the buffer). <string2> is the character string that has to be inserted. The text pointer is set behind the last inserted character of the character string 2. Strings have to be terminated with a TAB character.
EN	END EDITING This control message will output the edit buffer on the output file and the remainder of the input file.

PART 5 INTERFACE

Installation Planning

This chapter provides details of the physical characteristics of the P855, P860 and associated equipment, to assist the user when planning an installation. The P855 is described first, then the P860 and finally the peripheral equipment for both CPUs.

CENTRAL PROCESSORS AND OPTIONS

The CPUs can be supplied as either a rack mounting or stand-alone module. Details of the physical sizes are given in a later section.

P855 Central Processor

This includes the instruction set, programmed channel, 8 interrupt levels with 23 lines, and a basic power supply.

P855-001
Central Processor -rack version- with space for 4 memory modules and 8 DCU cards.

P855-004
Memory Module, 4k 16-bit words, 1.2 microseconds cycle time.

P855-011
Stand-Alone Cabinet with control panel for P855.

P855 Options

The following options are available for the CPU.

P855-020
Control Panel for rack version.

P855-025
Real Time Clock, line frequency (20 milliseconds for 50 c/s mains).

P855-026
Crystal Clock, (1, 2, 5 and 10 millisecond versions available on request).

P855-027
Memory Protection.

P855-028
Hardware multiply, divide and double length arithmetic.

P855-030
Power failure detection with automatic restart.

P855-031
Interrupt levels 9-16.

P855-032
Interrupt levels 17-32.

P855-033
Interrupt levels 33-48

P855-040
Memory Increment Data Break (MIDB).

P855-041
Multiplex Option, to control up to 15 devices.

P855-042
Direct Memory Access channel (DMA).

P860 Central Processor

The processor includes the instruction set, programmed channel, 8 interrupt levels with 23 lines, and a basic power supply. The CPU is available in either basic or extended rack modules.

P860-001
Central Processor - basic rack version - with space for 4 memory modules and 8 DCU cards.

P860-002
Central Processor - extended rack version - with space for 8 memory modules and 18 DCU cards.

P860-005
Memory Module, 4k 16-bit words, 0.84 microsecond cycle time.

P860-029
Power Supply - additional - for 16k to 32k 16-bit words of memory.

P860-011
Stand-Alone Cabinet with control panel for P860.

P860-012
Additional Stand-Alone Cabinet.

P860 Options

The following options are available for the CPU.

P860-020
Control Panel for rack versions.

P860-025
Real Time Clock, line frequency (20 milliseconds for 50 c/s mains).

P860-026
Crystal Clock (1, 2, 5 and 10 millisecond versions available on request).

P860-027
Memory Protection.

P860-028
Hardware multiply, divide and double-length arithmetic.

P860-030
Power failure detection with automatic restart.

P860-031
Interrupt levels 9-16.

P860-032
Interrupt levels 17-32

P860-033
Interrupt levels 33-48

P860-040
Memory Increment Data Break (MIDB).

P860-041
Multiplex Option, to control up to 15 devices.

P860-042
Direct Memory Access channel (DMA).

PERIPHERAL EQUIPMENT FOR P855 AND P860

Punched Tape

P801-001
Punched Tape Reader, 333 ch.p.s. (Digitronics 2540EP).

P801-100
Spool Equipment, for P801-001 (Digitronics 6040B).

P801-950
Control Unit, for P801-001.

P803-001
Tape Punch, 75 ch.p.s. (Facit 4070).

P803-950
Control Unit, for P803-001.

P804-001
Tape Punch, 150 ch.p.s. (Facit 4060).

P804-950
Control Unit, for P804-001.

Card Reader

P806-001
Card Reader, 250 c.p.m. (Siemag P115).

P806-950
Control Unit, for P806-001.

Line Printers - high speed

P810-001
Line printer, 356 l.p.m., 80 columns.
Data Products 2310.

P811-001
Line printer, 245 l.p.m., 132 columns.
Data Products 2410 and 2420.

P810-950
Control unit for P810-001 and P811-001.

Plotter

P813-001
Plotter 11" plotting width.
Calcomp 565.

P813-950
Control unit for P813-001

Display

P817-001
Character display, 640 characters.
Philips X1600.

P817-950
Control unit for P817-001

Discs

P821-002
Disc unit fixed head, 64k words.

P821-003
Disc unit fixed head, 128k words.

P821-004
Disc unit fixed head, 256k words.

P821-055
Control unit for P821 units.

P822-001
Disc unit, 2.7M characters, moving head.
Philips X1210.

P822-055
Control unit for two P822-001 units.

Magnetic tape

P831-001 Tape unit
P831-002 Tape unit
P831-003 Tape unit
} 12.5-45 i.p.s., 800 b.p.i., NRZ1, IBM compatible.

P831-055
Control unit for four P831 units.

Cassette tape

P833-001
Cassette tape drive unit, 2-12 i.p.s., 800 b.p.i. phase encoding

P833-950
Control unit for two P833-001 units.

Line printers-low speed

P814-001
Line printer, 21 columns.

P814-950
Control unit for P814-001

I/O Typewriters

P814-001
I/O typewriter, 10 ch.p.s., light duty, with punched tape.
Teletype - ASR33.

P841-002
I/O typewriter, 10 ch.p.s., heavy duty, with punched tape.
Teletype - ASR35.

P841-003
I/O typewriter, 10 ch.p.s., light duty.
Teletype, KSR33.

P841-004
I/O typewriter, 10 ch.p.s., heavy duty.
Teletype, KSR35.

P841-050
Control unit for P841.

OTHER MAIN-FRAME OPTIONS FOR P855 AND P860

P849-050
Control panel with cabinet, cables and connectors.

P849-002
General purpose card with I/O bus connectors and 2 I/O Connectors.
This card can be used for a customer-designed control unit or I/O interface (one I/O connector is already mounted on the card).

P849-004
I/O bus connector extension card for test purposes.
This card can be used to test C.U. cards during maintenance. During a test the card is extended outside the CPU cabinet.

P849-005
Male card connector. A spare connector for mounting on a P849-002

P849-006
Female card connector. A spare connector for mounting on a P849-002.

P849-007
Cabinet-mounted Male connector, to be used for external device cabling. Mounts on the rear of the CPU cabinet.

P849-008
Cabinet-Mounted female connector, as for P849-007.

P849-009
Power supply, for 21 I/O cards, rack version.

P849-011
Cabinet equipment shelf for 21 cards with connectors, rack version. This cabinet includes cabling to connect the I/O bus to the CPU cabinet; it must be mounted in the same or an adjacent equipment rack.

P849-013
Cabinet equipment shelf for 21 cards with wired I/O bus, rack version.

P849-015
I/O bus extender. These cards contain the necessary amplifiers and connectors for extension of the I/O bus beyond the standard cabinet.

P849-016
Standard cabinet for 19" rack (including a 19" rack).

DATA COMMUNICATION EQUIPMENT

P845-050
Multiple line control unit for 8 lines, 50 to 300 bits/second (MELCU).

P845-101
Line termination unit for Teletype ASR interface (LTU).

P845-102
Line termination unit for V21 interface (LTU).

P846-050
Asynchronous line control unit for 600-2400 bits/second (ASYLCU).

P847-050
Synchronous line control unit for 600-9600 bits/second for outplant lines, for inplant lines up to 50.000 bits/second.

P847-101
Cyclic redundancy check (CRC: used optionally in conjunction with the SYLCU).

P848-001
Clockpulse generator for standard speeds for P845 and P846.

P848-002
Clockpulse generator for 133.2 bits/second, 134.5 bits/second and standard speeds for P845 and P846.

P848-003
Clockpulse generator for asynchronous data communication control unit for non-standard speeds for P845 and P846.

P849-014
Power supply for data communication units.

EQUIPMENT FOR DIOS (DIGITAL I/O SYSTEM)

P839-050
Digital input/output control unit (DIOC) for P850, P855 and P860.
Includes
- control unit attaching to I/O bus
- 2 gated 16-bit input words
- 2 buffered 16-bit output words
- 1 call line per word
- 1 response line per word
- TTL level adaption (device connections 5 m. max).

The control unit occupies one slot.

P839-150
Digital input control unit (DIC) for P850, P855 and P860 as for P839-050, but 4 gated 16-bit input words instead of 2 input and 2 output words.

P839-250
Digital output control unit (DOC) for P850, P855 and P860, as for P839-050, but 4 buffered 16-bit output words instead of 2 input and 2 output words.

P839-001
Input buffer for 2 words, includes:
- buffer for two 16-bit input words
- call and response line per word
- TTL level adaption (device connections 5m. max).

The buffer occupies one slot and requires either the P839-050 or the P839-150 and P839-200.

P839-002
Input adapter, 2 words, provides input voltage level adaption between -48 vdc and +48 vcd, the switching level is +1.5 volt. The adaptor occupies one slot and requires the P839-050 or the P839-150 and P839-200.

P839-003
Input adapter, 2 words, provides input voltage level adaption between -48 vdc and +48 vdc, the switching level is +6 volt. The adapter occupies one slot and requires the P839-050 or the P839-150 and P839-200.

P839-004
Input isolator and adapter, 2 words, provides electrical isolation of incoming connections; the incoming bits are mutually isolated.
Absolute values of adaption level are:
HI 12-48 vdc
LO 0- 2 vdc.
The input isolator requires the P839-050 or the P839-150.

P839-005
Input buffer and change-of-state detector, 2 words. The specifications are identical to those for the P839-001, plus detection option. The buffer occupies one slot and requires the P839-050 or the P839-150.

P839-006
Input buffer and adapter, 2 words. The specificatios are identical to those for the P839-002, -001 except TTL-level. The buffer occupies one slot and requires the P839-050 or the P839-150 and the P839-200.

P839-007
Input buffer and adapter, two words. The specifications are identical to those for the P839-003, -001, except TTL-level. The adapter occupies one slot and requires the P839-050 or the P839-150 and P839-200.

P839-008
Input buffer with change-of-state detector and adapter, +1.5 volt 2 words. The specifications are identical to those for the P839-002, -005, except TTL-level. The buffer occupies one slot and requires the P839-050, or the P839-150 and P839-200.

P839-009
Input buffer with change-of-state detector and adapter, 6 volt 2 words. The specifications are identical to those for the P839-003, -005, except TTL-level. The buffer occupies one slot and requires the P839-050, or the P839-150 and P839-200.

P839-010
Output adapter, 2 words, accepts current of 50 mA d.c. in conducting state and 48 volts connection in non-conducting state. The adapter occupies one slot and requires the P839-050, or the P839-250 and P839-200.

P839-011
Output isolator and adapter, 2 words, provides electrical isolation of outgoing connections and of mutually isolated outgoing words. It accepts a current of 50 mA d.c. in conducting state and a 48 volt connection in non-conducting state. The isolator occupies one slot and requires the P839-150 or the P839-250.

P839-200
Power supply for adapter (up to 14 adaption items: i.e. P839-002 -003, -006, -008, -009, -010).

CPU CONFIGURATIONS

Figures 1, 2 and 3 show the different configurations for the two CPU's. For the rack versions, figure 1 will contain either a basic P860 or a maximum configuration P855; Figure 2 will contain the maximum configuration P860. The Stand-Alone cabinet is the same for both CPU's for any configuration.

