1900 ICT series

1900 ICT series

1900 ICT series

Central Processors

4095

1900 ICT series

I·C·T

1900 series

Central Processors

From 9th July 1968 for any reference in printed literature to International Computers and Tabulators Limited or I.C.T.

Please read

International Computers Limited or ICL

Registered Office:
ICL House, Putney, London SW15
Telephone 01-788 7272

ICL

Technical Publication 4095

First Edition April 1968

Issued by Technical Publications Service
International Computers and Tabulators Limited
Head Office: I.C.T. House, Putney, London S.W.15
and printed in Great Britain by
Printing and Stationery Department
Letchworth, Hertfordshire

Preface

This manual describes the 1900 Series central processors and includes both information that is of importance to 1900 users and information that is for the sake of interest only. Information in the latter category is clearly marked as such, and no inference should be drawn from it as regards any central processor.

Chapter 1 of the manual provides a brief introduction to the 1900 Series central processor and mentions salient points of the 1900 philosophy. The succeeding chapters presuppose a level of familiarity with computers such as is to be expected with data processing personnel. It has been assumed that most readers will either peruse the manual to seek a general information on the 1900 Series or else will require details of specific central processors. In the former case the reader may progress through the manual chapter by chapter. In the latter case, the reader is recommended to turn first to one of the Chapters 10 to 12 in which will be found a description of the relevant central processor. If any facilities mentioned in this description are not familiar to the reader he should refer to Chapter 13. This Chapter will contain either a description of the facility or a reference to the page in the manual where the facility is described.

This manual replaces Volume 1 of the System Manual and part of the 1901 Processor And Peripherals and differs from them substantially. In particular it contains information on central processors recently introduced into the 1900 Series and on new features of the series such as the priority interrupt feature, paging, and extended addressing facilities. This information has not been published previously. Existing descriptions of other aspects of the 1900 Series have been expanded to take account of recent processor developments.

Contents

Preface iii

Chapter 1 Introduction to the 1900 Series central processor 1
CENTRAL PROCESSOR CONSTITUENTS 1
FEATURES OF THE 1900 SERIES CENTRAL PROCESSOR 1
Compatibility 1
Storage 1
Standard interface 1
Executive 1
The console typewriter 2
Floating-point arithmetic 2
Programming languages 2
Dual processor capabilities 2
Interprocessor buffer 4
Multiprogramming 4

Chapter 2 The core store 5
STORE SIZE 5
STORE SPEED 5
STORE ACCESS 5
Store Multi-access control 5
PRINCIPLES OF READING AND WRITING IN STORE 7
PARITY CHECKING 9
STORE INTERLEAVING 9
Two-way interleaving 9
Four-way interleaving 9
Efficiency increase 9
Store failure 9
ALLOCATION OF STORAGE 9
LAYOUT IN STORE OF OBJECT PROGRAM 10
Accumulators – Words 0 to 7 11
Reserved store and entry parameter store – Words 8 to 29 11
Switch word – Word 30 11
Reserved words – Words 31 to 45 11
Lower data store – Words 46 up to 4,095 12
Program store 12
BRANCH MODES 12
Upper data store 12
ADDRESSING MODES 12

Additional words reserved when subprogramming is employed 12
AVAILABILITY OF CORE STORE 13

Chapter 3 The arithmetic and control units 15
GENERAL 15
CLOCK PULSES 15
THE CONTROL UNIT 15
THE ARITHMETIC UNIT 15
LAYOUT OF THE CONTROL AND ARITHMETIC UNITS 16
The mill 16
Register A 16
Register B 16
Register P 16
Datum and limit registers 16
Register N 16
Registers F, X, M 16
HOW INSTRUCTIONS ARE PERFORMED 17
First Instruction 17
Second instruction 17
Third instruction 17

Chapter 4 Representation of data 19
WORD FORMAT 19
CHARACTER FORM 19
PURE BINARY FORM 19
Counter modifiers 19
WORD COUNTER MODIFIER 20
CHARACTER COUNTER MODIFIER 20
COUNTER MODIFIER WORDS IN 22AM or EBM 20
Fixed-point numbers 20
SINGLE-LENGTH INTEGER 20
MULTI-LENGTH INTEGER 20
SINGLE-LENGTH FRACTION 21
MULTI-LENGTH FRACTION 21
MIXED NUMBER 21
Floating-point numbers 21
SINGLE-LENGTH 22
DOUBLE-LENGTH 22
QUADRUPLE-LENGTH (EXTENDED PRECISION) 22
OVERFLOW AND CARRY 23
Carry 23
Overflow 23
FIXED-POINT 23
FLOATING-POINT 23
FIELD DESCRIPTIONS 23
Defined 23
Indeterminate 23
Ignored 23
Reserved 24

Chapter 5 Format of instructions 25
GENERAL FORMAT 25
Normal instructions 25
Branch instructions 25
Shift instructions 26
Floating-point instructions 26
ADDRESSING AND BRANCH MODES 26
Concepts 26
Setting and switching of modes 26
MODE SETTING ON PROGRAM LOADING 26
MODE SWITCHING DURING PROGRAM EXECUTION 27
MULTI-MEMBER PROGRAMS 27
22AM Considerations 27
MODIFICATION 27
COUNT INSTRUCTIONS 27
SUBROUTINE ENTRY AND EXIT 27
CONSOLE DISPLAYS 27
EBM Considerations 27
THE 023 (OBEY) AND BRANCH INSTRUCTIONS 28
SUBROUTINE LINKS 28
SUMMARY OF FUNCTION CODES 28
Definition of notation 28
DERIVATION OF DN AND BN 29
Definition of DN 29
Definition of BN 30
Definition of functions 30
GROUP 00 30
GROUP 01 31
GROUP 02 31
GROUP 03 32
GROUP 04 33
GROUP 05 33
GROUP 06 34
GROUP 07 34
GROUP 10 36
GROUP 11 36
GROUP 12 38
GROUP 13 39
GROUP 15 40
GROUP 16 40
AVAILABILITY OF INSTRUCTIONS WITH CENTRAL PROCESSORS 43

Chapter 6 Peripheral transfers 45
THE NATURE OF INFORMATION TRANSFERRED 45
STANDARD INTERFACE 45
The purpose of standard interface 45
Standard interface lines 45
THE Do AND Di LINES 45
THE C LINE 47

THE R LINE AND T LINE 47
THE B LINE 47
THE L LINE 47
THE A LINE 47
HESITATIONS 47
Timesharing of the central processor for program instructions and hesitations 47
Operations performed during a hesitation 49
Single-character hesitation 49
Burst mode hesitation 49
TRANSFER FROM THE CORE STORE TO A PERIPHERAL 50
TRANSFER FROM A PERIPHERAL TO THE CORE STORE 50
High-speed mode 50
Crisis times 50
METHODS OF DECREASING HESITATION TIME 51
Peripheral control connector 51
TRANSFER FROM A PERIPHERAL TO THE CORE STORE 51
TRANSFER FROM THE CORE STORE TO A PERIPHERAL 51
Store access control 52
Peripheral autonomous control 52
Store access manager 53
Peripheral processing unit 53
STANDARD INTERFACE SWITCHING UNIT 53
Types of module 54
MODULE TYPE 7204/1 (Y TYPE) 54
MODULE TYPE 7204/2 (N TYPE) 55
MODULE TYPE 7204/3 (X TYPE) 55

Chapter 7 Executive 57
THE NATURE OF EXECUTIVE 57
THE PURPOSES AND FUNCTIONS OF EXECUTIVE 57
THE COMPOSITION OF EXECUTIVE 57
TYPES OF EXECUTIVE 58
ENTRY TO EXECUTIVE 59
MULTIPROGRAMMING 59
Program protection 59
Program priorities 60
Program deletion 60
Multiprogramming under GEORGE 3 or GEORGE 4 control 60
DUALPROGRAMMING 60
Program protection 60
Program deletion 61
SUBPROGRAMMING 61
The purposes of subprogramming 61
Members of a program 61