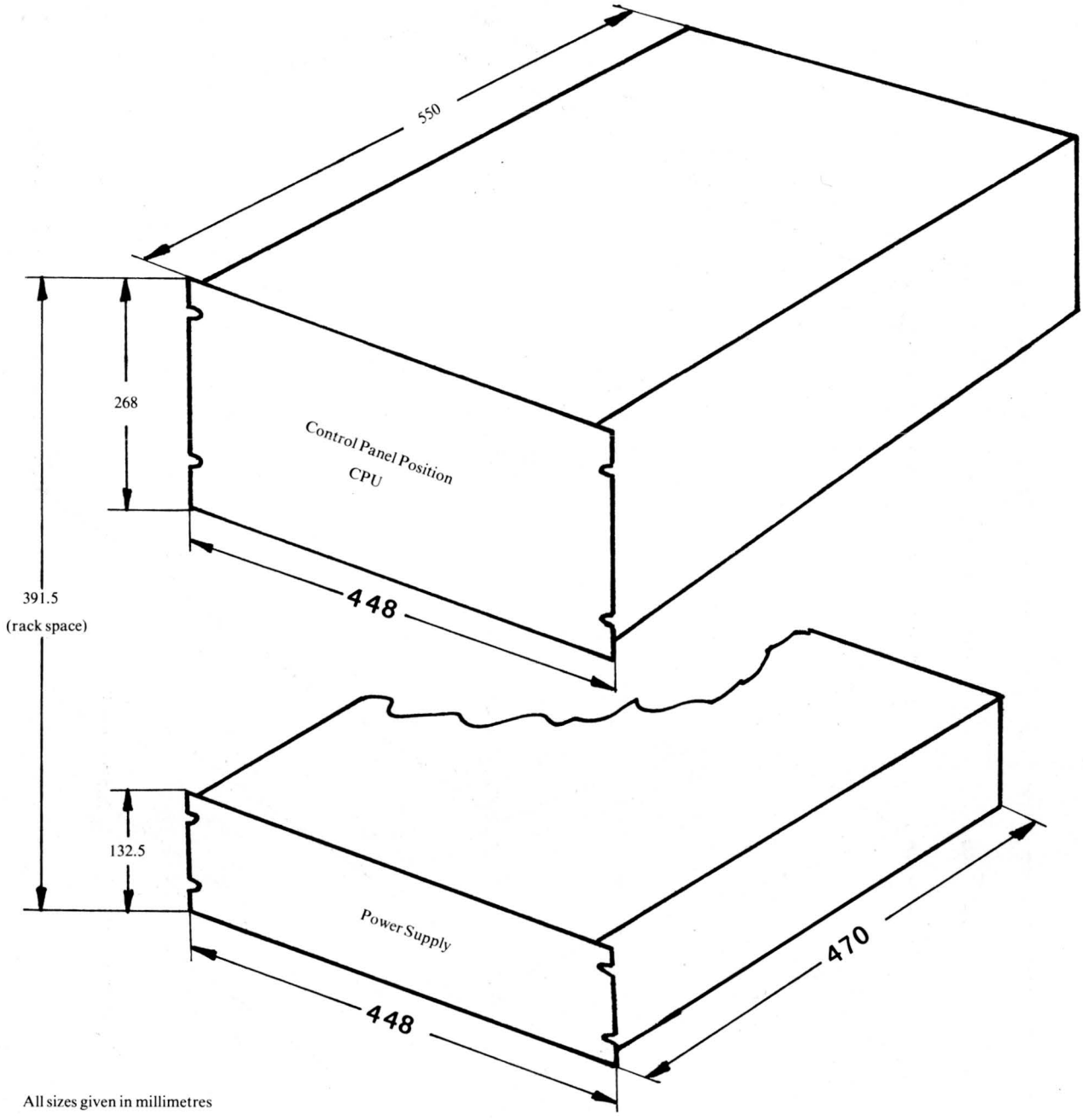


Figure 1 P855 Rack version cabinet, P860 Basic rack version cabinet

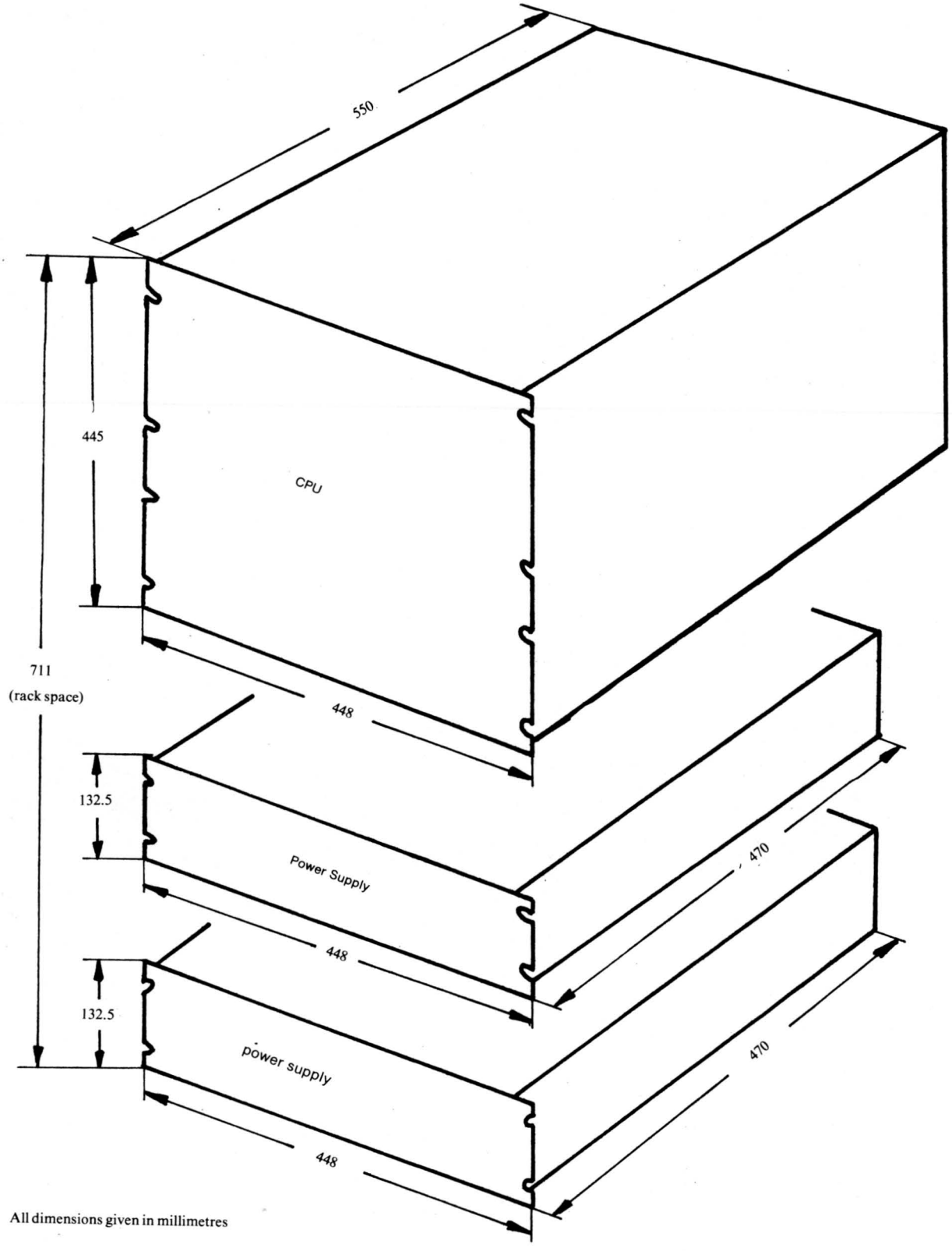
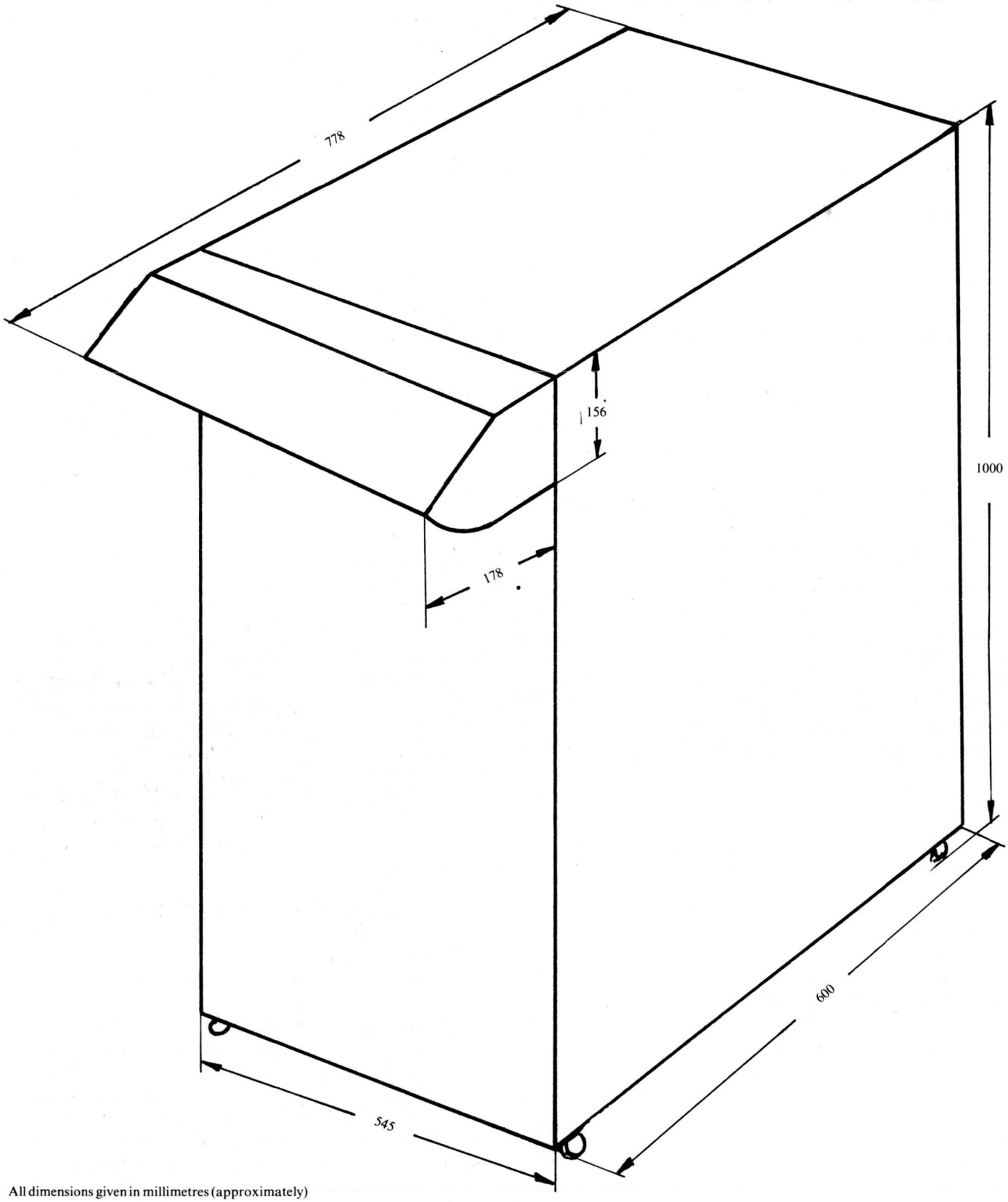


Figure 2 P860 Extended rack version cabinet



All dimensions given in millimetres (approximately)

Figure 3 P855/P860 StandAlone Cabinet

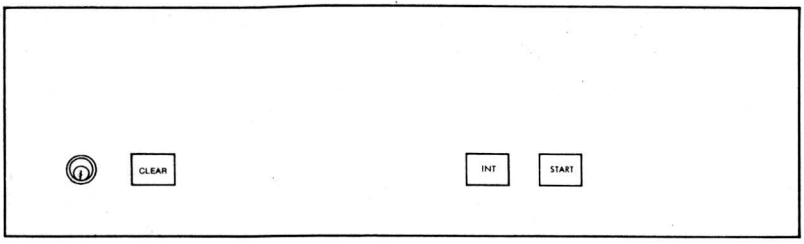


Figure 4 Mini Panel

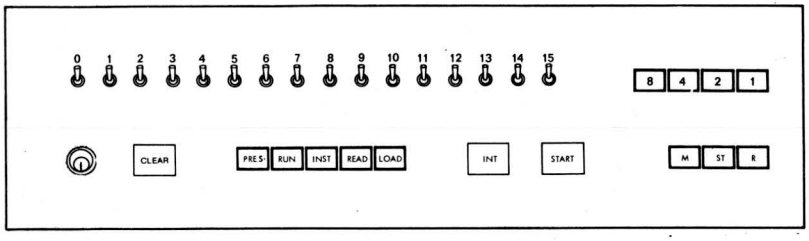


Figure 5 Full Control Panel. Standard on stand-alone models, optional on rack mounting models.

ENVIRONMENTAL REQUIREMENTS

Computer only; other limitations may be applied to peripheral equipment.

Temperature

Operating	0° to +45°C
Non-operating	–30° to +70°C
Transit and storage (in original packing)	–30° to +70°C

Temperature changes (maximum)

Operating	0.5°C per minute
Non-operating	1°C per minute
Transit and storage (in original packing)	1°C per minute

Humidity

Operating	0-85%

Atmospheric pressure (minimum)

Operating	700 millibars
Transit	540 millibars

Vibration

Operating, non-operating, and storage

Frequency	5-150 Hz
Acceleration	.5g maximum
Amplitude	2.5 mm peak to peak

According to BEAMA standard 209/2. 5/69 group II requirements.

POWER REQUIREMENTS

Voltage	110, 115, 220 or 240 Volts ±10%
Frequency	48-63 Hz
Voltage transients	2KV peak amplitude, 100 μsec maximum duration measured at half peak voltage points, and 1 second repetition rate.
Power interruptions	20 ms each 10 seconds
Power consumption	200 Watts for P860 minimum configuration (exclusive power consumption for the power supply).

P841-001 I/O TYPEWRITER (Teletype, ASR33)

Dimensions and weight

See figure 6.

Printing and Stationary Handling

Feed	6 lines per inch. Single or double line feed.
Paper	210 mm (8.5 inches) wide, maximum roll diameter 125 mm (5 inches).
Character Spacing	10 characters per inch, 74 per line.
Legible Copies	Original and one carbon copy.

Punched Tape

Width	1 inch
Format	8-channel, 10 characters per inch.

Power Requirements

Voltage	115 or 220 Volts ±10%
Frequency	50 or 60 Hz.

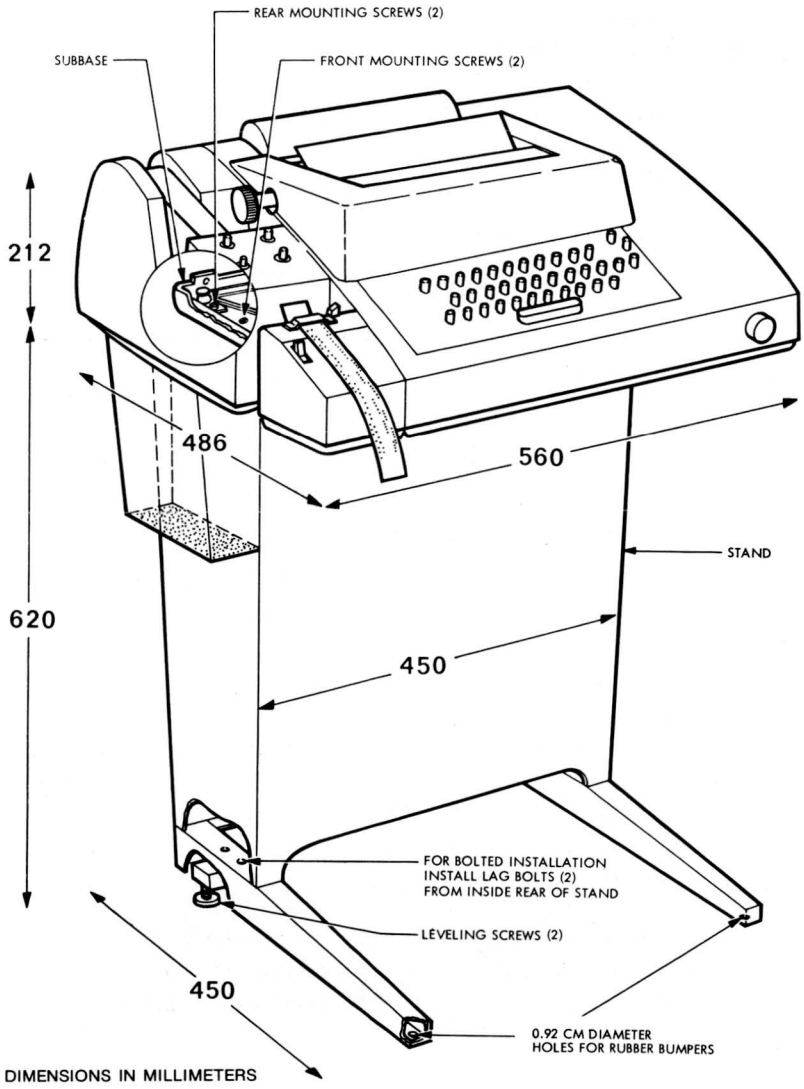


Figure 6 I/O Typewriter (P841-001). Showing installation details and dimensions

P801-001 PUNCHED TAPE READER (Digitronics, 2540 EP)

Dimensions and Weight

front panel width	480 mm (STANDARD 19 inch panel)
front panel height	133 mm (5.25 inches)
mounting holes	Spaced to fit standard RETMA Rack
Depth Behind Panel	200 mm (8 inches)
Protrusion from Front Panel	70 mm (2¾ inches)
Weight	12.7 kg (28 pounds)

Environmental Requirements

Temperature	
operating	0° to 55°C
storage	-55° to +65°C
Humidity	
operating	10 to 90%
storage	0 to 100% without condensation

Power Requirements

Voltage	117 or 220 Volts ±10%
Frequency	48 to 62 Hz.

P803-001 TAPE PUNCH (Facit, 4070)

Dimensions and Weight

Length	432 mm (17 inches)
Width	220 mm (8⅝ inches)
Height	190 mm (7½ inches)
Weight	13.2 kg (29 pounds)

Reels

Diameter	200 mm (8 inches approx.)
Hubs	NAB compatible.

Automatic Cut Off

Power is cut off at -15% of nominal supply voltage.

Power Requirements

Voltage	110/127/220/240 Volts + 15% – 10%
Frequency	50 to 100 Hz
Power Consumption	Average 100 Watts
	Maximum 190 Watts.

P801-100 PUNCHED TAPE HANDLER (Digitronics, 6040 B)

Dimensions and Weight

Front panel width	480 mm (standard 19 inch panel)
Front panel height	222 mm (8¾ inches)
Mounting holes	Spaced to fit standard RETMA Rack
Depth Behind Panel	204 mm (8 inches)
Protrusion from Front Panel	76 mm (3 inches)
Weight	16.5 kg (36 pounds)

Reels

Diameter	204 mm (8 inches)
Hubs	Accepts standard NAB reels

Automatic Cut Off

Power is cut off by 'no tape' or 'broken tape'.

Environmental Requirements

Temperature

Operating	0° to +55°C
Storage	–55° to +65°C

Power Requirements

Voltage	115 or 230 Volts ±10%
Frequency	47 - 63 Hz
Power Consumption	Idle 75 Watts
	Peak 650 Watts

P806-001 CARD READER (Siemag, P115)

Dimensions and Weight

Height (lateral face)	760 mm
(including cover)	858 mm
Width	910 mm
Depth	400 mm
Weight	75 kg.

Environmental Requirements

Temperature range	5 - 55°C
Relative humidity	<90%

Power Requirements

Voltage	110, 117, 127, 220, 245 Volts
Frequency	50 - 60 Hz
Power consumption	0.20 kVA

Cards

Measurements

187.32 mm (±0.13) × 82.55 mm (+0.18 – 0.08)

Thickness	0.178 (±0.01) mm

Material

Standard American punched card paper ($161\ g/m^2 \pm 8g/m^2$).

According to DIN66018 specifications

Temperature	15° - 25°C
Relative humidity	between 50 and 70%

Card Reader Interface

Signals from card reader to control unit

CH	Channel data (12 lines).
ZS	Send data, a pulse referencing the time the data pulses are read out.
CP	Card presence, time during which the card covers the read station.
OPER/	Reader not operable.

Signals from the control unit to the card reader

POC	Pick one card.

P822-001 DISC UNIT (Philips, X1210)

Recording technology

recording method	double frequency
recording medium	standard oxide coated 14 inch disc
bit density	2200 bits per inch (innermost track)
track density	100 tracks per inch
number of read/write heads	2
number of tracks per surface	200 (with 4 spare)
data transferrate	833 K bits per second
disc speed	800 revolutions per minute

Recording capacity

capacity without formatting	62500 bits nominal per track
	25×10^6 bits per drive
formatting	16 or 24 sectors available, depending on cartridge

Error rate

recoverable errors	1 in 10^{10} bits

Power consumption 0.15 kVA

Access times

head positioning (including head settling)

track-to-track	50	milliseconds
full stroke (200 tracks)	260	milliseconds
random average	125	milliseconds

average rotational delay	37,5 milliseconds

Start/Stop time

start sequence	10 seconds
stop sequence	5 seconds

Environmental conditions

temperature

operating	+10 to +32° C
non-operating	−15 to +65° C

relative humidity

operating	20 to 80%
non-operating	5 to 85%

INSTALLATION

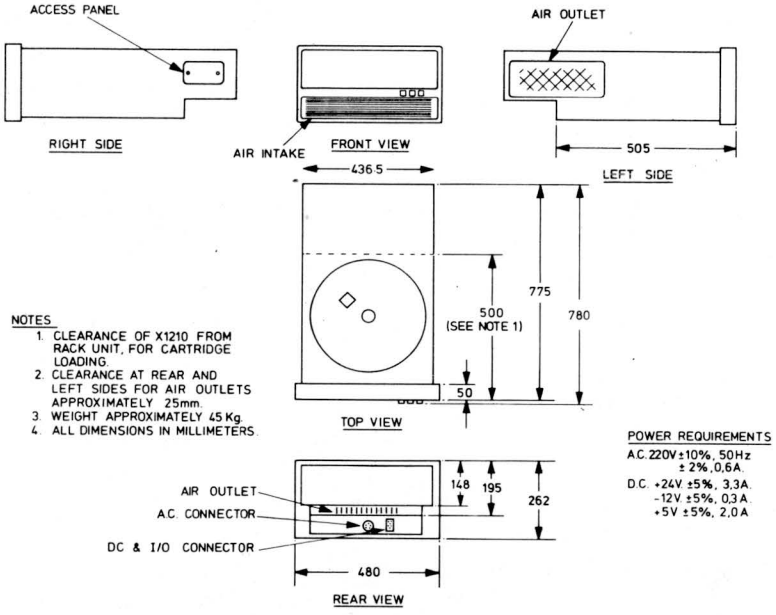


Figure 7 P822-001 Installation

Interface

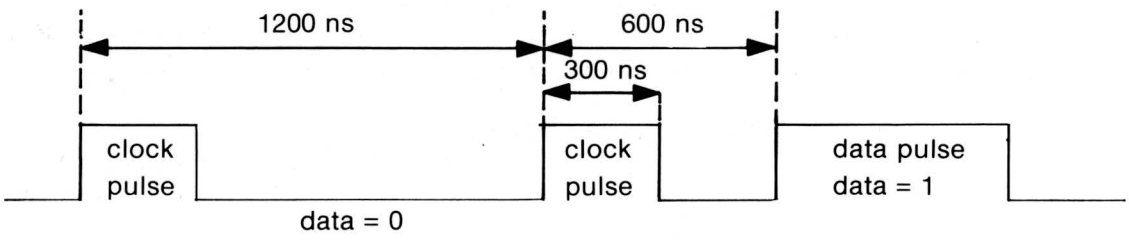
The interface has been tailored to the needs of small computer systems. These systems often require connection of several disc drives. Economy, inherent simplicity and trouble free operation have been the guiding principles to adopt the star type interface. Connection of the DC power supply and control signals should be effected by means of multicore flat cables. These cables should be no longer than 3 meters and the control signal cable should be terminated at the control unit by receiver/transmitter circuits using TTL positive logic.

Two connectors are provided at the rear of the drive to supply the DC and AC power and I/O signals.

Data signals

Special care is given to the read and write data signals, which are transferred by separate coaxial cables.

The information on the transmission lines has the following form:



Input signals
Control of the X 1210 is effected by eight parallel instruction lines (bit 0 to bit 7) together with one of three tag signals. An instruction is only accepted by the X 1210 while one of these tag signals is active. Moreover, the instructions implemented depend upon the tag signal active:

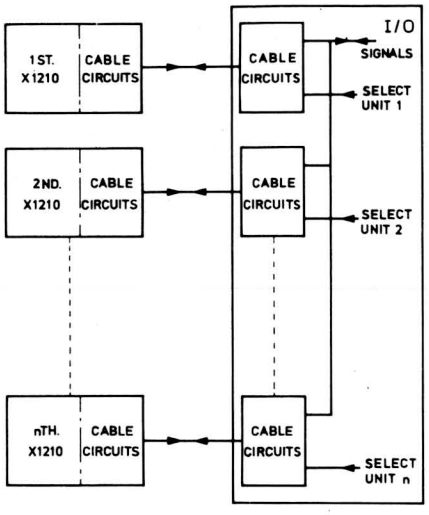


Figure 8 Typical interface structure

- When the difference select tag signal is active, this signal informs the X 1210 of the difference (in BCD form) between the actual track position and the required track position.
- Similarly the head select tag signal informs the drive (via bit 0) which head should be selected. The lower head is selected when bit 0 is active, the upper head when bit 0 is not active.
- The control select tag signal instructs the drive which operation should be performed, as indicated by the particular active instruction bit.

The following table summarizes the instructions supplied to the X 1210 in a short form.

	Instructions		
Instruction signals	Difference select	Head select	Control select
bit 0	1	1	write enable
bit 1	2	not used	read enable
bit 2	4	not used	seek forward
bit 3	8	not used	not used
bit 4	16	not used	erase enable
bit 5	32	not used	seek reverse
bit 6	64	not used	return to zero
bit 7	128	not used	not used

Note: The difference select instructions are shown in decoded form.

Output signals
The X 1210 supplies the following output signals:

Index signal	a pulse generated once per disc revolution.
Sector signal	pulses generated 16 or 24 times per disc revolution, dependent on the type of the cartridge used.
Unit ready signal	this indicates that the X1210 is ready to operate i.e. a cartridge has been inserted, the drive is locked in the rack and the disc has been run up to speed.
On cylinder signal	this indicates that the heads have been positioned and stopped.
Unit unsafe signal	this indicates a fault condition due either to a wrong input command or to failure within the X 1210.

P817-001 CHARACTER DISPLAY (Philips, X 1600)

Dimensions and Weight

	display	*keyboard*
Height	435 mm	80 mm
Width	544 mm	440 mm
Depth	480 mm	200 mm
Weight	app. 25 kg.	app. 10 kg.

Environmental Requirements

Temperature	0 - 45°C
Rel. humidity	20 - 80%
Shock resistance	up to .25g
Pressure	860 - 1060 millibars

Power Requirements

Voltage	110 or 220 V ± 10%
Frequency	50 or 60 Hz ± 2%

P811-001 LINE PRINTER (Data Products 2410)

Specifications

Print Rate (Lines per minute using 64 character drum)
245 LPM - 132 columns
290 LPM - 120 columns
356 LPM - 96 columns
460 LPM - 72 columns
650 LPM - 48 columns
1110 LPM - 24 columns

Number of Characters
64

Type and Size of Characters
ASCII, open Gothic print characters with printed symbols typically 0.095" high and 0.065" wide

Character Spacing (horizontal)
0.100" ± 0.005" between centers. Maximum possible accumulative error for nominal spacing ± 0.010" per 132 character lines

Line Spacing (vertical)
Operator adjustable -
0.167″ ± 0.010″ (6 lines/inch)
0.125″ ± 0.010″ (8 lines/inch)
Each character within ± 0.010″ from a mean line through the characters

Hidden Lines
Line visible after 10 lines of print

Character Synchronization
Variable reluctance pick-offs sensing drum position

Paper Dimensions
Standard fanfold, edge-punched paper 4″ to 19-7/8″ wide with 11″ between folds

Drum Speed
1760 RPM NOMINAL (64 character drum)

Line Advance Time
20 Milliseconds (max)

Paper Slew Speed
13 inches per second

Characters per Line
132

Paper Type
Single copy, 15 lb. bond minimum weitht. Multi-copy up to six parts, 12 lb. bond with single-shot carbon

Paper Loading
Paper loaded directly on tractors from the suppliers box. No paper threading necessary

Print Ribbon
14.5″ wide roll of 28 yards of .004″ thick inked ribbon

Paper Feed
Pin-feed tractors for 12.7 mm. (½″) hole center, edge-punched paper. Adjustable for any paper width from 101.6 mm. (4″) to 504.8 mm. (19-7/8″)

Mechanical Controls
3-position lever for multi-copy thickness adjustment
Rotary knob for vertical paper adjustment
Horizontal paper tension via vernier thumb wheel

Format
Top-of-form control, single line advance and perforation step-over

Signals

Input
7 Data lines
1 Data Strobe
1 Input Connection Verification

Output
1 Ready line
1 Demand line
1 On Line
1 Input Connection Verification

Ribbon Changing
Completely accessible and easily removed upon opening of drum gate

Buffer
24 characters (one line segment)

Control Panel Switches
3 momentary toggle switches for Paper Step, Top Of Form and On-Line/Off-Line

Power (Single Phase)
0.5 kVA 117VAC ± 10%, 60 Hz, 50 Hz ± 2 Hz

Weight
Printer - 260.8 kg.
Shipping Pallet - 45.4 kg.

Dimensions (in millimetres)
Height - 1168.4 (46″)
Width - 1231.9 (48.5″)
Depth - 622.3 (24.5″)

Temperature (°C)
Operating 10 - 43
Non-operating -18 - 65.5

Humidity

Operating
30% - 90%
without static eliminator option
5% - 90%
with static eliminator option

Non-operating
5% to 95%

User System Interface
The interface lines between the standard printer and the user system are listed below and defined on the next page. Additional interface lines are required when selected option features are included with the standard printer.

Signal Name	*Signal Source*
Ready	Printer
Input Connection Verification	Printer
Demand Line	Printer
ON LINE	Printer
Data (7 lines)	User
Data Strobe	User
Input Connection Verification	User

The timing diagram in Figure 9 describes the relationship between the above interface lines. Either positive or negative logic levels can be accommodated. Unless otherwise specified by the customer the standard logic levels are typically +5 volts for logical one and 0 volts for logical zero.

Interface Signal Definitions

signal	*Definition*
READY LINE	This signal is furnished to the user system to indicate when the printer is ready to be put 'ON LINE' by the printer operator. When READY is true, the following are true: (a) Power is on. (b) The printer drum gate is closed. (c) Paper has been loaded. (d) An overtemperature condition does not exist in the paper drive motor.
ON LINE	This signal is furnished to the user system to indicate when the printer has been put 'ON LINE'. When ON LINE is true the following are true: (a) The Ready light is on. (b) The printer operator has actuated the On Line switch.
DEMAND LINE	This signal is provided to synchronize data tranmission between the printer and the user system. The DEMAND LINE signal requests a character from the user. The DEMAND LINE remains true until a data strobe is received. It is disabled while the character is stored in memory and during the print operation. The DEMAND LINE can only come true after the operator has placed the printer ON LINE.
DATA STROBE	This line is controlled by the user to define when the information on the data lines is to be accepted by the printer. Each time a Data Strobe occurs the printer samples the data lines and the demand line is made false while the character is stored. Once the appropriate number of characters have been transferred to the printer, the demand line remains false while the characters are processed.
DATA LINES	These seven lines provide a transfer path for the ASCII character set.
INTERFACE VERIFICATION	Two pins on the interface connector are jumpered together to allow a remote user to verify that the interface connector is attached.

Interface Signal Characteristics

Signals between the user and printer are transmitted over twisted pair wires by signal receivers and transmitters in the printer. The transmitter's output impedance acts as a series terminator for the twisted pair. The receiver and transmitter characteristics are listed below.

Transmitter Characteristics

POSITIVE DRIVER (Output Impedance -100 Ω)

Logical 1 Output	+8V max, +3V min	Load Resistance = 10K to Ground
Logical 0 Output	+.4V max	Load Resistance = 10K to Ground
Signal Transition Rate	40V/μsec (Typ)	Load Resistance = 10K to Ground Load Capacitance = 220 pf

NEGATIVE DRIVER (Output Impedance -100 Ω)

Logical 1 Output	−8V max, −3V min	Load Resistance = 10K to Ground
Logical 0 Output	−1V max	Load Resistance = 10K to Ground
Signal Transition Rate	40V/μsec (Typ)	Load Resistance = 10K to Ground Load Capacitance = 220 pf

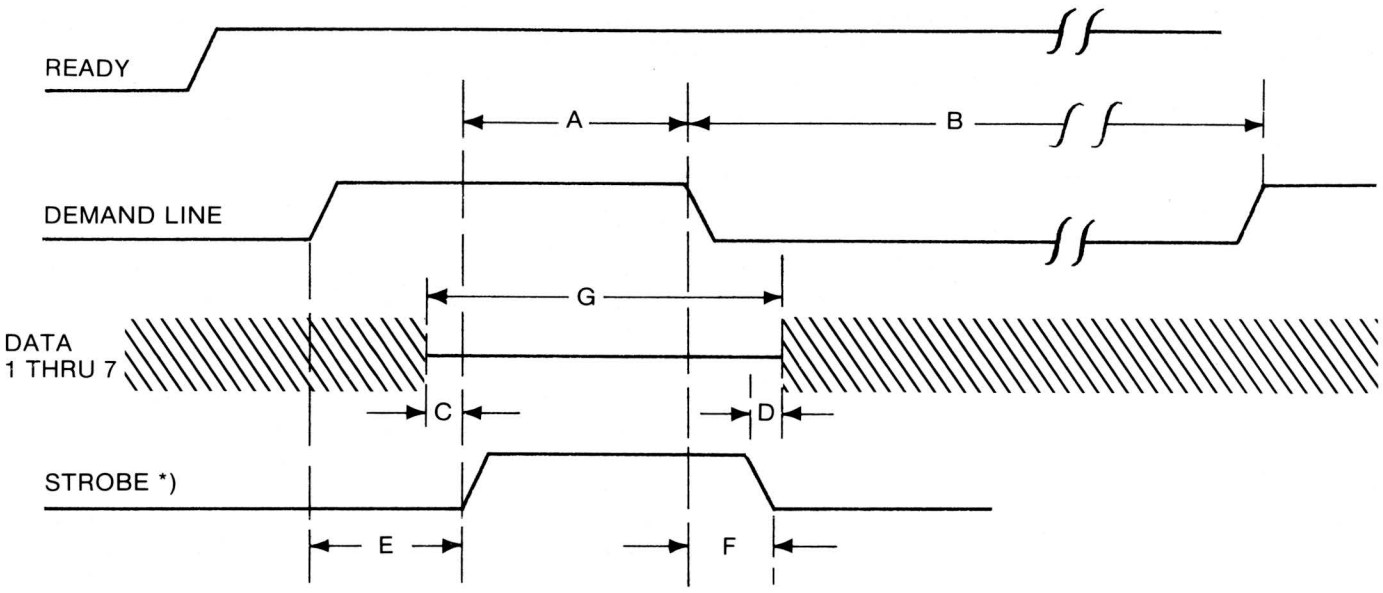
Receiver Characteristics

POSITIVE LOGIC (Input Impedance -10K)

Logical 1	+10V max/ +3.0V min
Logical 0	+1V max/0.0V min
Threshold Voltage	½ of Logical 1

NEGATIVE LOGIC (Input Impedance -10K)

Logical 1	-10V max/-3.0V min
Logical 0	-1V max/-0.0V min
Threshold Voltage	½ of Logical 1



A = 500 nsec max
B = 1.0 μsec min
C & D = 50 nsec min
E = 0+ min
F = 10 nsec min, 200 nsec max (without reducing transfer rate)
G = Data Settled
▨ = Undefined Area

Note 1: The Interface is designed for a Demand - Response system. The strobe line must be brought up after the data lines are settled and Demand Line is active. The Strobe should then reset on the fall of the Demand Line.

Figure 9 Timing diagram at printer interface connector

P813-001 PLOTTER (CalComp 565)

Specifications

Speed:	Drum axis: 300 steps per second maximum. Carriage axis: Same as drum axis. Pen: 10 operations, 5 up and 5 down per second maximum.
Step sizes:	0.01 or 0.005 inch, or 0.1 mm on both drum and carriage.
Resolution:	±1 step on either axis over entire 120-foot roll of paper.
Inputs:	Pulses of either positive or negative polarity with amplitude in excess of 10 volts, rise time less than 10 microseconds, minimum width of 4 microseconds, from a source impedance of less than 500 ohms (input circuits may be modified for specific pulse characteristics).
Power:	105 to 125 volts, 50 or 60 cycles, single-phase (1.5 amps at 115 volts).
Weight:	15 kg. (33 lbs).
Size:	Width – 457.2 mm. (18″) Depth – 374.6 mm. (14¾″) Height – 247.7 mm. (9¾″) to top of pen holder.
Paper size:	Roll chart paper - 304.8 mm. (12″) wide, 33.6 mtr. (120 feet) long, sprocket holes 3.3 mm. (0.13″) diameter and 3.3 mm. by 7.2 mm. (0.130″ by 0.282″).

Installation

The P813-001 Digital Incremental Plotter is shipped in a single packing container. Installation of the unit is simple and requires no special tools or test equipment. It is completely self contained and may be operated on any level surface that is not subject to excessive vibration.

Unpacking

The P813-001 is shipped completely assembled except for the pen assembly, which is packed separately within the shipping container. A roll of chart paper is installed on the rear chart spool. Separate rolls of chart paper are supplied in separate boxes within the shipping container. To unpack the unit, proceed as follows:

a. Remove the top half of the molded styrofoam liner that surrounds the plotter and accessories. Lift the plotter and accessories from the lower piece of styrofoam packing, remove the cellophane wrapping, and the plotter is ready for installation.
b. Remove the accessory packages; check items against the packing slip.

Note: It is recommended that the shipping container, liner, and accessory boxes be stored for reuse whenever the instrument is transported between operating sites or returned to the factory for service.

Inspection
After unpacking the instrument, check for signs of physical damage in shipment. Make certain that all front panel controls are secure on the control shafts. Check that the pen carriage can be moved freely by hand across its track, and that the drum can be rotated manually on its axis.

Cable Fabrication
A female cable connector is supplied with the unit. This connector mates with chassis connector P5 on the rear of the P813-001. A signal cable of desired length to interconnect the plotter and the digital signal source can be made by following the signal nomenclature for connector P5 as shown in the service documentation.

Installation of Chart paper
To install the chart paper shipped on the rear chart spool, pull the end of the paper over the drum so that the holes on both edges of the paper engage the sprockets on the drum. Guide the chart paper under the carriage rods and behind the tear bar. The chart paper winds on the take-up spool from the back; fastens the end on the spool with Scotch tape provided in the accessory kit. Using the DRUM FAST RUN switch, wind a few turns onto the take-up spool.