Information associated with each member 62
MEMORY INDICATORS 62
Information associated with the Priority Member 62
Further notes on the Priority Member 63
PRIORITY 63
TIME-OUT FEATURE 63
ORDER CODE RESTRICTIONS 63
Further notes on other members 63
PRIORITIES 63
ADDRESS AND BRANCH MODES 63
Loading and dumping 63
LOADING 63
DUMPING 64
Reference to common storage area 64
PROGRAMS WITHOUT A PRIORITY MEMBER 64
PROGRAMS WITH A PRIORITY MEMBER 64
Subprogramming control instructions 65
THE 162 (SUSMA) INSTRUCTION 65
Restrictions 65
THE 163 (AUTO) INSTRUCTION 65
Initial activation 65
Subsequent re-activations 65
THE 164 (SUSAR OR SUSIN) INSTRUCTION 65
The 164, X = 1 (SUSAR) variant 66
The 164, X = 2 (SUSIN) variant 66
The 164, X = 3 (SUSIN) variant 66
The 164, X = 4 (SUSIN) variant 66
States of members 66
STATE TRANSITION TABLES 66
Examples 68
EXAMPLE 1: WHY THERE MUST BE M INDICATORS 68
EXAMPLE 2: A 163 INSTRUCTION ISSUED WHEN M IS SET 69
EXAMPLE 3: COMPLETE PARALYSIS 69
EXAMPLE 4: SPURIOUS RE-ACTIVATION 69
EXAMPLE 5: THE USE OF THE 162 INSTRUCTION 70
PERIPHERAL TRANSFERS 74
BINARY PROGRAMS 81
Binary program formats 81
Block types 81
Block type 1: Request block (request slip) 81
WORD 0 82
WORD 1 82
WORD 2 82
Bits 0 and 1 82
Bits 2 to 5 82
Bits 6 to 20 82
Bits 21 to 23 82
WORD 3 82
Bits 0 to 8 83

Bits 9 to 16 83
Bits 17 to 23 83
WORD 5 83
WORD 6 83
WORD 7 83
Bits 0 to 11 83
Bits 12 to 17 83
Bits 18 to 23 83
WORD 8 83
WORD 9 83
WORD 10 83
WORD 11 83
WORD 12 83
WORD 13 83
WORD 14 AND 15 84
Block type 0: data block 84
WORD 0 84
WORD 1 84
WORD 2 84
WORDS 3 TO 17 84
NEXT WORD 84
WORD 19 84
Block type 5: data block 84
WORD 0 84
WORD 1 84
WORD 2 84
WORD 3 84
WORD 4 85
Block type 2: entry block 85
WORD 0 85
WORD 1 85
WORD 2 85
WORD 3 85
Block type 3: entry block 85
WORD 0 85
WORD 1 85
WORD 2 85
WORD 3 85
Block type 4: entry block 85
WORD 0 85
WORD 1 85
WORD 2 86
WORD 3 86
Block type 6: supplementary request block 86
WORD 0 86
WORD 1 86
Bit 21 86
Bit 23 86
WORDS 2 to 6 86
WORD 7 86

Block type 62: pre-request block sentinel 86
WORD 0 86
WORD 1 86
WORD 2 86
WORD 3 86
Block counts and sequence numbers 87
Layout of binary program on various media 87
PAPER TAPE 87
80-COLUMN CARDS 87
MAGNETIC TAPE 87
DIRECT ACCESS 88
Executive treatment of request and supplementary request blocks 88
EX1H, E1HS 88
EX2L
EX1T, E1TS, EX1V, E1DS, E1MS, EX2S, E3TS, E3TS, E3TE, EX2V, E3DS, E3DE, EX2M, E4BM, E3TM, E3DM 88
Program loading 88
Changes during execution 88
Program dumping 89
E6BM 89
Dumping 89
DUMPING OF ZEROS 89
DEVICES THAT MAY BE USED FOR DUMPING 89
PROGRAM INITIATED DUMPS 89
OPERATOR INITIATED DUMPS 89
Loading 90
PROGRAM INITIATED LOADING 90
During the running of a program 90
When a program deletes itself 90
OPERATOR INITIATED LOADING 90
LOad 90
FInd 90
Action of the FInd directive with magnetic tape (industry compatible) and cassette tape as primary backing storage medium 91
Action of the FInd Directive with E.D.S. or F.D.S. 91
OPERATOR/EXECUTIVE COMMUNICATION 92
Monitoring of input messages 92
Description of executive treatment of input typewriter message 92

Chapter 8 Paging 99
THE PRINCIPLES OF PAGING 99
Unpaged systems 99
Paged systems 99
PAGING OPTIONS 99
Programming for paging 99
Addressing 100

TABLES 100
Program table 100
Segment table 100
Page table 101
CURRENT PAGE REGISTERS 101
Loading current page registers 101
Action when store access required 101
Programming restrictions on paged 1904A 102

Chapter 9
This Chapter has been deleted 103

Chapter 10 The 1901 to 1907 central processors 105
GENERAL 105
Individual processors 105
THE 1901 105
Store sizes 105
Store cycle time 105
Number of cabinets 105
Operator communication 105
Peripheral interfaces 106
Fixed-point operations 106
Floating-point operations 106
Other facilities 106
THE 1902 106
Store sizes 106
Store cycle time 106
Number of cabinets 106
Operator communication 106
Peripheral interfaces 106
Fixed-point operations 106
Floating-point operations 106
Other facilities 106
THE 1903 107
Store sizes 107
Store cycle time 107
Number of cabinets 107
Operator communication 107
Peripheral interfaces 107
Fixed-point operations 107
Floating-point operations 107
Other facilities 107
THE 1904 107
Store sizes 107
Store cycle time 107
Number of cabinets 108
Operator communication 108
Peripheral interfaces 108
Fixed-point operations 108
Floating-point operations 108
Other facilities 108

THE 1905E 108
Store sizes 108
Store cycle time 108
Number of cabinets 108
Operator communication 108
Peripheral interfaces 108
Fixed-point operations 109
Floating-point operations 109
Other facilities 109
THE 1906 109
Store Sizes 109
Store cycle time 109
Number of cabinets 109
Operator communication 109
Peripheral interfaces 109
Fixed-point operations 109
Floating-point Operations 109
Other facilities 111
THE 1907 111
Store sizes 111
Store cycle time 111
Number of cabinets 111
Operator communication 111
Peripheral interfaces 111
Fixed-point operations 111
Floating-point operations 111
Other facilities 111

Chapter 11 The E and F central processors 113
GENERAL 113
Individual processors 113
Dual processors 113
THE 1904E 114
Store sizes 114
Store cycle time 114
Number of cabinets 114
Operator communcation 114
Peripheral interfaces 114
Fixed-point operations 114
Floating-point operations 114
Other facilities 114

THE 1904F 114
Store sizes 114
Store cycle time 115
Number of cabinets 115
Operator communication 115
Peripheral interfaces 115
Fixed-point operations 115
Floating-point operations 115

Other facilities 115
THE 1905E 115
Store sizes 115
Store cycle time 115
Number of cabinets 115
Operator communication 116
Peripheral interfaces 116
Fixed-point operations 116
Floating-point operations 116
Other facilities 116
THE 1905F 116
Store sizes 116
Store cycle time 116
Number of cabinets 116
Operator communication 116
Peripheral interfaces 117
Fixed-point operations 117
Floating-point operations 117
Other facilities 117
THE 1906E 117
THE 1906F 117
THE 1907E 117
THE 1907F 117

Chapter 12 The A central processors 119
GENERAL 119
Individual processors 119
THE 1901A 119
Store sizes 119
Store cycle time 119
Number of cabinets 121
Operator communication 121
Peripheral interfaces 121
Fixed-point operations 121
Floating-point operations 121
Other facilities 121
THE 1902A 121
Store sizes 121
Store cycle time 121
Number of cabinets 121
Operator communication 121
Peripheral interfaces 122
Fixed-point operations 122
Floating-point operations 122
Other facilities 122
THE 1903A 122
Store sizes 122
Store cycle time 122
Number of cabinets 122
Operator communication 122

Peripheral interfaces 122
Fixed-point operations 123
Floating-point operations 123
Other facilities 123
THE 1904A 123
Store sizes 123
Store cycle time 123
Number of cabinets 123
Operator communication 123
Peripheral interfaces 123
Fixed-point operations 123
Floating-point operations 123
Other facilities 123
THE 1906A 124
Store sizes 124
Store cycle time 124
Number of cabinets 124
Operator communication 124
Peripheral interfaces 124
Fixed-point operations 124
Floating-point operations 124
Other facilities 124.1

Chapter 13 Other central processor features 125
GENERAL 125
COMMERCIAL COMPUTING FEATURE 125
CONSOLE LOGGING PUNCH 125
CONSOLE TYPEWRITER SWITCH 125
DUALPROGRAMMING 125
EXTENDED MATHEMATICAL UNIT 125
FIXED-POINT OPERATIONS 126
FLOATING-POINT OPERATIONS 126
Extended precision floating-point facilities 126
FLOATING-POINT UNIT 126.4
INSTRUCTION OPERAND COUNTER 126.4
INTERLEAVING 126.4
MILL TIMER 127
MULTIPLY/DIVIDE UNIT 127
MULTIPROGRAMMING 127
OPERATOR COMMUNICATION 127
PAGING 127
PERIPHERAL AUTONOMOUS CONTROL 127
PERIPHERAL CONTROL CONNECTOR 127
REAL TIME CLOCK 127
REMOTE STORE UNIT 128
SCIENTIFIC COMPUTING FEATURE 128
SLOW HESITATION CONTROL 128
STANDARD INTERFACE SWITCHING UNIT 128
STORE ACCESS CONTROL 128
STORE ACCESS MANAGER 128
STORE EXTENSION UNIT 128

STORE MULTI-ACCESS CONTROL 128
SUBPROGRAMMING 128

Index 129

Chapter 1 Introduction to the 1900 Series central processor

The 1900 Series comprises a single range of compatible processors to each of which can be linked various combinations of peripheral devices. Individual central processors are identified by a number from 1901 upwards and are further defined by additional numbers preceded by a solidus, or by additional letters; e.g. 1904/2, 1904E, 1904A.

CENTRAL PROCESSOR CONSTITUENTS

Each central processor is made up of basically similar units. The differences between processors lie in any extra facilities incorporated. Thus there must always be a power supply, and there may be more than one, depending on the size of the configuration and the central processor itself. There must also be a core store, which may hold just over four thousand words of storage or hundreds of thousands of words. Similarly engineers controls, a number of interface channels with associated logic rows, and the central processing unit, incorporating the bulk of the logic circuitry, are present in all central processors. These constituents of the central processor are held in one or more cabinets, for instance as illustrated in Figure 1.

FEATURES OF THE 1900 SERIES CENTRAL PROCESSOR

Compatibility

The compatibility referred to above is fundamental to the philosophy of the 1900 Series. It means that users are able to extend their configuration either by changing central processors or by adding peripherals without reprogramming and without having to change existing equipment that is retained.

Any program written for a 1900 Series central processor can be run on any other processor at the same level or higher in the series hierarchy, or on any smaller processor that has all the facilities used by the program.

A part of this philosophy of compatibility is the I.C.T. standard interface.

Storage

The basic unit of storage in 1900 Series central processors is termed a *word* and consists of 25 consecutive binary digits (*bits*). One of these bits is used for parity checking, a means of ensuring that data is intact after being transferred, so that 24 bits are available to hold data or an instruction.

Standard interface

The I.C.T. standard interface is a standard means of attaching peripheral devices to the central processor. Most standard interface peripherals can be attached to any central processor in the 1900 Series provided that the central processor's hesitation time or data handling capabilities are not overloaded. Some central processors can also be connected to non-standard interface peripherals.

Note: Every 1900 configuration must include a line printer, or there must be a means of producing hard copy from paper tape or cards.

Executive

Each central processor is provided with an Executive control program compiled specifically to suit the processor and peripheral combination in the installation concerned. The Executive program includes routines, called extracodes, that carry out certain functions in the 1900 Series order code. On small processors extracodes are used to perform some functions that are performed by hardware on larger processors. In this way order code compatability between small and large central processors is maintained without the price of small processors being affected.

On medium and large central processors the Executive program can, and in some cases must be, supplemented by one of a number of operating systems. The principal operating systems are the GEORGE systems, details of which can be found in the appropriate manuals.

The console typewriter

In order to allow for human intervention in the operation of the computer, there must be a means of communication between the operator and Executive. In all but the smallest computers, this communication is achieved by means of the console typewriter.

The console typewriter is situated on a free-standing desk and is connected directly to the central processor. The keyboard consists of up to 50 keys and a space bar by means of which the operator can type in instructions. The typewriter is used both to inform the operator of incidents occurring within the central processor, the peripherals and the programs, and also to permit the operator to give instructions such as to load, activate, suspend, or delete programs. These typed instructions and messages form a permanent record of all communications in the sequence in which they occurred.

Large processors in the 1900 Series are supplied with two console typewriters, the second to act as a spare in case of emergencies. A console typewriter switch provides the means of switching control to the spare console typewriter.

On small 1900 computers, a panel of lights and switches is sometimes used instead of a console typewriter to achieve communication between the operator and Executive. Messages are received by the operator in the form of a pattern of lights, and the operator enters instructions by setting a pattern of switches. No automatic record of such communications is kept.

Floating-point arithmetic

Facilities for performing floating-point arithmetic are available with all central processors. By allowing numbers to be stored in the form of an argument and an exponent, floating-point arithmetic provides for calculations involving very small or large numbers using the minimum of storage space. Floating-point functions are performed either by hardware that forms part of the central processor or by extracode.

Programming languages

A number of programming languages, including Compact and full COBOL, PLAN, NICOL, Basic and full FORTRAN and Algol, can be used to write programs to be run on 1900 Series central processor. PLAN is the general purpose language devised specifically for use with the 1900 Series. NICOL, devised for the smaller 1900 Series processors, is a commercially oriented language that is extremely simple to use.

Dual processor capabilities

Certain central processors can be paired with another central processor of the same type to form a dual-processor configuration. Details of dual processor configurations are given in the descriptions of individual processors in Chapters 10 to 12. The principles of dual processor operation are outlined below.

The dual processor systems share control of the peripherals which may be connected to either processor provided that at least one card or paper tape reader and at least one line printer or paper tape punch is connected to each processor. Programs in either processor have access to peripherals connected to either processor.

The processors share a commoncore store that must consist of at least two modules. Each module has a store multi-access control (see page 5) associated with it. Each S.M.A.C. controls access to the module with which it is associated and ensure that only one processor has access to a given module at any one time.

There is one common Executive and operating system which is shared by the two constituent processors and stored in the common core store, but program instructions are able to run in each of the processors simultaneously. A program is not necessarily carried out by one processor but may run at different times in either of the processors on the system. Executive will not allow the same program to run in more than one processor at the same time. The choice of which processor is to be used at any one time for a particular program lies with Executive and/or the operating system.

In the event of one processor of the pair failing, the system can operate in crippled mode, whereby the remaining processor is used on its own. If the fault is a store failure, however, it may be possible to continue with both processors and a reduced store. Otherwise only the peripherals attached to the remaining processor will be available unless a standard interface switching unit is used (see page 53).