Note: If the paper does not lie smoothly against the drum, or if it tends to pull diagonally so that the pin holes do not engage properly, check the feed spool adjustment. This condition may result from large changes in humidity that have caused the paper roll to shrink or expand slightly.

To remove chart paper, grasp the paper roll and press the left-hand idler spool to the left. This compresses the spring on the idler spool allowing the roll to be removed. To insert a new roll of chart paper, see Volume 2.

Installation of Pen
Ballpoint pens normally supplied with the recorder are black, blue, red, and green. To assemble the pen, insert the desired color pen into the plunger, then insert the pen and plunger into the holder and install the threaded cap. Align the key on the holder with the key slots in the carriage and press the pen assembly into the pen mounting. Tighten the knurled nut on the bottom of the pen assembly.

Installation Precautions
The P813-001 depends upon free circulation of air under the base plate for proper cooling. Do not place the unit on top of any loose papers or cloth. Loose materials of this type can block the ventilating louvres in the base plate and cause overheating. In addition, the unit should not be placed on top of any other heat-producing equipment.

P839 DIGITAL INPUT/OUTPUT SYSTEM (DIOS)

The Digital Input Output System (DIOS) is a general purpose system which permits to interface any external digital equipments to Sagittaire computers.

The main function of the DIOS is to control the exchange of 16-bit data words in both directions via the programmed channel.

There are three types of DIOS control units (basic DIOS). Each of them handles four 16-bit words in the following way:

P839-150: DIC : Control of 4 input words - I
P839-250: DOC : Control of 4 output words - O
P839-050: DIOC: Control of 2 input and 2 output words - M

The output words are always buffered, the input words are gated.

Extra functions are available (Extended DIOS):

A – 16 bit input buffer register B – Change of state detection	For input lines
C – Level adaptation D – Galvanic isolation and level adaptation	For input or output lines

DIOS to CPU Interface
It is the standard I/O bus interface.

Interface between basic DIOS and external equipment
The input and or output lines of the basic DIOS are either attached directly to the external equipment, or connected to the optional electronic part when an 'extended DIOS' is used.

Input lines

CALA/ CALB/ CALC/ CALD/	External call signals asking for a data exchange on the corresponding A, B, C or D group of lines.
EA00 EA15	Group of 16 input lines A (only for models I and M)
EB00 EB15	Group of 16 input lines B (only for models I and M)
EC00 EC15	Group of 16 input lines C (only for model I)
ED00 ED15	Group of 16 input lines D (only for model I)

Output lines

AOK, BOK, COK, DOK — Response signals from the control unit indicating the exchange duration on the corresponding A, B, C and D group lines, has been performed. The trailing edge of these signals may be delayed with a delay preset between 0 and 20 μsec.Long term stability of the delay: 20%.

SA00 … SA15 — Group of 16 output lines A (only for model O)

SB00 … SB15 — Group of 16 output lines B (only for model O)

SC00 … SC15 — Group of 16 output lines C (only for models O and M)

SD00 … SD15 — Group of 16 output lines D (only for models O and M)

Interface between DIOS and external equipment

The voltage levels from or to the miscellaneous external devices may be included in a wide range so the first function of the interface is the voltage adaptation between the C. U. and the external devices.

If the ground voltage of the device is very different from the control unit ground voltage or if the power supply or transmission lines are noised at high level the interface shall be realized by means of electrical isolation.

Basic DIOS system

The minimum configuration of the DIOS system consists only of logic circuitry and provides DTL and TTL interface to external equipment.

The DIOS control unit handles four 16 bit data words by means of standard INR or OTR instructions. Thus 4 different addresses are needed per control unit. For that reason the 6-bit DA field of I/O instructions is divided into two parts: the 4 left bits are used to address the control unit itself, the 2 least significant bits indicate the word concerned by the transfer.

Two possibilities of use are offered:

a. Software control:

The data transfer is directly controlled by the program using a single I/O instruction per word exchange.

b. External control:

From the external system a seventeenth line, to each word, permits to ask for an exchange on the corresponding data path. When one or several of these lines are activated, DIOS generates an interrupt signal to the CPU. Before a data exchange, it is possible to program a special INR instruction to know what word(s) is (are) asking for an exchange.

Extended DIOS system

A: Input buffer

The input buffer consists of:

- two 16-bit registers for model M
- four 16-bit registers for model I

The buffer receives information from the external equipment or from the electronic part when an 'extended DIOS' is used. It is loaded when:

- an external CALL signal (external control) appears
- a response signal from the control unit (AOK, BOK, COK or DOK) for software control appears.

The information is available until a new 'CALL' or 'response' signal appears.

The output of a 16-bit buffer is connected to the 16 corresponding input lines of a BASIC DIOS system.

B: Change-of-state detection

The change-of-state feature can be added for each input word.

If one or more of the 16 data lines switch from 0 to 1 state, a signal is sent to the external CALL line of the basic control unit. The level of the '1' state will be defined later.

C: Level adaptation

With the level adaptation electronic circuits, the signals at the interface level with the external device, may have the following characteristics:

Input signals:

first option

first option

Low level $-48V \leqslant VIL \leqslant +0.4V$

High level $+2V \leqslant VIH \leqslant +48V$

Second option

Low level $-48V < VIL < 4.5V$

High level $7.5V < VIH < +48V$

In both cases

I sink at low level $\leqslant$ 1,8 mA

I load at high level $\leqslant$ 1 μA

Output signals:

Current amplitude: 50 mA at 0.4V low level

High level voltage range 2.4 to 48 volts at 0.1mA.

D: Electrical isolation

This function is required when the external device may cause trouble in the functioning of the system.

Voltage isolation shall be better than 380 Volt rms

Switching speed up to 10k bits/second.

Electrical interface

Input (on two wires)

0V VIL $\leqslant$ 2V

12 V $\leqslant$ VIH $\leqslant$ 48 V I load max 50 mA

All the input bits are insulated from each other.

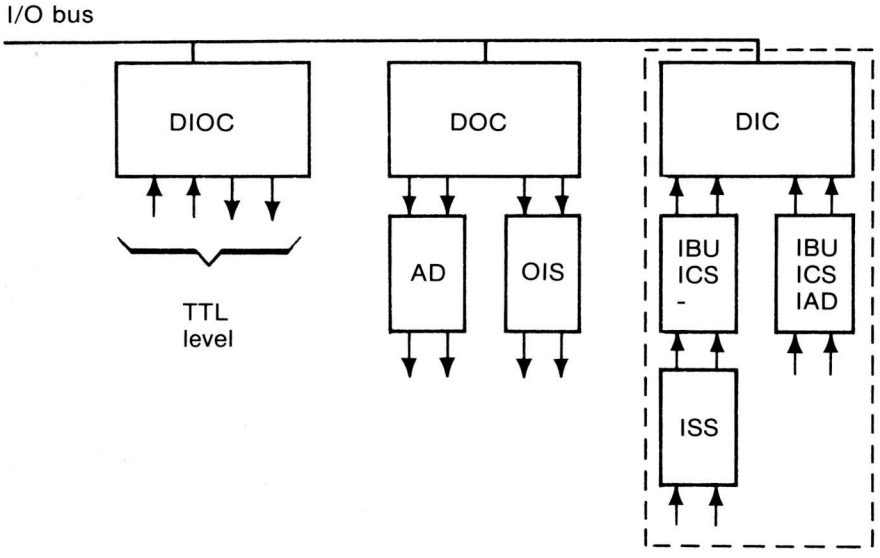
Output

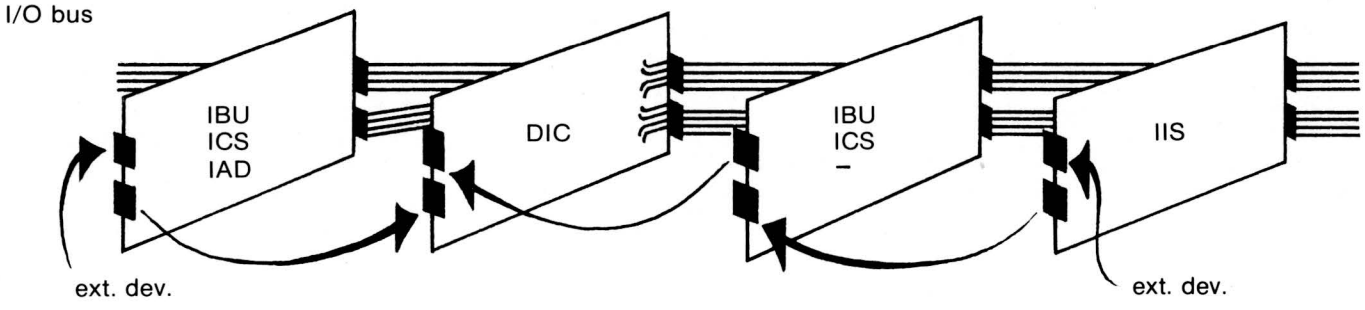
0V VOL $\leqslant$ 0,4 V (I sink max 50 mA)

+ 12 V $\leqslant$ VOH $\leqslant$ + 48 V (I load max 0.1 mA)

The value of VOH with I load = 0 will be equal to the supply voltage provided by the user (i.e. the value of this supply voltage will be included between 12 V and 48 V).
All the output words are isolated from each other.

Example of a DIOS configuration





Key

IBU Input buffer (2 words)
ICS Input change-of-state (2 words)
IAD Input adaption (2 words)
IIS Input isolation (2 words)
OIS Output isolation (2 words)

I/O Commands

OTR: Output data word

0	1 0 0 0	R1	0 0	DA
1	4	3	2	6

R1 : indicates the register from which data are exchanged during the instruction

DA: Device address
Bits from 10 to 13: Controller address
Bits 14 and 15:
0 0 Word A
0 1 Word B
1 0 Word C
1 1 Word D

This instruction is always accepted except for DIOS model I.

During the execution of an OTR instruction the following operations occurs:

- the condition register CR of the CPU is reset to zero
- the contents of R1 are loaded into the output buffer specified by the bits 14 and 15 of the DA field
- If an external call, relative to the concerned word, was pending, the corresponding flip flop is reset to zero.

- If active, the interrupt signal is reset, unless the call flip-flop of another word is set.
- A response signal is sent to the external equipment.

INR: Input data word

0	1 0 0 1	R1	0 1/0	DA
1	4	3	2	6

R1 : indicates the register into which data are exchanged during the instruction.
DA: Device address
Bits from 10 to 13: control unit address
Bits 14 and 15:
0 0 Word A
0 1 Word B
1 0 Word C
1 1 Word D
bit 9:
1: bits 14 and 15 not significant
0: bits 14 and 15 specify which words to be input

This instruction is always accepted except for DIOS output modules.

During the execution of an INR 'data' the following operations occur:

- the condition register CR of the CPU is reset to zero.
- the contents of the 16 input lines specified by the bits 14 and 15 of the DA field are loaded into the R1 register
- If an external call relative to the concerned was pending, the corresponding flip-flop is reset to zero.
- if active, the interrupt signal is reset, unless the calling flip flop of another word is set
- a response signal is sent to the external equipment.

INR: Input word address

0	1 0 0 1	R1	0 1	DA
1	4	3	2	6

R1 : indicates the register into which the address is loaded during the instruction
DA: Device address
Bits from 10 to 13: Controller address
Bits 14 and 15: not significant

This instruction is always accepted.

During the execution of an INR 'address' the following operations occur:

- the condition register CR of the CPU is reset to zero
- the states of the calling flip flops are loaded into the R1 register in the following way:

	A B C D
12	4

A one bit, in the field 12 to 15, indicates that the corresponding flip-flop is set.

Programming Rules
At wished timing, a scanning of digital input or output is performed by using successive I/O data instructions:

```
INR   QA
INR   QB
OTR   QC
OTR   QD
```

(Q being the 4 bit control unit address).

MODULAR INPUT/OUTPUT SYSTEM (MIOS - PC1800 - D)

MIOS makes it possible to connect a variety of devices in almost all possible application fields to the P855 or P860 computer. MIOS is composed of different parts: MIOS - 4S, MIOS - 4D and MIOS - 4A.

MIOS - 4S is to be used in applications like process control, where large amounts of analog inputs of various signal levels are involved.
MIOS - 4D is designed for simple applications like X-ray, where small amounts of digital input and output, and possibly analog output is involved. MIOS - 4D is not designed for analog input.
MIOS - 4A is an extension of MIOS - 4D and is able to handle small amounts of digital input and output, and possibly small amounts of analog output and input. MIOS - 4A does not include standard items to connect ADC's, sample and hold units, data amplifiers and subscanners to the system.

In general, MIOS performs such tasks as:
a. Signal acquisition and conditioning (which includes filtering and rationalisation of input levels).
b. Sequential interrogation of the various inputs.
c. Analog-to-digital conversion of inputs, representing physical quantities for acceptance by the digital computer as data words.
d. Provision of outputs for display, control or data acquisition purposes.

The control unit card is inserted in the standard C.U. sockets of the I/C bus and will drive up to 16 MIOS modules.
The information to and from the system is led through cables and cable connectors.

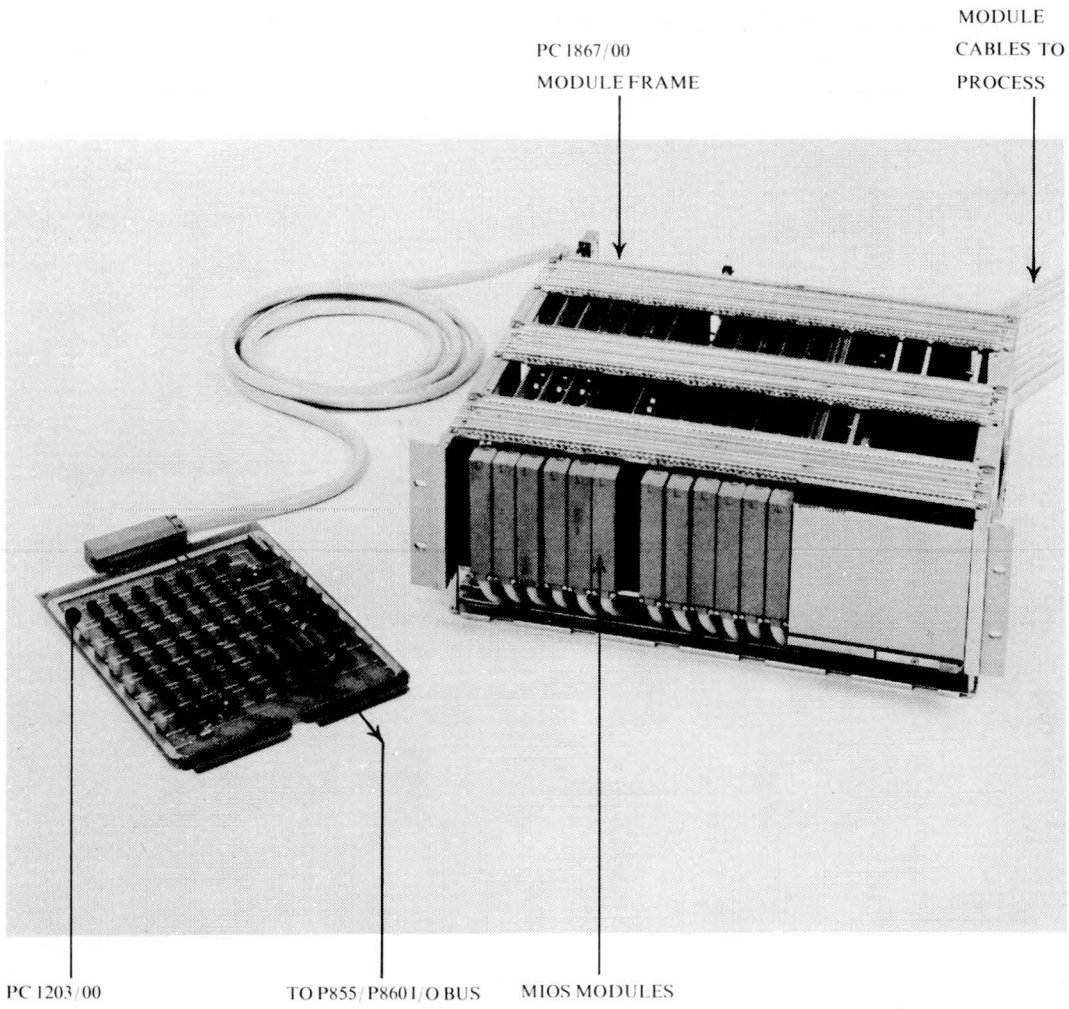


Figure 10 PC1800 MIOS-D BASIC SYSTEM PARTS

General specification of the modules
(More detailed information is available in the form of data sheets for each module).

PC 1841/00,*)
PC 1842/00
and derivated types

DIIM, Digital Input Isolating Module
- Input of 16 bit digital information
- Relay-type isolation with filter input
- Hold, tracking and interrupt modes.

PC 1843/00
and derivated types

DISM, Digital Input Solid-state Module
- Input of 16 bit digital information
- Conditioning with filters and Schmitt trigger circuits
- Hold, tracking and interrupt modes
- Overvoltage protection.