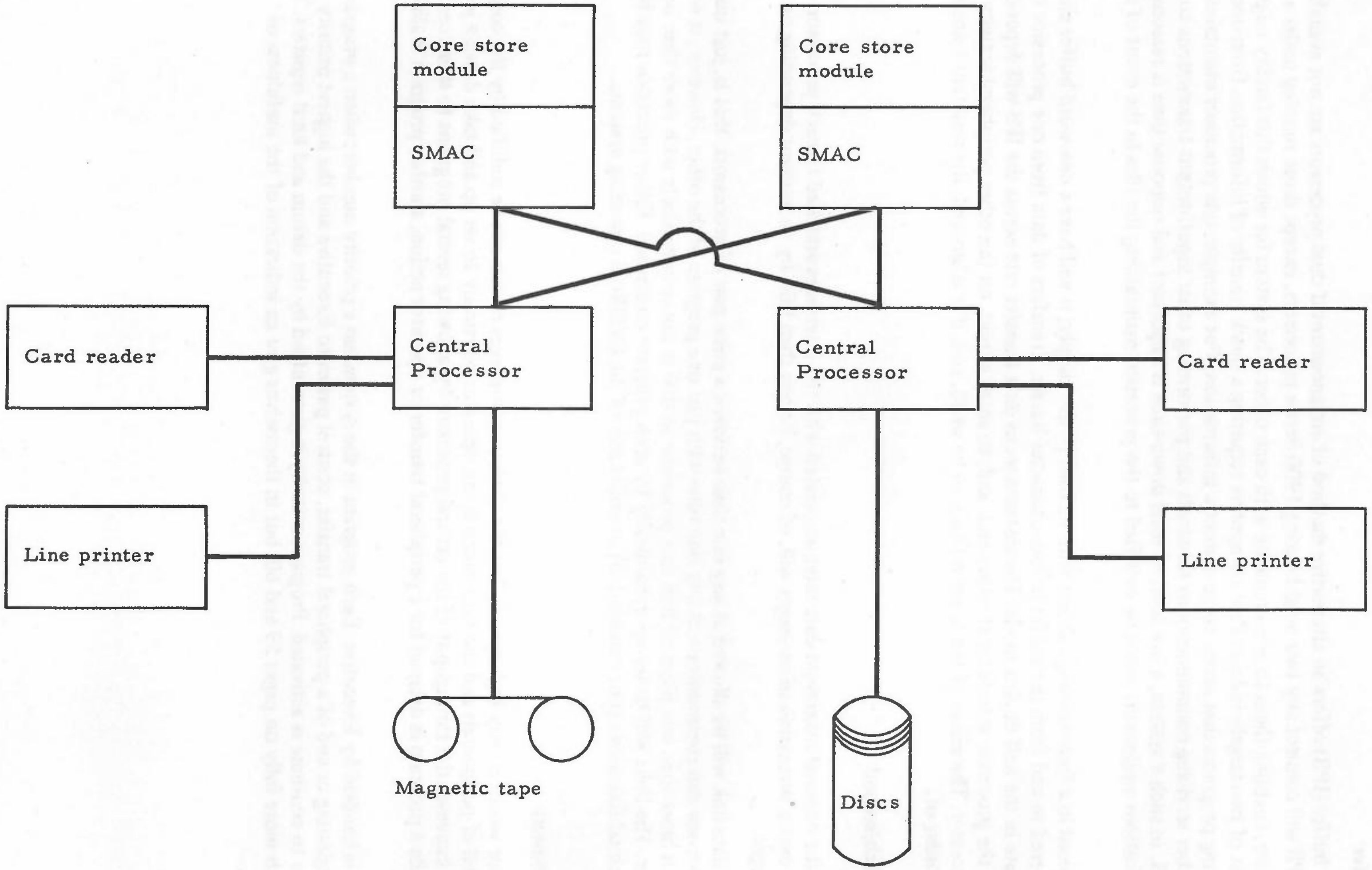


Figure 2 A dual processor configuration

If peripherals are connected via a standard interface switching unit they can all be switched to the remaining processor, provided that the system so formed is viable.

A dual processor configuration is illustrated in Figure 2.

Interprocessor buffer

The interprocessor buffer (IPB) offers an alternative method of enhancement if dual processors are not available or appropriate. The IPB will connect any two neighbouring 1900 Series processors, except those running under a handswitch Executive, enabling them to communicate with each other. The system for which this facility might be suited would consist of two largely independent computers requiring a quick transfer of information from one to the other and running programs that access some common information. For example, one processor maintaining main files and another servicing communications terminals and performing other input/output transactions could be linked by an IPB. In such a system, a low level of total down-time is important and response time is reasonably fast. The communications equipment could be switched to the processor maintaining the files in the event of a hardware failure.

The IPB will be housed in a free-standing cabinet with its own power supply; it will have a one-word buffer and data will be transferred to and from the buffer in four-character bursts. Transfers of data from one processor to the other will operate in the half-duplex mode. The instantaneous data transfer rate across the IPB will depend on the basic speeds of the processor interfaces at either end, and, to some extent, on the other peripheral activity present in each processor. The effect of the latter is likely to be small, and, if it is ignored, the resultant transfer rate will be of the order of:

$$\frac{RS}{R+S} \text{ Kch/second}$$

where R and S are the nominal maximum data rates associated with the interfaces attached to each processor. The sustained data rate over a succession of messages will, of course, be less than this, by an amount depending on the length of the messages.

Only one two-way data link will be allowed at any one time between a given pair of processors, that is, just one program in one processor can communicate at any one time with just one program in the other. However, it will be possible to attach more than one physical link to a processor so that it can communicate with more than one additional processor. The links will be set up dynamically by each program concerned. Either processor may be running under a manual Executive (not handswitch) or under any of the GEORGE operating systems.

MULTIPROGRAMMING

The central processor works at very high speeds, but the obvious advantages of this can be nullified by the comparatively slow speed of peripherals and the time taken by an operator physically to set up and take down a job. Multiprogramming increases the throughput of the central processor by allowing several programs to share central processor time. While a program is delayed for a peripheral transfer or operator action, another program is allowed to run.

Multiprogramming is handled by Executive. Each program in the system has a priority number; when a program interrupts at the beginning or end of a peripheral transfer, control passes to Executive and the highest priority program that is able to continue is activated. Program security is guaranteed by the datum and limit registers. These points are dealt with more fully on pages 59 and 60, but in themselves give an indication of the usefulness of multiprogramming.

Chapter 2 The core store

Note: The description of the operation of a core store given in this chapter is for the interest only of the reader. No inference should be drawn from it.

STORE SIZE

The size of core store in a central processor is given as a number of K words, 1K being 1024 words. Thus a 16K core store is one that consists of 16,384 words of storage. Although for programming purposes the core store may be regarded as a single entity it physically comprises one or more units known as *modules.* Modules vary in size, between 4K and 64K, according to the type of central processor and the size of the total core store. The principal significance of modules is in interleaving (see below) and in the incrementation in size of the core store in a central processor. Any central processor can be supplied with a selection of store sizes, and additional core store can be fitted in the field up to the maximum available with the machine. The additional storage must, however, consist of an integral number of modules of a size available with the machine in question, although the modules of which the store is constituted need not necessarily all be of the same size. The sizes of core store available with particular processors are given in Chapters 10 to 12.

STORE SPEED

Information read from a core store becomes available after a certain interval of time, termed the *access time* . The operation of reading destroys the information read and it must be written back; the next read request cannot be started until a further interval has elapsed. The total time from one read request until the next can be accepted is termed the *cycle time* and is the figure most commonly quoted to give the speed of a core store.

A further consideration in the effective speed of the store is the store *width.* Thus in the 1900 Series a store in which 25 bits are read in parallel would be termed one word wide.

STORE ACCESS

Information is transferred between the core store and the central processing unit or peripherals by means of a store access mechanism. Most 1900 Series central processors have a single store access mechanism. If an access to the store is required less than the store cycle time after a previous access, the latter access must wait unit the cycle is complete. This restriction is irrelevant in most systems since store accesses are rarely or never required so soon after each other. However, as processing speed is improved, the system can become core store limited. This condition is overcome where it is likely to occur by the provision of further store access mechanisms, possibly used in conjunction with an interleaved store.

Store multi-access control

The store multi-access control (S.M.A.C.) is a switching device associated with each core store module in a dual processor system. It controls the access of the central processing units to a given core module, ensuring that only one central processing unit has access to a given core store module at any one time.

The store multi-access control switches access to its associated module from one processor to another as requested or, if there is a store clash, according to a system of priorities. A processor can have access to a store module for only one store cycle without interruption and is unable to request another

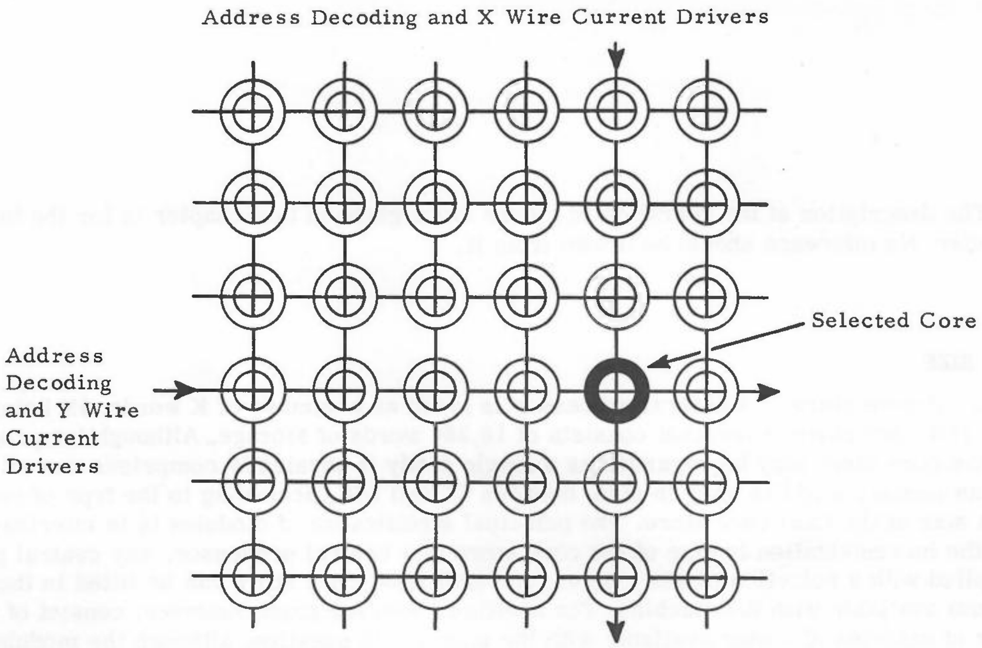

Address Decoding and X Wire Current Drivers
Selected Core
Address Decoding and Y Wire Current Drivers

Figure 3 Example of a magnetic core store matrix

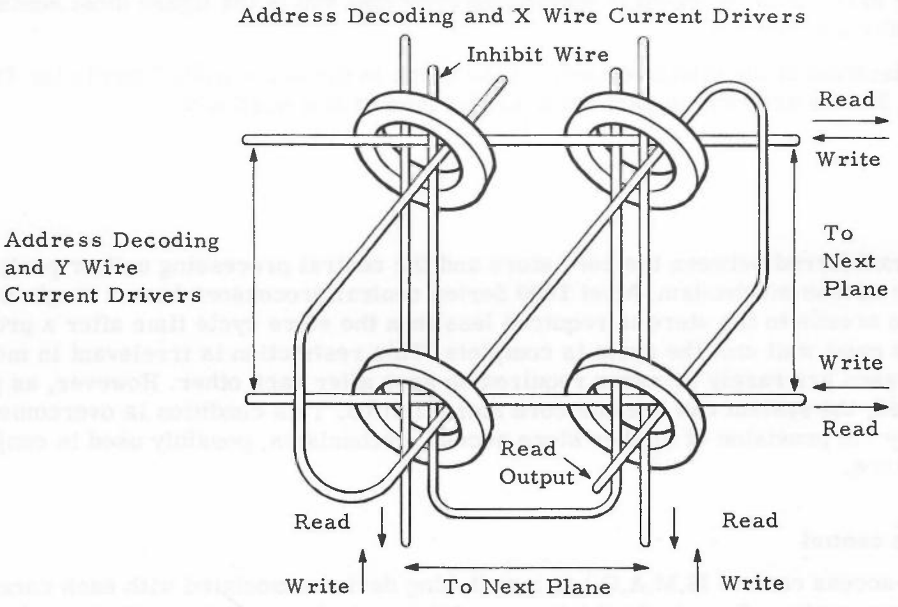

Address Decoding and X Wire Current Drivers
Inhibit Wire
Read
Write
Address Decoding and Y Wire Current Drivers
To Next Plane
Write
Read
Read Output
Read
Write
To Next Plane
Read
Write

Figure 4 Section of a magnetic core store plane

store cycle while that cycle is taking place. If a processor requests access to a store module that is being accessed by the other processor, the S.M.A.C. queues the request.

The basic rule for switching is that a store access request is granted unless there is a clash of requests, in which case access is granted to the processor that did not have access to the module last. This rule is qualified by the proviso that if the request for store access emanated from a peripheral channel, then this request has priority over one originating in the central processing unit. If requests from peripheral channels clash, then the original rule applies.

PRINCIPLES OF READING AND WRITING IN STORE

Reading and writing information in the core store is performed by the following operations.

1 Read Regenerate, in which information is read from the selected core store addresses and then written back in these addresses for further use.

2 Clear Write, in which the selected core store addresses are cleared (zeroized) during the read part of the read/write cycle and new information is written back to these addresses during the write part of the read/write cycle.

3 Read-Pause-Write, in which information is read from the selected core store addresses and is then updated by new information and written back to these addresses.

The uses of these operations are given below in the short account of the principles involved in reading from and writing to the core store.

A magnetic core store physically consists of matrices of ferrite rings threaded with wire through which current can be passed to magnetize the rings (see Figure 3). Each core (ring) can be magnetized in either direction in terms of two remanent states of magnetization. That is, if a current of sufficient strength is applied to the core and then removed, the core will settle to a certain stable state of magnetization. If this current is reversed and then applied and removed, the core will settle to another stable state of magnetization. These two states of remanence are known as positive remanence and negative remanence. Positive remanence is used to indicate 1 and negative remanence to indicate 0, so that each core can be used to store one bit.

A section of a plane of a core store matrix is shown in Figure 4 and it can be seen that each core is threaded with a number of wires: current drive, read and inhibit wires. The current drive wires are arbitrarily called the *X* and *Y* current drivers and any particular core can be addressed by selecting the two appropriate *X* and *Y* wires. This selection is performed by the store selection electronics.

If a current equal to half the strength required to magnetize a core is applied simultaneously to the selected *X* and *Y* wires the result will be a magnetizing current of sufficient strength being applied to the core at the intersection of the two wires. This method of applying half currents to the *X* and *Y* wires is used so that the only core affected by the full current is that which is at the intersection. Other wires through which these half currents pass are not affected because the half current is not of sufficient strength to change the magnetic state of the cores.

To read the state of a core, the method used is to write a 0 to the core and note the change, if any, in the state of the core. To enable this to be done, a third wire, called the read wire, is threaded through every core in the plane and a current will be induced electro-magnetically in this wire only if the state of the core changes. Thus, if the core had previously been storing a 0, no response will be given from the wire, but if the core had been storing a 1, a response will be given indicating a change in the core's state. At the end of this read cycle all the selected cores in the matrix will have been switched to the 0 state. Therefore, if it is necessary to retain the information in store, the information just read must be written back immediately. For this purpose the read cycle is followed by a regenerate cycle.

During the regenerate cycle the selected *X* and *Y* wires have a reverse current applied to them such that the cores at the intersections have a magnetizing current that is sufficient to return them to the 1 state. To prevent this reverse current switching cores that were read as 0's, every core in the matrix is threaded with a fourth wire called the inhibit wire. If a 0 was read, then a current is applied to the inhibit wire coincident with the half currents applied to the *X* and *Y* wires. The resulting net magnetizing current linking with the selected core is insufficient to switch the core from the 0 state. When the program is writing to core store, the write cycle is always preceded by the read cycle described above, which erases the store address's previous content by filling it with 0's. The method used to write information to store is similar to that of regeneration, the difference being that the currents applied to the inhibit wires are derived from the information to be written instead of from the information read from the store during the read cycle.