PC 1844/00,
and derivated types

DICM, Digital Input Counter Module
- Two 7-bit counters, counter content parallel into computer
- Interrupt possiblibity on overflow
- Counters can be chained to 14-bit counter with one overflow interrupt
- Slow and fast inputs
- Slow inputs with filters and Schmitt trigger circuits
- Overvoltage protection.

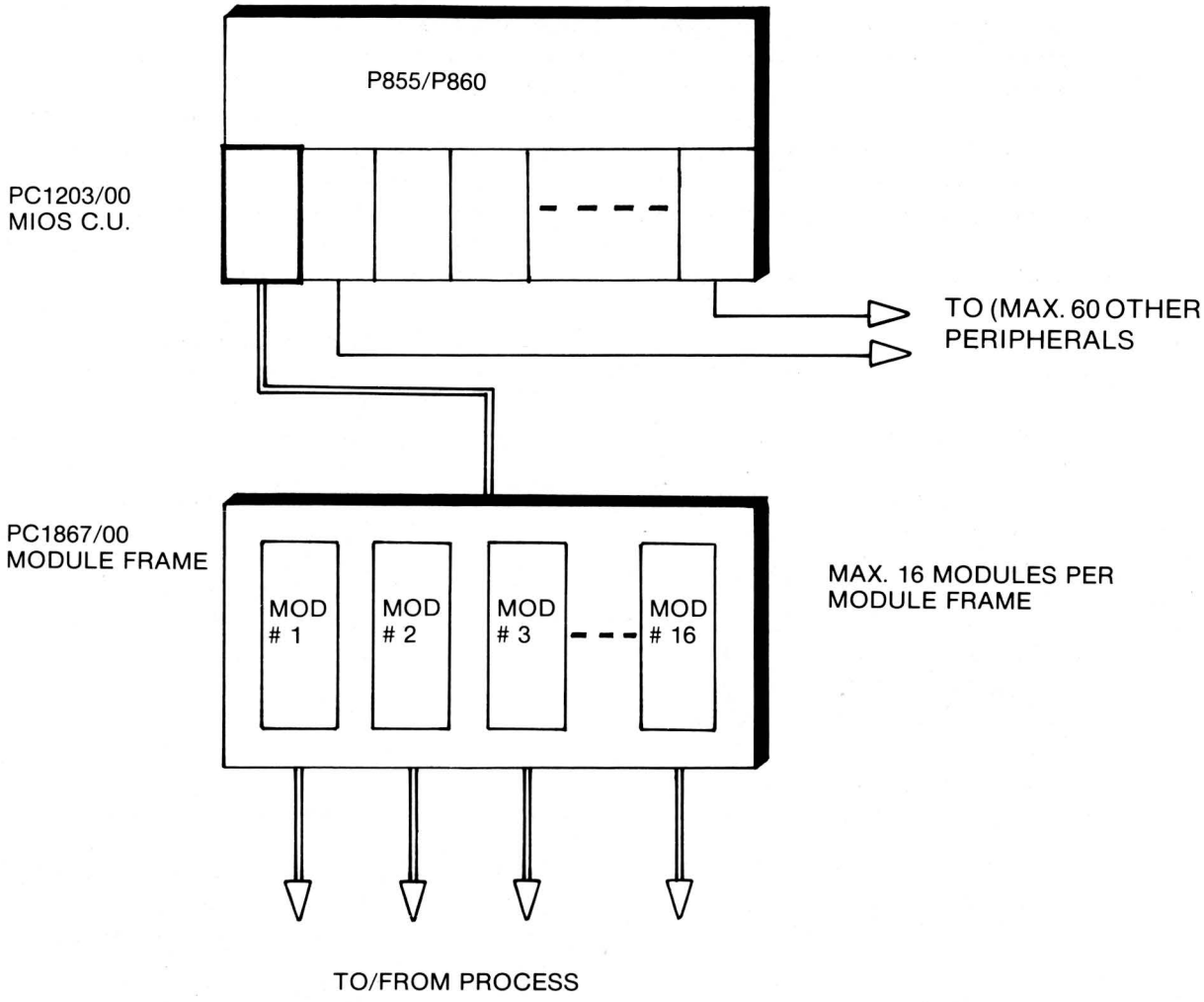
PC 1845/00,
and derivated types

DIPM, Digital Input Priority-interrupt Module
- 16 interrupt inputs, binary address of interrupting input into computer
- Individual memorization of interrupts
- Priority and incremental modes
- Slow or fast inputs
- Slow inputs with filters and Schmitt trigger sircuits
- Overvoltage protection.

PC 1851/00,
and derivated types

DOSM, Digital Output Solid-state Module
- Output of 16-bit digital information
- Max. output voltage: 120 Volts
- Max. output current: 50 mA or 500 mA (each bit)
- Overvoltage protection
- Pulsed or non-pulsed outputs.

*) The mentioned typenumbers will eventually be replaced by new numbers.



PER MODULE:
16 solid state digital inputs (PC1841)
16 galvanically isolated digital inputs (PC1842-PC1843)
16 program interrupt digital inputs (PC1844)
2 seven-bit counter inputs (PC1845)
16 medium level solid state digital outputs (PC1851)
1 pulse output (PC1852)
2 high-speed analog outputs (PC1855)
1 analog output for control purposes (PC1856)

Figure 11 PC1800 MIOS-D system layout for digital inputs, digital outputs and analog outputs

PC 1852/00, and derivated types

POCM, Pulse Output Control Module
- Puls-train output + up/down signal
- Watch-dog type output
- Stepping-motor drive output
- Several optional possibilities
- Controls Philips PCS control unit.

PC 1855/00, and derivated types

AOFM, Analog Output Fast Module
- Two 10-bit DAC (weighing resistors) analog outputs
- Range: 0 to 10V or -5 to +5V or +2 to +10V
- Overall accuracy: 0.4% (0-50°C)
- X T-, XY recorder, oscilloscope or CRT output
- One of the two buffers is a counter buffer
- Pen lift signal available
- Short-circuit protection
- Settling time $<30\mu$sec.

PC 1856/00, and derivated types

AOCM, Analog Output Control Module
- One 12 bit DAC (pulse width integration type) analog output

- Complete isolation between PC 1800-D system and analog output possible
- Range: 0 to +10V, +2 to +10V (accuracy 0.2%)
- Pulse-width modulated outpt available
- Short circuit protection
- Settling time: $\leqslant$ 1 sec. (for 100% step).

General properties of other parts of the PC 1800-D system.

PC 1203/00, PC 1800-D control unit
- Card to be insected in the P800 series computer
- Contains all logic for driving up to sixteen modules
- With 3m cable with connectors for connecting the module frame.

PC 1867/00, Module frame
- 16 slots for modules or 13 modules plus appropriate power supply PC 1862
- 19″ rack mountable frame, 4 E high
- Standard possibility for conducting the input and output cables
- Complete and standard wiring
- Cover plate is standard.

PC 1862/00, and derivated types — Power supply
- 6 Volt, 3A; 15 Volt, 0.5A
- Optional: another 15Volt, 0.5A; 24Volt, 0.3A
- 2/6 sub-assembly, 3E high
- The unit can be inserted in the module frame

PC 2230/20 — Power supply
- 6 Volt, 14.4A; 15 Volt, 4A; 15 Volt, 1A
- 19″ rack mountable unit, 3 E high.

Single analog input and/or analog output

PC 1204/00 — M4C control unit, MIOS-4-convertor control unit.
- Card to be inserted in the P800 series computer
- Contains all logic for interrogating one analog input and transferring one analog output
- The following modules can be mounted on the card giving the control unit special properties
  - ADC module (several possibilities)
  - DAC module (several possibilities)
  - Sample and Hold module
  - Amplifier module
  - Power module (+ 5 Volt to ± 15 Volt convertor)
- The analog-input can be connected to the multiplex mode of the computer.

Analog-input scanner

This scanner can be built up from a module frame PC 1867/00 and analog input modules.
Another PC 1867/00 in the configuration of fig.10 must be available because of the use of a PC 1851/00 (DOSM) for the address memorization and decoding. Each PC 1851/00 can drive a scanner of 64 channels (8 modules). It is possible to combine the two module frames into one module frame PC 1868/00. Each half of this frame has six slots for modules.

Several properties of the analog input modules

PC 1831/00, AILM, Analog Input Low-level Module
- 8 reed-relay channel switches
- Two signal wires and a guard wire are switched
- Max. speed 200 channels/sec.
- Input filters
- Accuracy: 0.1% (measured at 20 mV).

PC 1833/00, AIHM, Analog Input High-level Module
- 8 reed-relay channel switches
- Two signal wires and a guard wire are switched
- Max. speed 200 channels/sec.
- Input filters
- Accuracy: 0.1% (measured at 2 Volt).

PC 1832/00, AISM, Analog Input Solid-state Module
- 8 solid-state channels
- Two signal wires are sequentially switched
- Switching time: 5 μsec.
- Accuracy: 0.1% (measured at 2V)
- Input filters.

The timing of the AI scanner is determined by the PC 1203/00 control unit. Therefore it is not possible to build-up one scanner with two different speeds. All speeds can be formed in the PC 1203/00 control unit.
All three types of AI modules can be modified to select one of the three different standard analog input bus lines of the module frame.
The common analog-output bus of the AI scanner can be connected either to the PC 1204/00 control unit or to an analog-to-digital convertor. If necessary data amplifiers and a subscanner can be inserted in the system between the AI-scanner and the PC 1204/00 (or other ADC).

General properties of the remaining PC 1800-D items

PC 1811/20, Data amplifier
- Dimensions: width 47mm, height 150mm, depth 348mm
- High dynamic common mode-rejection
- System isolation
- Low noise, low drift, wide bandwidth
- The amplifier is used for amplification of the signals from the A.I. reed-scanner
- Voltage gain steps:
  1, 2, 5, 10, 20, 50, 100, 200, 500, 1000.

PC 1813/00, Data amplifier
- Fast solid state amplifier
- High common mode rejection
- The amplifier is used for amplification of the signals from the A.I. Solid-state scanner.

PC 1810/00, AIDC, Analog Input Data Concentrator
- A symmetric solid-state subscanner
- Range-adaptation post-amplifiers
- Built-in voltage reference source optional
- Programmable choice of one of the three analog input busses and choice of the post-amplifier (standard gain: 1, 2 and 5)

PC 1814/00, Analog-to-Digital Convertor
- 3/6 of 19" sub-assembly, 3 E high
- Successive approximation technique
- Full solid state
- Range: -10 Volt to + 10 Volt
- Accuracy: 0.05% of reading ± 0.01% of full Scale + ½ bit (0 to 50 deg. C.)
- Sign-magnitude representation of negative values
- 14 bit + sign.
- Conversion time: 140 μ sec.
- Display: 1 lamp/bit
- Input current:
  250mA at 10V
  150mA at 0V
  50mA at -10V
- BCD version available.

PC 1869/00, Standard frame
This frame can contain several items as PC 1813/00, PC 1810/00, PC 1862/00 and other items not from the PC 1800-D system

PC 1870, Unit frame-D
This frame can contain up to two amplifiers PC 1811/20 and several items as PC 1813/00, PC 1810/00, PC 1862/00 and other items not from the PC 1800-D system.

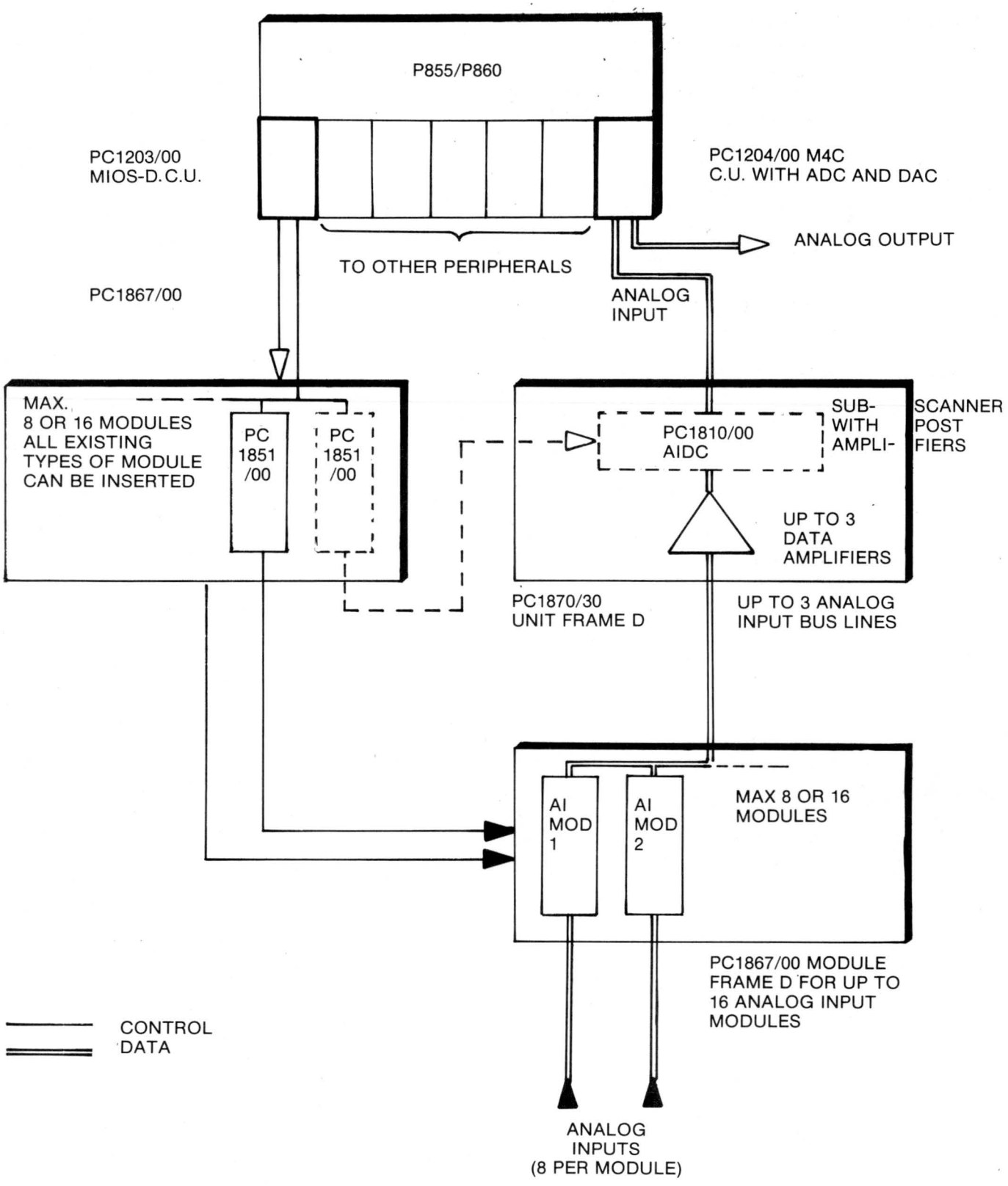


Figure 12 PC1800 MIOS-A example for analog inputs using M4C A/D control unit

Peripheral Connection

This chapter provides information for the user wishing to connect either the P855 or P860 into his own system.

UNIVERSAL I/O BUS

This bus allows external devices to be connected to the CPU so that data transfers can take place in either direction. If the device has a control unit that uses the same logic levels as the CPU it can be plugged straight into an I/O bus socket. Devices without a control unit - or that use different logic levels - can be connected to the bus via either a Philips designed DCU card of a DCU designed by the user.

The bus comprises 28 output lines and 20 input lines that are hard wired between the CPU logic cards and the Elco connector sockets reserved for DCU cards. Pin connections to these sockets have been standardized and all DCU sockets are wired in parallel. Thus any device can be plugged into any socket on the bus. Figure 13 shows the different combinations for connecting a device and the relationship between the CPU, I/O bus, the DCU and the device.

BUS SIGNAL LEVELS

The bus operates into standard TTL logic components that use +5 volts as a logic high level and 0 volts as a logic low level. Devices that use different logic levels can still use the I/O bus, provided that a level adaptation card interfaces the DCU to the bus.

BUS SIGNAL LINES

These lines are shown in Figure 14 and are:

BIN lines:
16 data input lines connecting the data channels of the DCU to the selected register of the CPU.

BOU lines:
16 data output lines connecting the selected register to the data channels of the DCU.

BAD lines:
6 lines used to send the 6-bit address code from the CPU to the DCU sockets.

BOF lines;
3 lines used to send the function code from the CPU to the DCU sockets.

DAV line:
the signal sent on this line - by the CPU - validates data on the BAD and BOF lines.

ARE line:
the signal sent on this line - by the DCU - indicates to the CPU that the address on the BAD lines has been recognized.

ACC line:
the signal sent on this line - by the DCU - indicates to the CPU that the function on the BOF lines has been accepted.

IR line:
this line is activated by the DCU when it is ready to effect either a data transfer - under control of the programmed channel - or to send status information to the CPU.

BR line:
this line is activated by the DCU when it is ready to effect a data transfer under control of the Multiplex option.

EOR line:
the signal sent on this line - by the CPU - indicates to the DCU the end of a multiplex exchange.

MC line:
this line sends a master clear signal - from the CPU to the DCU - when either the CLEAR button, on the control panel, is pushed or when the CPU is first switched ON.

BUS SIGNAL TIMING

The timing of the bus control signals is shown in Figure 15. Timing of the IR, BR, EOR and MC signals is not given because each is dependent upon other factors such as mechanical action, program timing, etc., to generate the signal. However, these outside factors will not affect the bus control signals because their timing has been designed to take care of any eventuality.

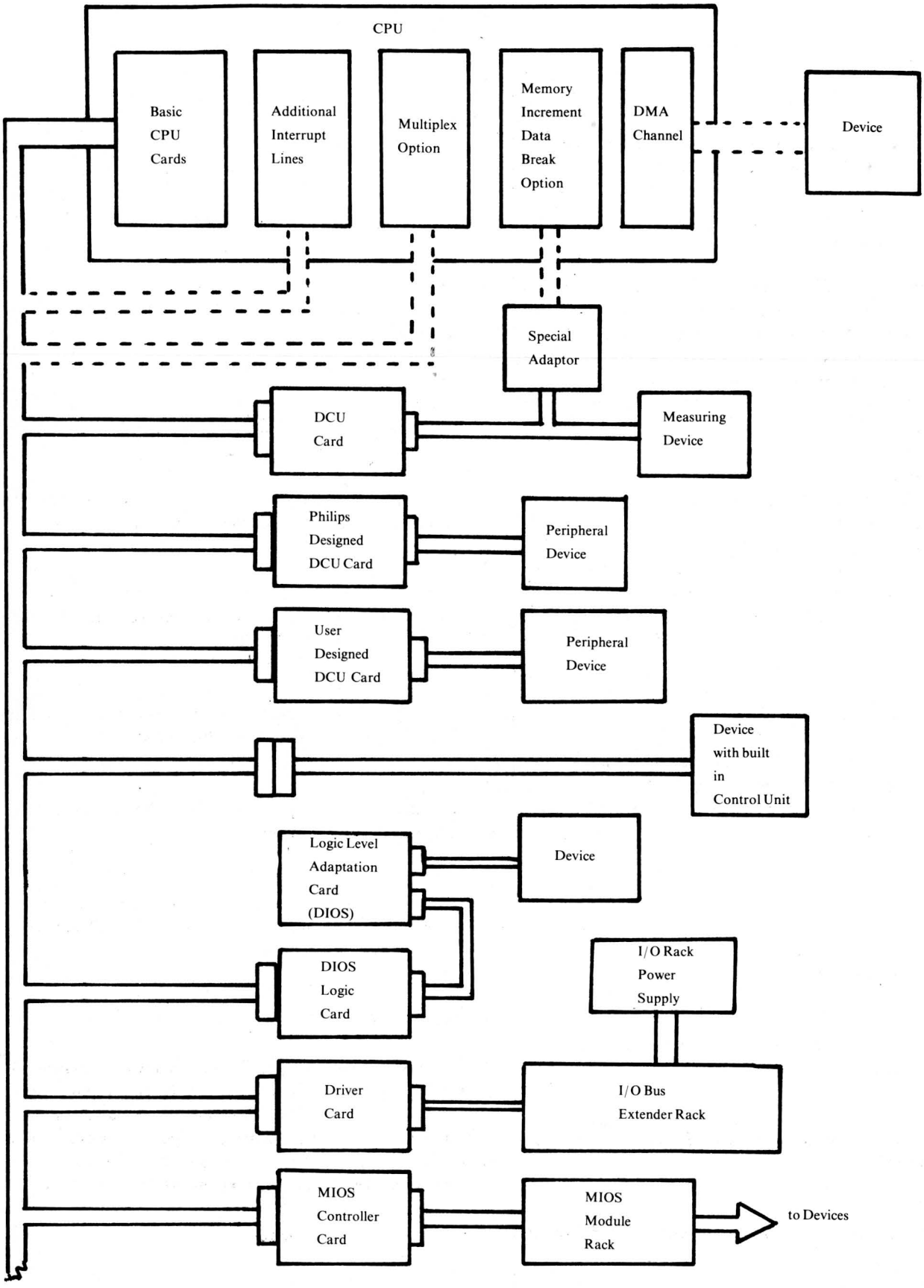


Figure 13 Different combinations to connect a device

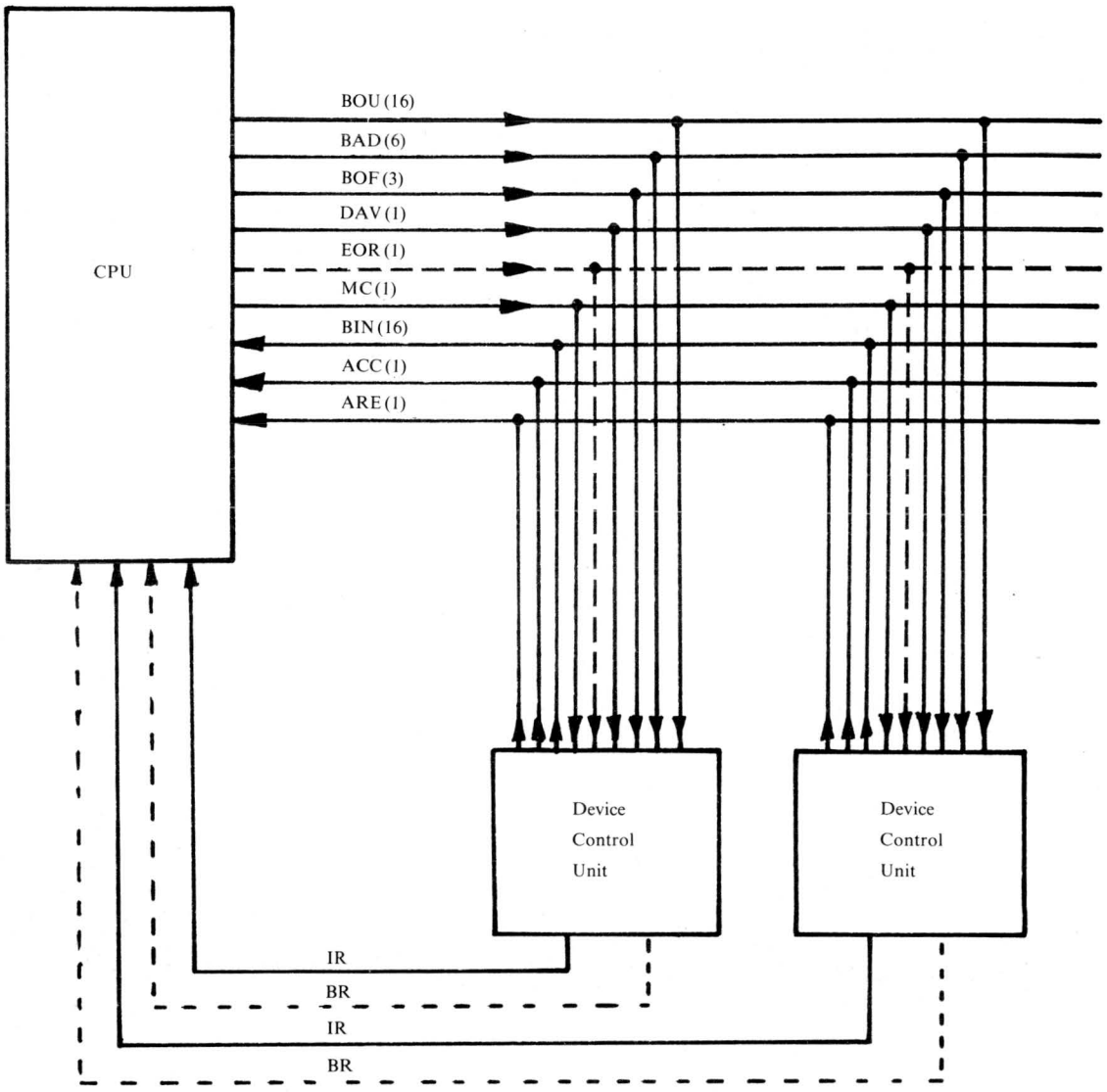


Figure 14 Bus signal lines

Note: The BR lines will only be connected when the Multiplex option is fitted,

BUS CONTROL

Data exchanges via the bus are programmed by I/O instructions and can be controlled either by the programmed channel or the multiplex option. The use of these instructions has already been given in part 2 of this book, therefore this paragraph deals only with their effect on the DCUs via the I/O bus.

Bits 10 to 15 of the instruction word are sent via the BAD lines to each DCU socket connected to the bus. Each BUS is wired to recognize its own address and will respond to it if the DAV signal is also present.

The DAV signal is sent by the CPU shortly after data has been gated on to the bus (see Figure 15). This signal validates the BAD and BOF data on the bus and allows response signals to be sent from the DCU to the CPU.

When the address is recognized, the first response signal will be ARE. This signal tells the CPU that the address has been recognized and that acceptance of the function is being considered.

Bits 4, 8 and 9 of the I/O instruction are sent to the DCU via the BOF lines of the bus. The functions sent on these lines are:

CIO Start - start I/O operation
Stop - stop I/O operation
INR - input data to a register
OTR - output data from a register
TST - test status
SST - sense status

Once the address has been recognized, the DCU checks that the function can be accepted and sends the ACC signal back to the CPU via the bus. Acceptance of the function depends on the state of the DCU and the status of the device.

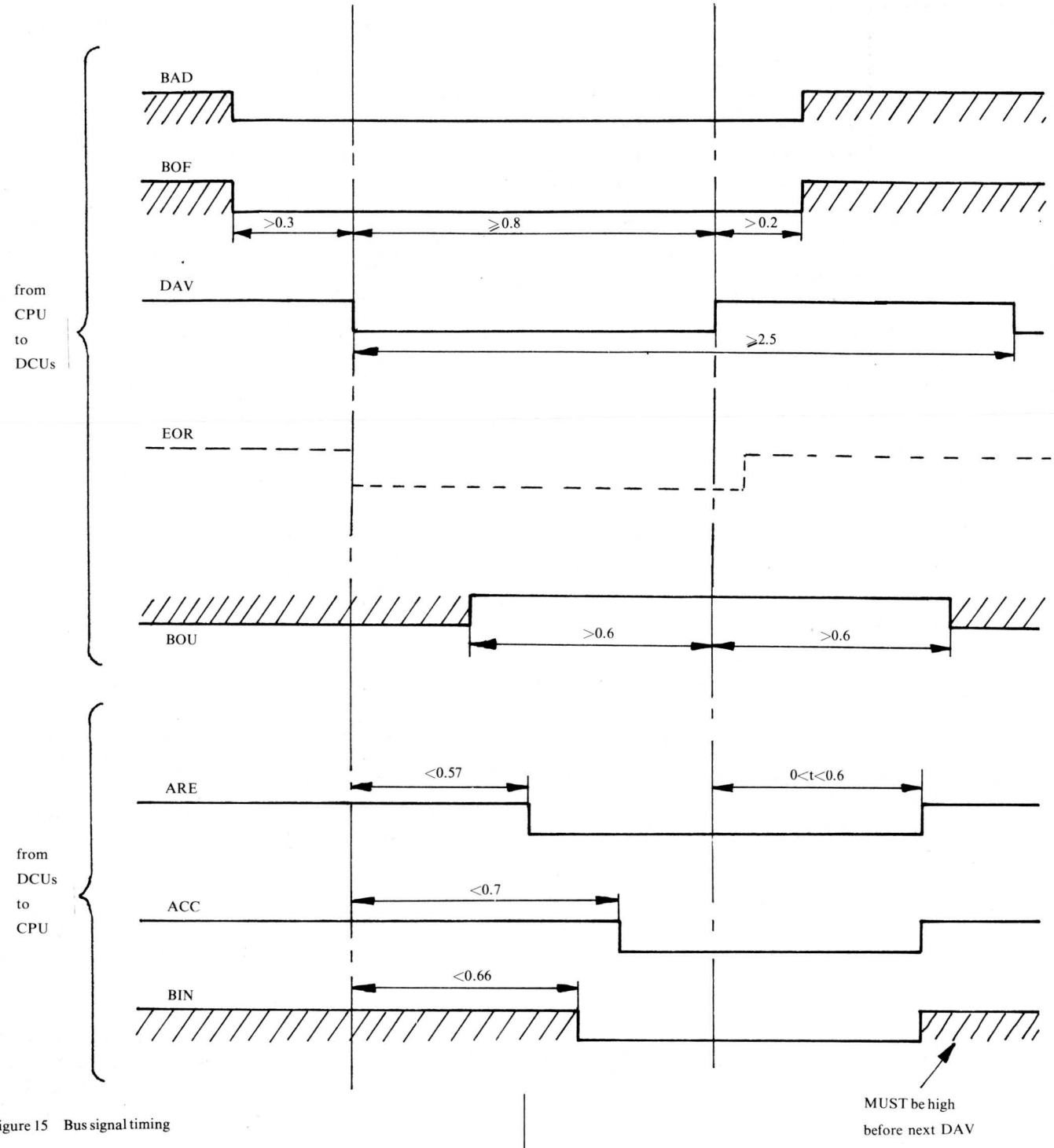


Figure 15 Bus signal timing

Note:

1 All levels are active low except BOU which is active high.

2 This timing does not take into account any delays due to the addition of I/O bus extenders.

DIRECT MEMORY ACCESS (DMA) CHANNEL

The DMA channel is connected to the I/O interface of the CPU (programmed channel) and to the Memory interface.

Up to two control units can be connected to the DMA channel but only one can work at a time.

The exchange is done by words (16 bits).

The block of words to be exchanged with the memory is defined by its length (2s complement) and starting address.

I/O Commands

The control unit has to recognize the following commands:

- CIO activation
- OTR write length
- OTR write address
- CIO deactivation
- INR read length

In a general way, a refused I/O command sets the condition register to '1' and an accepted one resets it to '0'.

During execution of any I/O command a non-recognized device address sets the condition register to '3'.

Invalid commands set the condition register to '1'.

CIO activation

0	1 0 0 0	R1	1 1	DCA
1	4	3	2	6

DCA: DMA channel address

R1: the contents of the register R1 is not significant, this command is used to initialize a DMA exchange.

This command is only accepted when the DMA-channel sequensor is in the inactive state.

A refursed CIO activation sets the condition register to '1', an accepted one resets it to '0' and switches the DMA channel sequensor in 'write length' state.

OTR write length

0	1 0 0 0	R1	0 0	DCA
1	4	3	2	6

DCA: DMA channel address

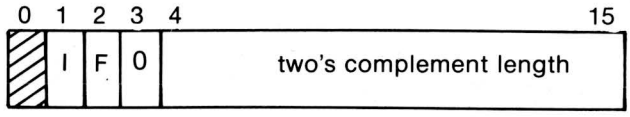
R1: indicates the CPU register from which the length and the function are loaded into the DMA channel length buffer.

This command is used to load the DMA channel length register.

It is only accepted when the DMA channel sequensor is in 'write length' state.

A refursed OTR write length sets the condition register to '1'; an accepted one resets it to '0', loads the contents of R1 into the length buffer and then switches the DMA channel sequensor in 'write address' state.

length format



// : not significant
I : input bit
F : function bit

Bit '3 is not used; it should be reset when the length register is loaded to prevent a change of bit '0' when overflow occurs in the length part.

OTR write address

0	1 0 0 0	R1	0 1	DCA
1	4	3	2	6

DCA: DMA channel address

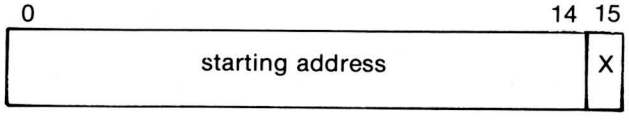
R1: indicates the CPU register from which the address is loaded into the address register.

This OTR command is used to load the DMA channel address register.

It is only accepted when the DMA channel sequensor is in the 'write address' state.

A refused OTR write address sets the condition register to '1'; an accepted one resets it to '0'; loads the contents of R1 into the DMA channel address buffer and then switches the sequensor in 'Execute' state.

address format:



X : not significant

CIO de-activation

0	1 0 0 0	R1	1 0	DCA
1	4	3	2	6

DCA: DMA channel address

R1: the contents of register R1 is not significant.

This command is used to free the DMA channel after the exchange with the control unit has been completed.

This command is always accepted and puts the DMA channel sequensor in 'inactive' state.

INR read length

0	1 0 0 1	R1	0 X	DCA
1	4	3	2	6

DCA: DMA channel address

R1: indicates the register into which data are loaded during this instruction.

X : not significant

This INR command is used to transfer the DMA channel length into the R1 register.

This command is only accepted when the DMA channel sequensor is in the inactive state; a refused INR sets the condition register to '1', an accepted one resets it to '0' and loads R1 with the contents of the DMA channel length register.