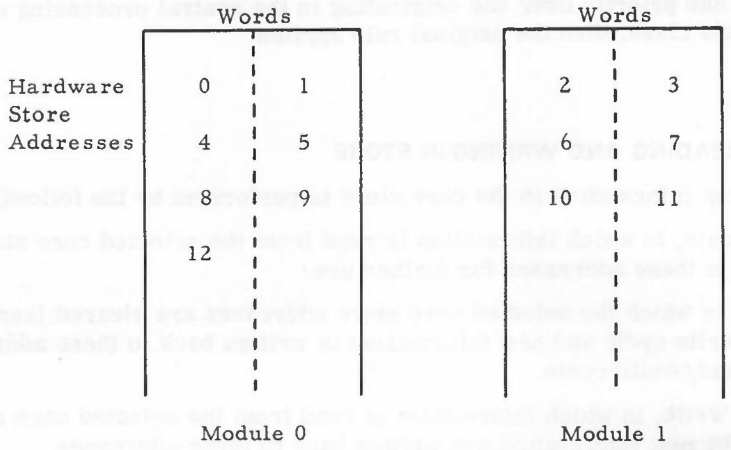


Figure 5 A two-way interleaved store

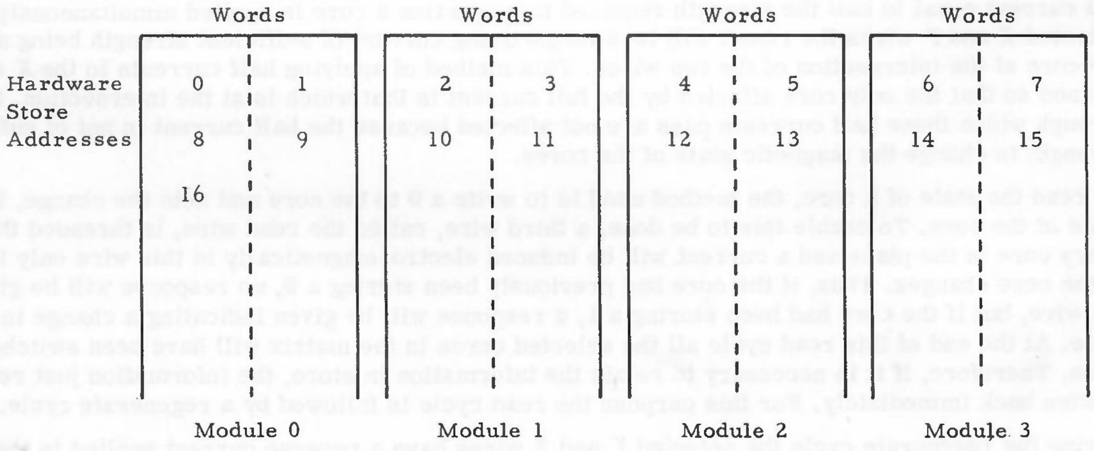


Figure 6 A four-way interleaved store

PARITY CHECKING

Parity checking is the term given to the practice whereby the number of 1-bits in a word is counted to ensure that it is always odd. When data and instructions are originally entered into the computer system, a count is made of the number of 1-bits. If the number is even, bit 25 is set to 1, thus making the total odd. Subsequently, every time the information is transferred from one location to another, a check is made to ensure that odd parity is maintained, this being sufficient evidence of a successful transfer.

Each central processor has a parity check unit which checks the parity of the information transferred from the core store during a read cycle, and generates the parity bit for information transferred to the core store during a write cycle. During a write cycle, the parity of the word, generated by the parity check unit, is written to the word as bit 25. During a read cycle, the parity of the word read from the core store is generated and compared with the content of the parity bit. If the comparison fails, the central processor halts. 1902A and 1903A processors however will halt if the parity error is in Executive. In object mode an interrupt occurs and Executive can determine which location is in error.

STORE INTERLEAVING

Store interleaving is a facility that, used in conjunction with two or more store access mechanisms, provides a means of avoiding core store limitation in a system. In an interleaved store each store access mechanism is associated with one module or group of modules. The store hardware addresses are arranged so that each successive address or pair of addresses, depending on the width of the store, is in a different module. The 1900 Series central processors that can have interleaved stores provide for either two- or four-way interleaving.

Two-way interleaving

A two-way interleaved store that is two words wide will have its hardware store addresses arranged as shown in Figure 5. Since each store mechanism can access two words at once, and there are two such mechanisms, a total of four words can be accessed at one time.

Four-way interleaving

A four-way interleaved store that is two words wide will have its hardware store addresses arranged as shown in Figure 6. A total of eight words can therefore be accessed at one time.

Efficiency increase

The effect of store interleaving depends on the relative speeds of the central processing unit and the store and also on the functions being performed. A program containing many floating-point instructions is less likely to be store limited, so that interleaving will have less effect. On the other hand, a sequence of instructions with short times will be performed with a considerable time saving if the store is interleaved. Interleaving will in any case greatly reduce the percentage degradation of C.P.U. performance caused by store access clashes resulting from coincident peripheral transfers.

Store failure

If part of the store fails it is not possible to continue work with the store interleaved in the same manner. However, with minimal intervention by I.C.T. engineers, it is possible to isolate the defective module and reconfigure the store so that work can be continued with reduced or no interleaving.

ALLOCATION OF STORAGE

The core store can be considered as being divided into two distinct areas; an Executive area, and an object program area. The Executive area contains the Executive program which, as one of its functions, allocates areas of core store to the object programs entered into the system. Executive is loaded into the store from Word 0 onwards and the number of words occupied by a specific Executive depends on the configuration of the installation on which it is used. Object programs are stored in the areas of core store immediately following the area reserved for Executive. The absolute starting address of an object program area allocated by Executive is always a multiple of a number that varies according to the central processor concerned as follows:

1901	128 words
1902/3	256
1904/5	64
1906/7	128
1904,5/E,F	64
1906,7/E,F	64
1901A	64
1902,3/A	64
1904A	64
1906A	64

Consequently, the first word of an object program will not necessarily be the word following the last word of the preceding program in store.

The number of words of core store required by a specific object program is calculated and rounded up to a multiple of 64 by the compiler and the result is included in the requisition slip that precedes that program. When the program is loaded, this value is used by Executive to determine whether sufficient storage is available to allow the program to be accepted. If sufficient storage is available, Executive allocates an appropriate area to that program and records the absolute starting address and ending address of the area; the program is then entered into the store.

The difference between the programmer's relative word address and the absolute word address is adjusted by hardware by reference to information held by Executive. If the central processor is operating in datum/limit mode (for details of paging mode see Chapter 8), each time a program is activated Executive sets into a hardware register, known as the datum register, the absolute word address corresponding to Word 0 (starting address) of that program area. For example, if the first word of the object program area were held in absolute address 2176, then Executive would set this value in the datum register. At each store reference by the object program the datum value is added to the relative address used within that program to obtain the absolute address.

A hardware register, known as the limit register, is also used to determine the limit of store area that an object program may reference. On the smaller 1900 Series central processors there is no actual limit register, but a prewired number is permanently set to correspond to the last absolute word address of core store. On multiprogramming machines, the limit value is variable and is determined by Executive in a similar manner to the datum value. On all of the central processors a check is carried out by hardware to ensure that no store reference is made by an object program to an address less than datum or greater than relative address limit. Thus, in a core store containing more than one object program and containing both a datum and limit register, each program is completely protected from any other program.

Each time an object program is run on a multiprogramming central processor, the program will almost certainly occupy a different area of core store from the area that it occupied on its previous run. However, this aspect of store allocation is automatically provided for by the fact that, as previously stated, each program area is considered to be numbered from zero relative to that program's datum. This method of programming by relative addresses allows programs to be positioned anywhere in the store without the need for the programmer to know the absolute store area that his program will occupy on a specific run.

When a program is finished and no longer required in the store it is deleted, that is, it is removed from the Executive directory of active programs. If necessary a multiprogramming Executive will then relocate the programs in the store; that is, it will move up programs in higher addresses towards the Executive area in order to fill up the area left by the deleted program. Thus, all free addresses are then in one continuous area. The Executive datum and limit records for the programs that have been moved are amended to the new absolute addresses.

LAYOUT IN STORE OF OBJECT PROGRAM

For the purposes of programming each area of core store can be considered as consisting of a number of distinctive sections, as illustrated opposite.

Limit of object program →

Upper Data Store
Program Store
Lower Data Store
Reserved Store
Switch Word
Entry Parameters Store
Reserved Store
Accumulators

Datum of object program →

The relevance of the sections is briefly described below.

Accumulators-Words 0 to 7

These eight consecutive words constitute the program's accumulators and are used for arithmetic, copying, testing and logical functions. Accumulators 1, 2 and 3 can also be used as modifying registers.

Reserved store and entry parameters store - Words 8 to 29

These two sections occupy an area from Word 8 to Word 29. Certain words within this area are reserved exclusively for use by Executive. Other words in this area may be used by Executive and/or, subject to certain restrictions, by an object program. The purpose and restrictions on the use of the words in this area are as follows:

Word 8 This word is used for control purposes by Executive and by the hardware of the central processor; it should not be changed by the object program. It is Executive's link word for that current object program number. (Note: Executive links are not of the same format as object program subroutine links, and their format varies according to the central processor used.)

Word 9 This word is used by Executive to provide a reply to the object program after certain peripheral instructions have been used. Word 9 is also used by the hardware of certain central processors. An object program may access Word 9 only in accordance with the specification of the relevant instructions.

Words 10 and 11 These words are used primarily for communication between parts of the object program written by the user, and standard ICL subroutines called in this object program. They are not used by the operating environment.