DMA channel sequensor

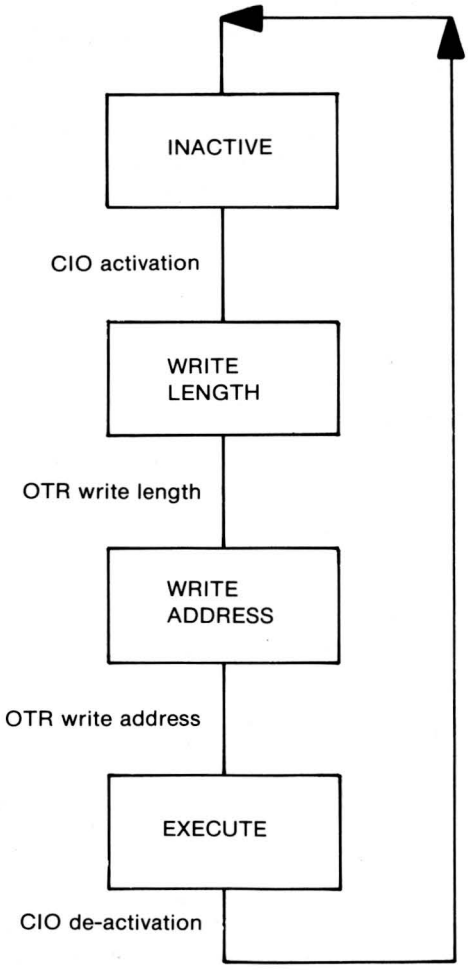


Figure 16 DMA channel sequensor

Data exchange

The DMA channel is first activated by a 'CIO activation'. The block of words to be exchanged between the memory and the peripheral device is then defined as follows:

- An 'OTR write length defines the length of the block and the direction of the exchange: input or output.

- An 'OTR write address' defines the starting address (or current address). The exchange itself is then started by sending a 'CIO start' to the control unit.

Input mode

The control unit sends a request (BR line active) to the DMA channel when data are ready.
The DMA channel answers by loading these data into the data buffer and then activates its DMA request (SSR line); the running program is interrupted at the end of the current memory cycle, the data of the DMA buffer are written into the memory at the current address; control is returned to the program and the length and current address defining the block of data are updated.
When the length becomes zero, an end-of-range signal (EOR line) is sent by the DMA channel to the control unit which activates its interrupt line for status analysis. The DMA channel is then de-activated.

Output mode

As soon as the 'OTR write address has been received the DMA channel activates its request to the CPU (SSR line): the current program memory cycle is ended and the data located at the current address are loaded into the DMA channel data buffer.
Block length and current address are updated and the control is returned to the program.

After receiving the 'CIO start' the control unit activates its request to the DMA channel (BR line) and the data previously read from the memory are sent to the control unit.
The DMA channel performs a new exchange with the memory while the control unit sends the data to the peripheral device.

When the block length becomes zero the end-of-range signal is sent to the control unit (after the last data has been exchanged) and the control unit will activate its program interrupt line for status analysis.
The DMA channel is then de-activated.
If the physical end-of-record does not correspond to the 'end-of-range' signal a throughput error occurs.
After status analysis and DMA channel de-activation, it is possible to perform an INR read length to know the remaining length of the block.

Programming rules

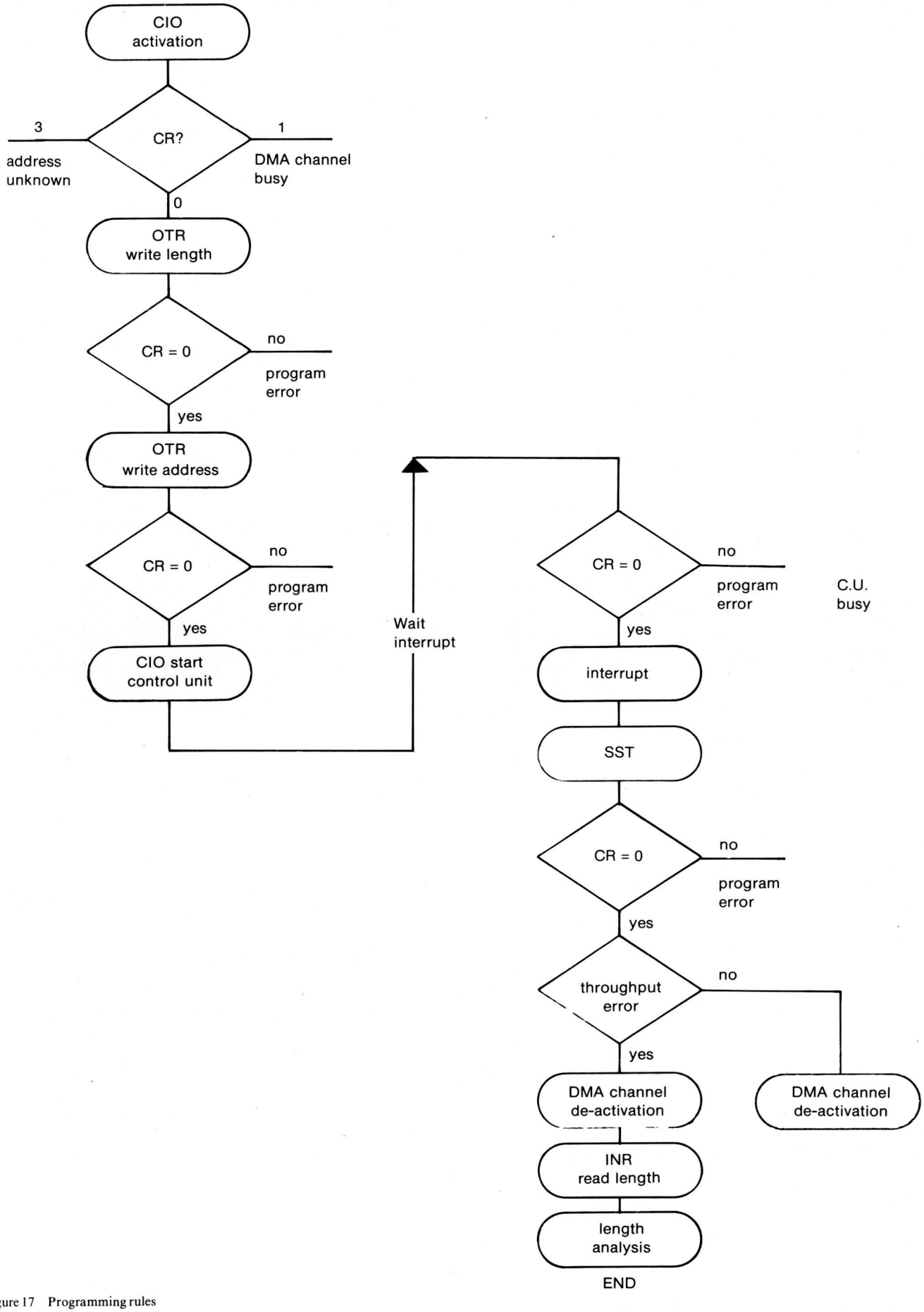


Figure 17 Programming rules

DMA channel interface signals with CPU

- I/O bus standard (see I/O bus)

- 3 CP clock pulses of CPU: AP/,BP/,CP/

- DMA signals

. SAD00,.., SAD14	: 15 memory address lines
. SD00, .., SD15	: 16 memory input lines
. D000, .., D015	: 16 memory output lines
. SSR/	: DMA request
. SMD/	: DMA mode
. SIN	: DMA initiate memory
. SRW/	: DMA read or write
. INT1 }	: Interrupt coming from
. INT2 }	the control units.

DMA/DCU interface

The DMA/DCU interface is compatible with the standard I/O bus interface.

DAVS/
ARES/
BOFS00/,..,BOFS02/
BADS00/,..,BADS05/
ACCS/
BOUS00,..,BOUS15
BINS00/,..,BINS15/
MCLS/

} These signals have the same functions as the corresponding signals of the standard I/O bus

BR/ : Break request the controller asks for an exchange

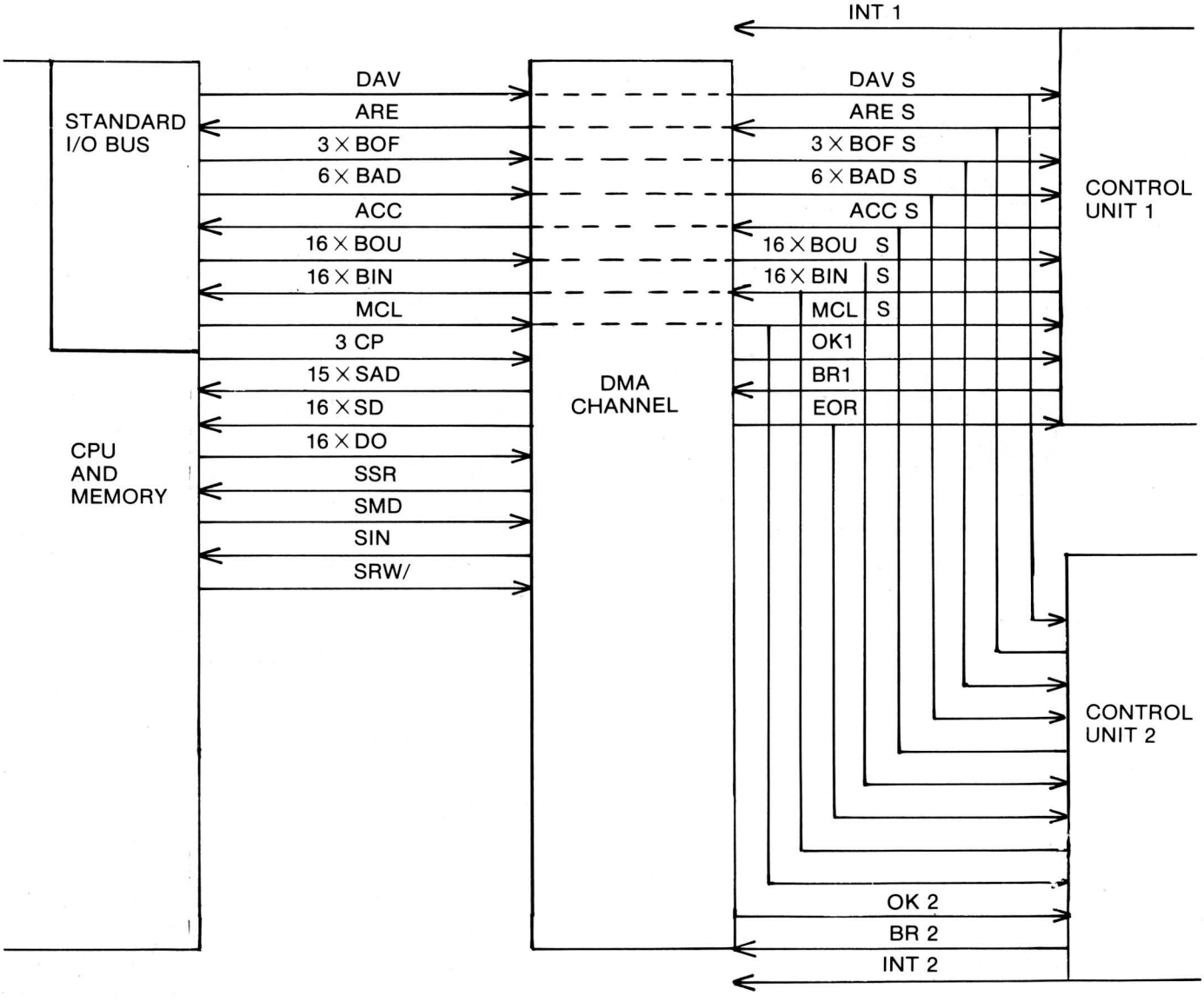


Figure 18 DMA channel interface

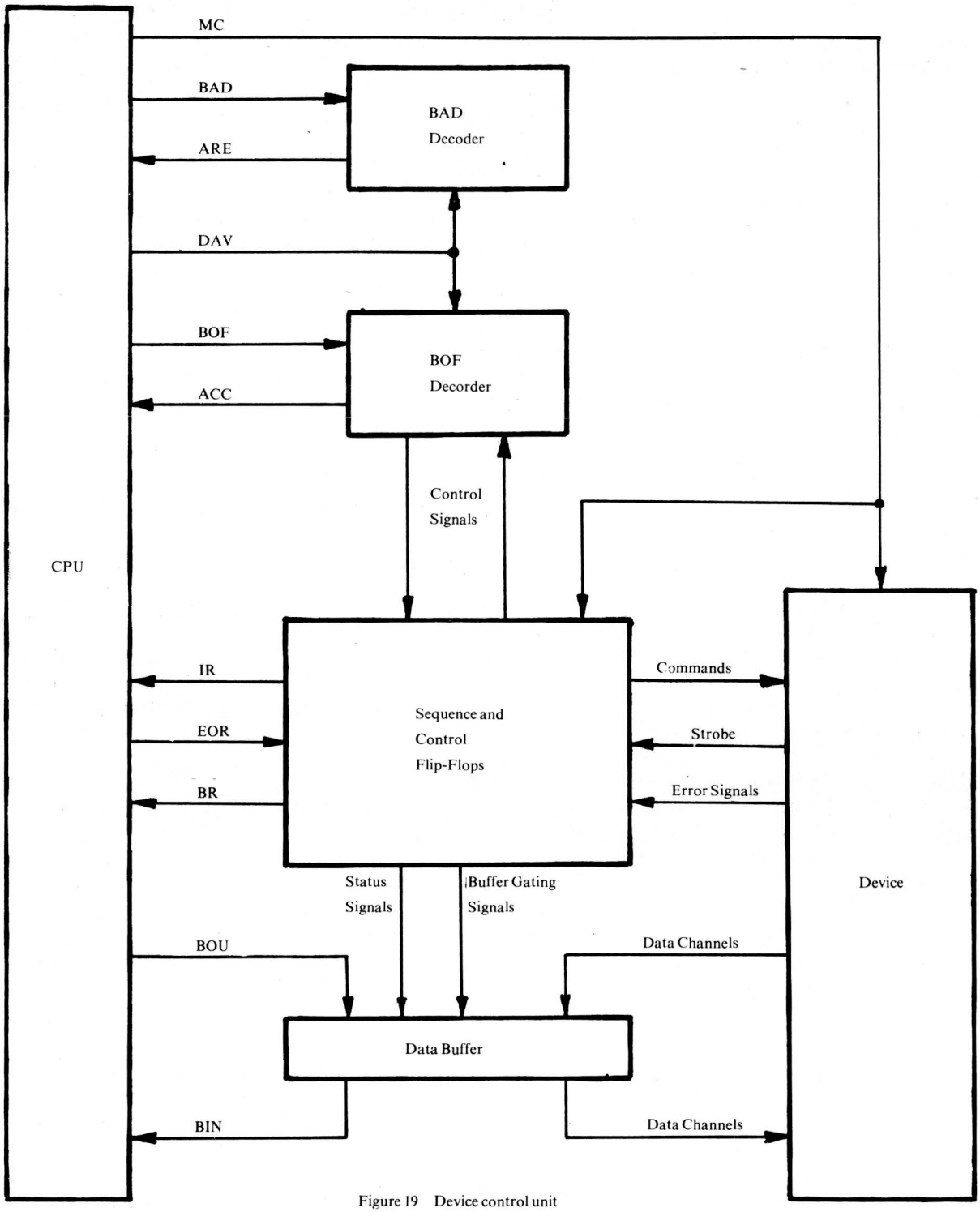


Figure 19 Device control unit

DEVICE CONTROL UNITS

Figure 19 shows a block diagram of a typical DCU. Depending on the complexity of the device, the DCU will be contained on one or more cards. It controls the sequence of the exchanges and synchronizes the timing of the device to the timing of the I/O bus and the CPU.

The sequence of the exchanges is controlled by flip-flops that switch the DCU into one of four states. These four states are:

Inactive:
in this state the DCU is waiting for a CIO start command to the initiate an exchange sequence. On receipt of this command, the DCU will switch to either the Execute or Exchange state depending upon the type of I/O instruction.

Execute;
in this state the DCU causes the device to perform the mechanical action necessary to effect a data transfer; either from the DCU buffer register on to the device medium or from the device to the DCU buffer register.

Exchange:
in this state data is transferred either from the BOU lines into the DCU buffer register or from the DCU buffer register on to the BIN lines.

Wait Status:
the DCU switches to this state when a transfer is ended.

The exact sequence of switching is controlled by the I/O commands. These commands will only be accepted if the DCU is in the appropriate state and a typical sequence would be:

CIO Start:
this command will only be accepted in the Inactive state. If the command is received in any other state, it will send the device busy signal back to the CPU. This signal is a one bit on BIN line 15. When the command is accepted, the DCU will switch into either the Execute or Exchange state depending on whether the transfer is input or output.

INR:
when this type of transfer is accepted, the DCU switches from the Inactive state to the Execute state. In this state the only commands that are accepted are CIO Stop and TST. Once the data has been transferred from the device into the buffer register of the DCU, the DCU switches to the Exchange state and sends an interrupt signal to the CPU via either the IR line - if the transfer is controlled by the programmed channel - or the BR line, if the transfer is controlled by the multiplex option.

When the CPU receives the interrupt, it sends the INR command to the DCU. The INR command will only be accepted in the Exchange state and once the data has been gated on to the BIN lines the DCU will switch back to the Execute state ready to transfer another character.

This switching from Execute to Exchange will continue until the transfer is ended by either a CIO Stop command - for a programmed channel transfer - or by the EOR signal for a multiplex transfer. When either of these is received, the DCU switches to the Wait Status and sends an interrupt to the CPU.

OTR:
when this type of transfer is accepted, the DCU switches from the Inactive state to the Exchange state and sends an interrupt to the CPU. The CPU then sends the OTR command - which will only be accepted in the Exchange state - and once the data has been gated from the BOU lines into the buffer register of DCU, the DCU switches to the Execute state. The Execute state records the data - in the DCU buffer register - in the device medium and switches back to the Exchange state.

This inter-state switching will continue until the transfer is ended by either a CIO Stop command or the EOR signal from the multiplex option. The DCU will then switch to the Wait Status state and send an interrupt to the CPU.

SST:
this command must be sent at the end of every transfer. It will only be accepted in the Wait Status state and is used to sense the status of the device - i.e. if the transfer has been error free - and to cause the DCU to switch to the Inactive state.

TST:
this command can be sent at any time during a transfer and is always accepted in any state of the DCU. It is used to test the status of DCU or device.

CIO Stop:
this command can be sent at any time during a transfer and is always accepted. It causes the DCU to switch from its present state into the Wait Status state. There are two ways to cause this switch into the Wait Status state; one is on receipt of an EOR signal from the multiplex option, the other is when an error condition is detected by the DCU.

MEMORY INCREMENT DATA BREAK

This option also uses the I/O bus to send or receive control signals from the external equipment. Other signal lines are needed to complete the interface and these are supplied by the external adaptor. Figure 20 shows these lines connected between the measuring equipment and the central processor. These lines have the following functions:

CNL:
Channel address lines, this address is given in binary and the number of lines will depend on individual requirements. Each channel line addresses a specific pair of dedicated memory locations which are used for increment operations.

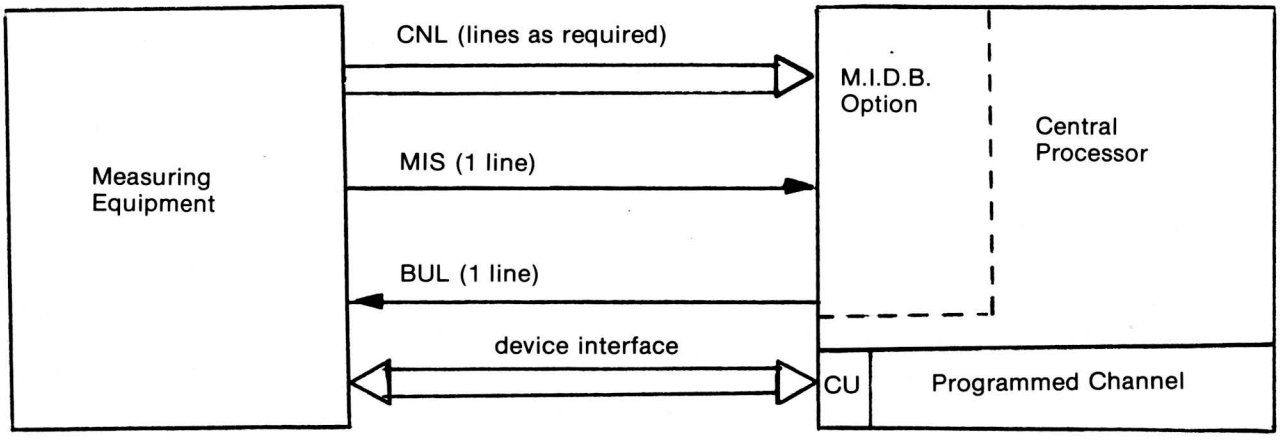


Figure 20 M.I.D.B. Interface lines

- MIS:
Memory Increment Signal line; requests the processor to stop and perform an increment operation.
- BUL:
Busy Line; is a response signal from the processor to the measuring equipment to stop counting new pulses until the present increment operation has been completed.

Timing
The following list gives the minimum, maximum normal and maximum with overflow times of an increment operation:

Minimum: 5 microseconds
Maximum (normal): 12 microseconds
Maximum (overflow): 16 microseconds

Figure 21 shows the timing of the M.I.D.B. Interface signals.

CNL must be present at the same time as MIS and remain significant until BUL is deactivated.

BUL becomes active as soon as MIS signal line is active and remains active until increment is completed.

MECHANICAL ASPECTS

The DCU logic and special circuits are mounted on one or more standard circuit cards shown in Figure 22. Blank cards - of the same size - can be supplied for users who wish to design their own DCU's. These cards have hole patterns printed on them that will allow IC modules to be mounted on them. There is no other printed circuitry except a short lead out from each connector pin on the card.

Connectors 1A and 2A plug into the I/O bus sockets, connector 5A is used to connect the device to the DCU.

BUS PIN CONNECTIONS

The pin connection to sockets 1A and 2A have been standardized so that any DCU card can be plugged into any I/O bus sockets. These are:

1A01	+5 volts	2A01	BOF02/
1A02	reserved	2A02	EOR
1A03	BIN00/	2A03	BOF01/
1A04	BIN01/	2A04	BOF00/
1A05	BIN02/	2A05	BOU14
1A06	BIN03/	2A06	BOU13
1A07	BIN04/	2A07	BOU12
1A08	BIN05/	2A08	BOU09
1A09	BIN06/	2A09	BOU10
1A10	BIN07/	2A10	BOU11
1A11	ground	2A11	BOU15
1A12	reserved	2A12	MC/
1A13	ACC/	2A13	BAD03/
1A14	ARE/	2A14	DAV/
1A15	BRIR/	2A15	BAD04/
1A16	BR1/	2A16	BAD05/
1A17	BR2/	2A17	BOU08

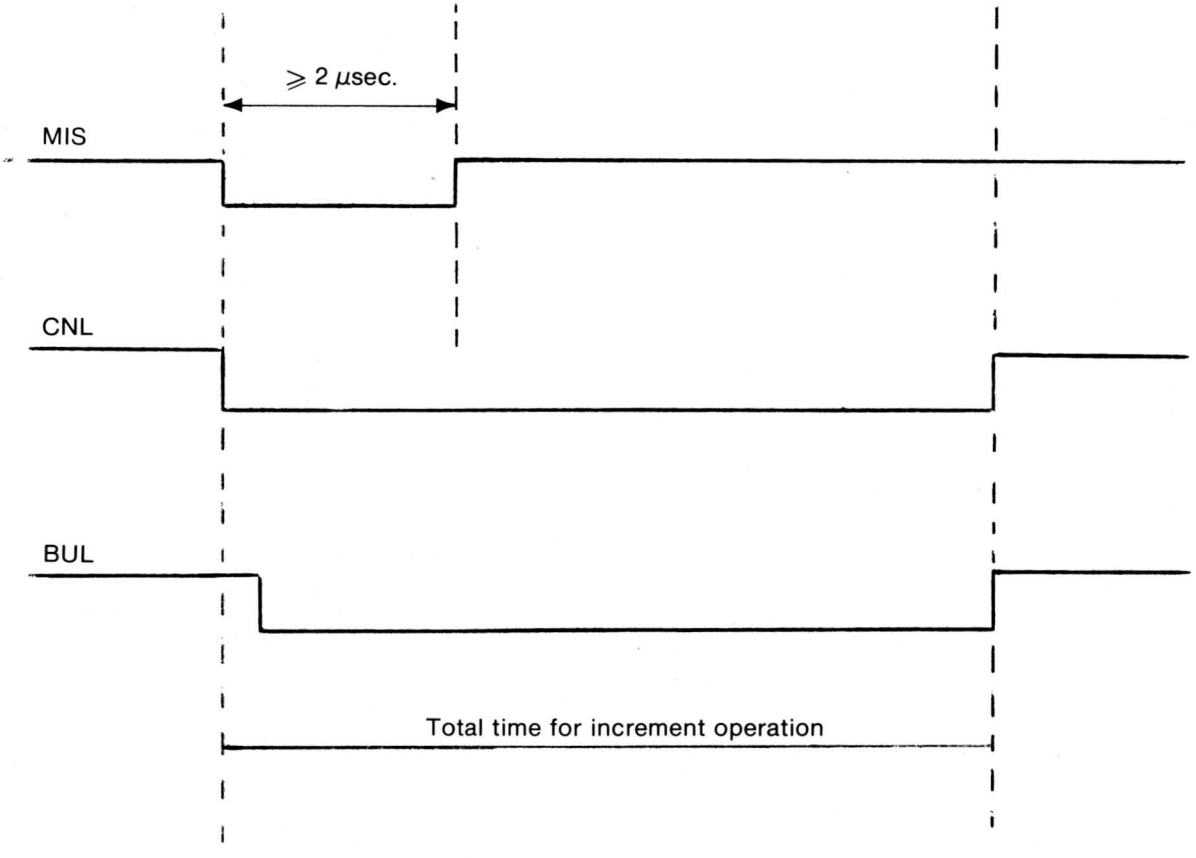
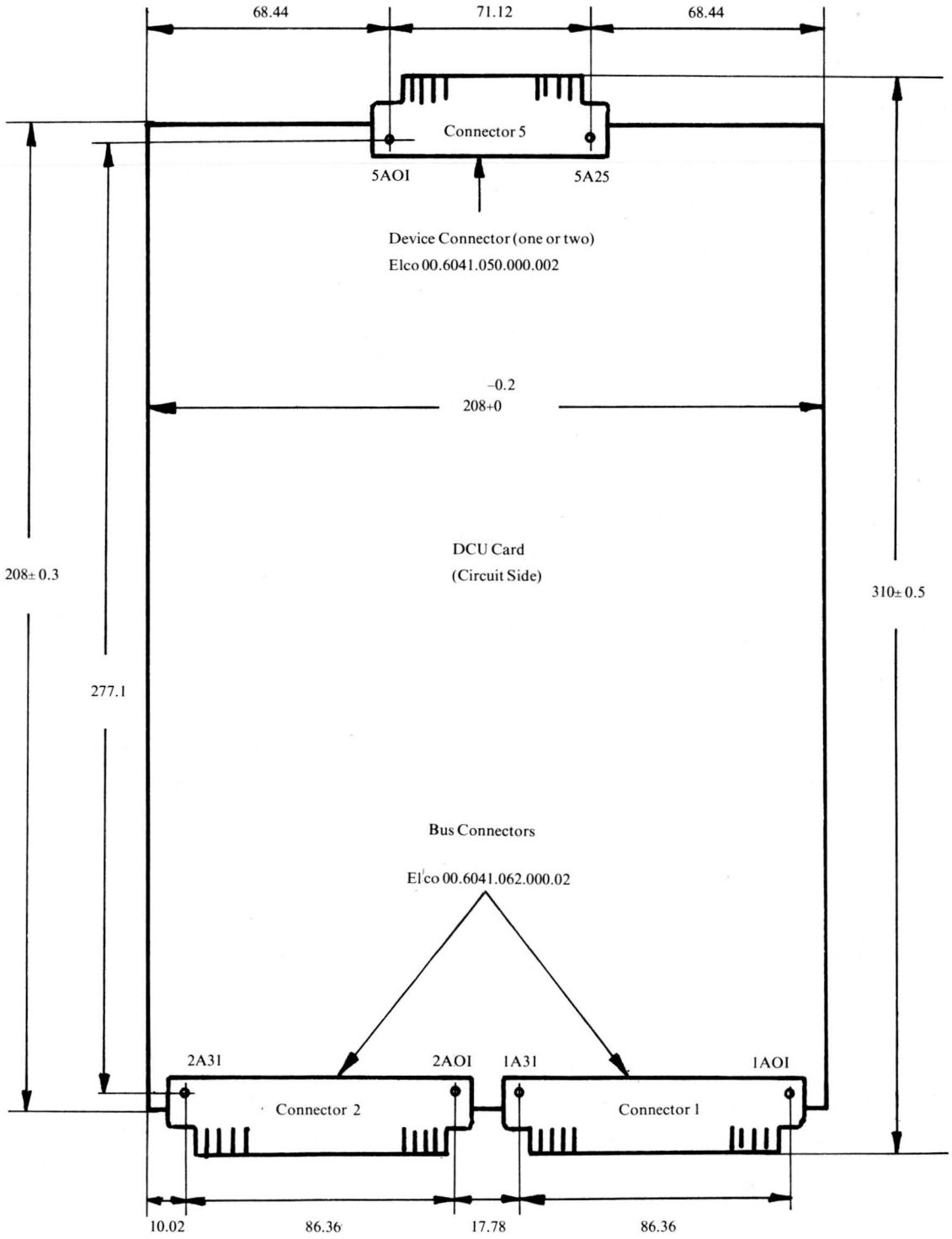


Figure 21 M.I.D.B. Timing diagram

1A18	BR3/	2A18	BOU07	1A26	BIN13/	2A26	BOU02
1A19	BR4/	2A19	BOU06	1A27	BIN15/	2A27	BOU01
1A20	reserved	2A20	BOU05	1A28	BIN09/	2A28	BOU00
1A21	BIN08/	2A21	BOU04	1A29	BIN10/	2A29	+24 volts
1A22	BIN14/	2A22	BAD00/	1A30	BIN11/	2A30	−5 volts
1A23	+6 volts	2A23	BOU03	1A31	BIN12/	2A31	+5 volts
1A24	−6 volts	2A24	BAD01/				
1A25	−12 volts	2A25	BAD02/	1B31	ground	2B31	ground



All dimensions given in millimetres

Figure 22. Control Unit Circuit Board

DCU TO DEVICE CONNECTIONS

The connections between the DCU and the device are made by twisted pairs of wires. One of the wires is connected to ground - both at the Elco connector socket that plugs on to the 5A end of the DCU card and at the device plug - the other is connected to the signal level. The signal lines will be different for each device type, but in general each device type will need lines for:

the data channel
the control signals (start/stop)
the strobe pulse
the operability signals (end-of-tape)

DIOS CONNECTION

The DIOS cards - both logic and electronic - plug straight into the I/O bus sockets. When the logic cards are used on their own - i.e. with a device that has logic levels similar to those of the CPU - the device is connected directly on to the 5A end of the card. For devices that have logic levels different to those of the CPU, the device is connected to the electronic card which in turn is connected to the logic card (see Figure 19).

MIOS CONNECTION

The MIOS controller card - that connects the module frame sockets to the CPU - is plugged straight into an I/O bus socket. The devices are connected directly on to the MIOS modules (see Figure 19).

I/O BUS EXTENDER

The I/O bus extender consists of 2 cards mounted in the rack that can obtain up to 21 additional DCU cards. The 2nd card is plugged into the I/O bus in the basic cabinet. The maximum length of the cable between these two cards is 15 metres. When this extension rack is fitted, an extra power supply must be used to supply working voltages to the cards mounted in the rack. Typical connection is shown in Figure 19.

Full details can be had from: **NV Philips-Electrologica**
Industry Group Small Computers
Postbox 245, Apeldoorn
The Netherlands
Tel. 05760 - 30123; telex 49142

or from any of the following organizations:

EUROPE

The Netherlands
Philips-Electrologica Nederland N.V.
De Horst 4 (Postbus 2408)
Den Haag - Mariahoeve,
Tel. 070 - 814571

Sweden
Svenska AB Philips Data Systems
Fack 183 03 Täby 03,
Stockholm,
Tel. 756 0020

Denmark
Philips Electrologica A/S
Prags Boulevard 80
2300 København S,
Tel. 2222

Norway
Norsk Aktieselskap Philips
Avd. Data Systems
Haakon VII'S GT. 5
Oslo 1
Tel. 332470

Finland
OY Philips AB
Kaivokatu 8
Helsinki 10
Tel. 10915

Belgium
NV Philips-Electrologica
Anspachlaan 1
1000 Brussel,
Tel. 193900

France
Philips M.E.P.
Division Ordinateurs
5, Square Max Hymans
75, Paris 15e,
Tel. 734 7759

Western Germany
Philips Electrologica GmbH
Geschäftsbereich Computer-Systeme
Liesegangstrasse 15
4 Düsseldorf,
Tel. 360361

Italy
Philips S.P.A.
Divisione Sistemi
Viale Fulvio Testi, 327
20162 Milano,
Tel. 6420951

Switzerland
Philips AG
Edenstrasse 20
8027 Zürich
Tel. 01 - 442211

England
M.E.L. Equipment Company Ltd.
Manor Royal,
Crawley,
Sussex,
Tel. 0293 28787

NORTH AMERICA

U.S.A.
North American Philips Corp.
Dept. 007
100 East 42nd Street,
New York N.Y. 10017
Tel. 212 697 3600

FAR EAST

Japan
Philips Industrial Development
and Consultant Co. Ltd.
Kokusai Building 7th floor,
1-1, 3-Chome Marunouchi, Chiyoda-Ku,
Tokyo 100
Tel. 213 6752 9