Words 12 and 13 An object program must not alter and cannot rely on the contents of these locations. They are used by Executive to hold or dump the contents of the normal two-word floating-point accumulator.

Words 14 and 15 An object program must not alter and cannot rely on the contents of these locations, if the program member makes use of extended precision floating-point facilities, or the own monitoring of illegals (and floating-point overflow or underflow, if applicable). The locations are overwritten upon each own monitoring entry to the member, and used as a dump for the extension to the floating-point argument and variable limit register.

Words 16 to 19 These words are reserved for use by the various compilation systems. An object program may use these words only in accordance with the compiler specification.

Words 20 to 29 These words are used by compilers to store entry parameters; that is, instructions referring to entry and restart points in the object program.

Switch word - Word 30

This word contains the setting of the 24 operator's switch bits. Switch 0 is represented by the most significant bit of Word 30. The 'ON' state of a switch bit is represented when the appropriate bit is '1'. An object program may examine and alter the state of any of the bits in Word 30.

Reserved words - Words 31 to 44

This word is subject to the same restrictions as Words 16 to 19; additional restrictions apply if subprogramming is used, (see below) but in this case the additional areas reserved for subprogramming start at Word 32.

Lower data store - Words 45 up to 4,095

The lower data store normally starts at Word 45 and the area that it occupies varies according to the needs of a particular program with the limitation that this area may never extend beyond Word 4,095. The lower data store consists of those areas in which the programmer wishes to hold information that will be frequently and directly addressed.

The division of data storage is introduced for the reason that all program instructions, except branch instructions, can address directly only the first 4,095 words of the object program area in which they are used. This limit is imposed by the fact that the N field of the instruction word format consists of only twelve bits (see Chapter 5). Thus the maximum address that can be held in the N field of the instruction is that for Word 4,095. To refer to words beyond Word 4,095, indirect addressing entailing the use of modifier registers must be employed. The division of the two data store areas, which is under the programmer's control, may therefore be used to ensure that the lower store is reserved to hold information frequently and directly accessed, while the upper store is reserved for information of a lengthy or infrequently used nature, such as input and output areas and tables that would normally be accessed by modified instructions anyway.

The programmer must specify to the compiler the areas required; the number of words of core store specified will then be allocated by the compiler during the compilation of the object program.

Program store

The location of the program store area will vary according to the lower data store requirements of the program, but it always immediately follows the lower data store. The program store contains the program instructions.

BRANCH MODES

There are two branch modes in the 1900 Series order code, normal branch mode and extended branch mode. If the central processor is working in normal branch mode, the address part of a branch instruction is limited to 15 bits. This effectively limits the addressable size of store holding instructions to 32K. To allow for larger core stores, the central processor may alternatively work in extended branch mode, in which case the effective N address of the branch instruction consists of 22 bits, and the | addressable size of core store holding instructions is extended to 4M, that is, 4,096K. For details of these modes see Chapter 5.

Uppe. data store

The upper data store immediately follows the program store and its location will vary according to the needs of a particular program. The upper data store is used to store data of lengthy or infrequently used nature, which will normally be addressed by modified instructions.

ADDRESSING MODES

There are two addressing modes in the 1900 Series order code, 15-bit address mode (15AM) and 22-bit address mode (22AM). Instructions in a program to be run in 15AM have 12 bits for the address of the operand; by modification, the address size can be extended to 15 bits so that data can be stored in an area of 32K. To allow for larger core stores, a program can alternatively be run in 22AM, 22 bits being allowed for the operand address. Thus data can be stored in an area of 4M. For details of these modes see Chapter 5.

Additional words reserved when subprogramming is employed

When subprogramming is employed, the datum of the object program area is common to all members of a subprogram group. Furthermore, all members use the same accumulator area. To eliminate the programming difficulties this condition could cause, Executive dumps and restores Words 0 to 15 inclusive when switching from one member to another. For this purpose an additional sixteen words of the object program area are reserved for each member; for example, if a program has Members 0, 1 and 2 the number of additional words reserved will be 3 x 16 = 48. Thus the lower data store will begin at Word 80.

AVAILABILITY OF CORE STORE

The sizes of core store available with particular processors are given in the description of the processors concerned in Chapters 10 to 12. If a large core store, requiring free-standing core store cabinets, is fitted, it is usually necessary to have a store extension unit (S.E.U.) and possibly remote store units (R.S.U.s) as well. These devices are needed to supply extra power and to control transfers of information between cabinets. A store extension unit is fitted in the first core cabinet and a remote store unit is fitted in each free-standing cabinet. The devices are supplied as standard for every configuration that requires them and, since they are contained within the core cabinets, they take up no extra space.

Chapter 3 The arithmetic and control units

GENERAL

The arithmetic and control units are the part of the central processor principally concerned with the manipulation and flow of data throughout the computer configuration. These units are made up of gates and registers, and associated with the control unit is at least one timing generator.

CLOCK PULSES

Clock pulses are produced at intervals by the timing generator or generators and are used to control the operation of microprograms. Every instruction comprises a number of operations, collectively known as the microprogram, and each of these operations must be completed in the interval between pulses, the next pulse triggering the next operation. Each instruction is therefore performed over an integral number of pulses.

The intervals at which pulses occur vary from one central processor to another and from one operation to another within any one central processor. The interval is usually between 0.5 and 6 microseconds, the long delay occurring when a store cycle is included in the operation. A mill timer, or clock pulse counter as it is also known, may be provided to count the number of pulses occurring during a program.

If all operations within the control and arithmetic units are associated with the pulses produced by one timing generator the central processor is termed *synchronous*. However, within a synchronous central processor, certain features may have their own control and timing generator, in which case they can operate autonomously. Such is the case, for instance, with the P.A.C. and the S.A.M. (see Chapter 6) and some floating-point hardware. A distinction between the floating-point hardware supplied with the 1905 and that which may be supplied with the 1901 is that the former can operate autonomously whereas the latter cannot.

THE CONTROL UNIT

The control unit provides the means of controlling the carrying out of instructions and is also used in the control of peripheral transfers. Its principal components are:

1 A control register, in which the control address is stored. The control address is the address of the next instruction to be obeyed, and it is normally increased by unity after each instruction has been obeyed, so that the next instruction may be read from store when required.

2 Circuits that decode the function part of an instruction and set up other circuits to obey the instruction.

3 Circuits that decode the address part of an instruction so that the required data can be read from the appropriate store address.

THE ARITHMETIC UNIT

The arithmetic unit consists of a mill and registers that enable calculations and logical functions to be performed. Its principal components are:

1 A mill, or adder as it is sometimes termed.

2 One or more registers into which operands may be transferred whilst being operated upon.

3 Facilities for shifting operands right or left for such purposes as multiplication and division and facilities for forming the inverse of a value.

LAYOUT OF THE CONTROL AND ARITHMETIC UNITS

The arithmetic and control units are illustrated in Figure 7. This figure represents a typical control unit and arithmetic unit as found in many 1900 Series computers. However, the smaller computers in the range, below the 1902, have fewer registers, and the larger computers, such as the 1906 and 1907, have additional registers. The number of registers used affects the price and speed of the machine; nevertheless the principles of operation are in all cases the same.

The mill

The mill is a 24-bit parallel adder/subtractor that can carry out logical and arithmetic operations on the contents of the registers when these are transferred to the mill. When the operation has been performed, the result is transferred back to a register. On less expensive 1900 processors the width of the mill is reduced to six bits thereby eliminating a considerable amount of circuitry but also increasing the times of arithmetic operations.

Register A

Register *A* is used as a working register in the performance of intermediate operations. It has normal and inverse outputs, the latter being used to form the complements of numbers for binary subtraction. This register is also used in high speed mode peripheral transfers (see page 50).

Register B

Register *B* is used as a working register and also to transfer operands to and from store. Parity is formed from the contents of *B* when store accesses are made.

Register P

Register *P* is used to hold the instruction address (Bits 9 to 23), special Executive modes (Bits 2 to 8), the carry from multiple length operations (Bit 1) and a record of any overflow that occurs during the execution of an instruction (Bit 0).

When more complex instructions, such as multiplication, are performed, register *P* is required as a working register, and its ability to shift operands left or right internally is used. When *P* is used as a working register, its normal contents are temporarily stored in location eight of the program's store area.

Datum and limit registers

These registers are used to store the datum and limit of the program being executed. The datum is the first word in store of the area reserved for the program, and the limit is the last word of the program's area. (See also Chapter 7 Multiprogramming.) Single program machines have no limit register since the limit is constant, the last word in store, and it is impossible to violate it.

Register N

Register *N* contains various quantities as an instruction progresses. Initially it contains the instruction address and then the operand address part of the instruction. It also has the ability to count down in decrements of unity and is used as a counter in instructions such as multiply and shift.

Registers F,X,M

These registers are used to hold the function, accumulator address and modifier address of an instruction respectively. Register *X* has an additional output *X*+1 that is used to specify the second accumulator used in double length operations.

HOW INSTRUCTIONS ARE PERFORMED

A method of gathering insight into the way in which the control unit, arithmetic unit and store work together is to run through the short program, "add x to y and place the result in the store".

Assume that:

1 Operand x is in store address 2003;

2 Operand y is in store address 2004;

3 The answer is to be placed in store address 2005;

Three instructions are required and these are:

4 Transfer the content of store address 2003 (x) to an accumulator, $X1$, (function 000), where $X1$ is also in store.

5 Add the content of store address 2004 (y) to the content of $X1$ (function 001).

6 Transfer the content of $X1$ $(x+y)$ to store address 2005 (function 010).

Assume that these instructions are located in store addresses 5000, 5001, 5002.

FIRST INSTRUCTION

1 Sends the control address (equal to 5000) from the control register to store selection.

2 Receives the content of address 5000 (first instruction) and separates the function and address parts.

3 Decodes the function, which causes the address bits to be sent to the store selection circuits and sets up the route for store read-out of Word 2003 to the arithmetic unit, and thence to Register A.

4 Causes the content of store address 1 to be cleared and then the contents of Register A to be copied to store address 1.

5 Unity is automatically added to the control address in order that the next instruction may be retrieved from store.

SECOND INSTRUCTION

1 Sends the control address (equal to 5001) from the control register to store selection.

2 Receives the content of address 5001 (second instruction) and separates the function and address parts.

3 Decodes the function, which causes the address bits to be sent to the store selection circuits, sets the route for the store contents of Word 2004 to Register A, then reads.

4 Sets 1 on the store address highway. Reads the contents of address 1 to the mill and causes the mill to add the incoming number to the content of Register A, storing the sum in Register B then writing to Word 1.

5 Unity is automatically added to the control address in order that the next instruction may be retrieved from store.

THIRD INSTRUCTION

1 Sends the control address (equal to 5002) from the control register to store selection.

2 Receives the contents of address 5002 (third instruction) and separates the function and address parts.

3 Decodes the function, which causes the address bits to be sent to the store selection circuits and sets the route to the arithmetic unit from the store. Reads Word 1 to Register A.

4 Sets 2005 on the store highway, clears word 2005 and causes the content of Register A to be transferred to store address 2005.

5 Unity is automatically added to the control address in order that the next instruction may be retrieved from store.

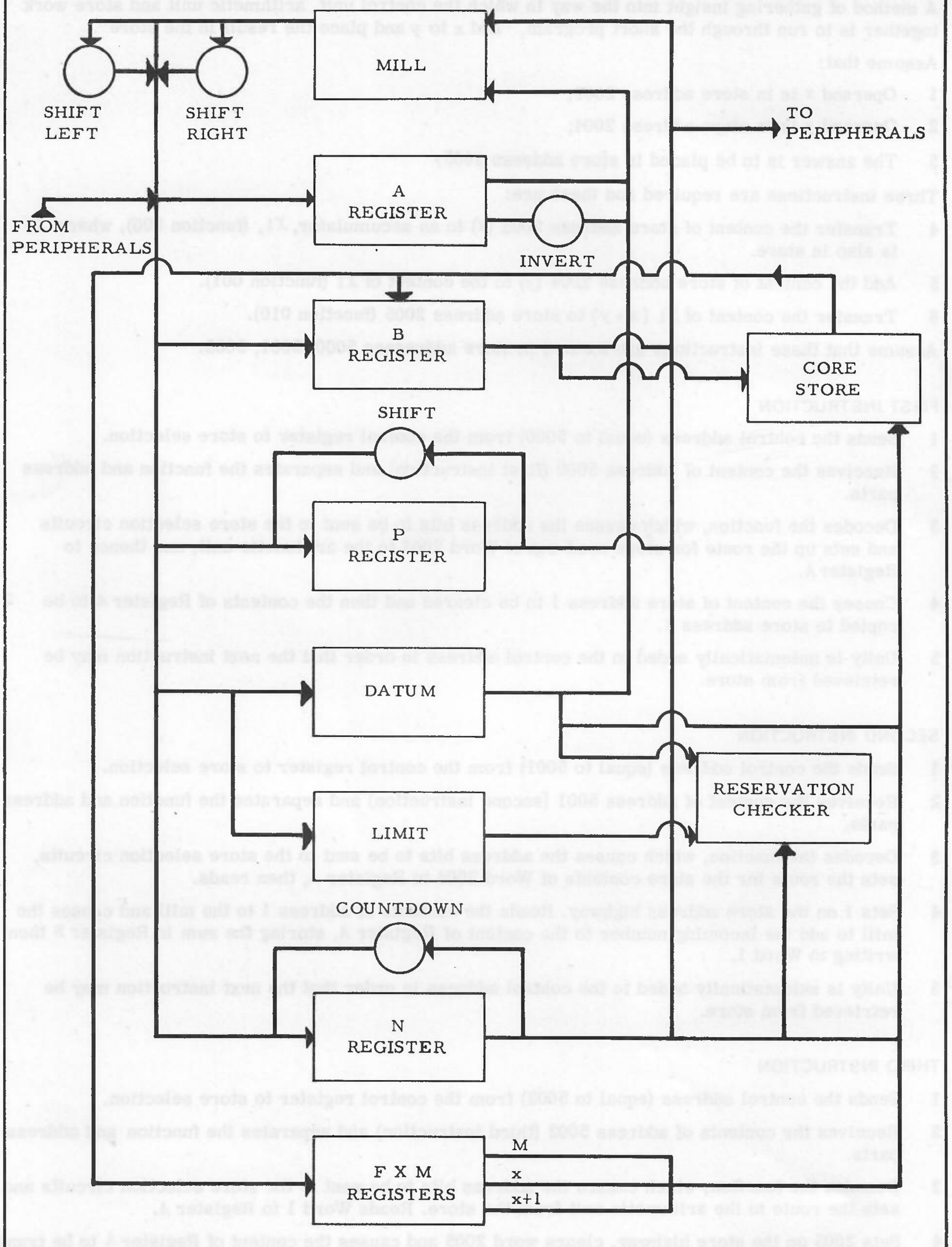


Figure 7 The arithmetic and control units

Chapter 4 Representation of data

WORD FORMAT

As stated previously, the basic unit of storage on the 1900 Series computers is called a *word* and consists of 24 consecutive bits. For programming purposes, these 24 bits are numbered from 0 to 23 starting at the most significant (left-hand) end of the word.

A word can hold data in various forms, the 24 bits being interpreted according to the manner in which the word is used.

CHARACTER FORM

Data is nearly always input in character form, i.e. in the form of decimal numbers, letters of the alphabet, and other symbols such as commas, solidi, and asterisks. This is evidently the most convenient form for the user. Data may remain stored in this form provided that it is not involved in any calculation other than the production of hash totals. If data is to be involved in calculations it must first be converted into pure binary form, i.e. one of the forms described below.

A character of data is stored in six successive bits, character positions being referred to for programming purposes as n_0 to n_3 as shown in the following diagram.

n_0	n_1	n_2	n_3
character 0	character 1	character 2	character 3
0 to 5	6 to 11	12 to 17	18 to 23

Bit positions

PURE BINARY FORM

When data is held in character form each pattern of six bits has a unique significance but individual bits have no significance at all. When data is held in pure binary form, individual bits have a defined value and the value of any group of bits is the total of the values of individual bits in the group.

Data in pure binary form is always interpreted as having a numerical value. Apart from counter modifier words, words holding pure binary data generally have the most significant bit (Bit 0) reserved to indicate whether the value contained in the rest of word is positive or negative. If Bit 0 is set to zero the value of the word is positive, and if Bit 0 is set to one the value of the word is negative. If a number is negative, it is expressed as a complement.

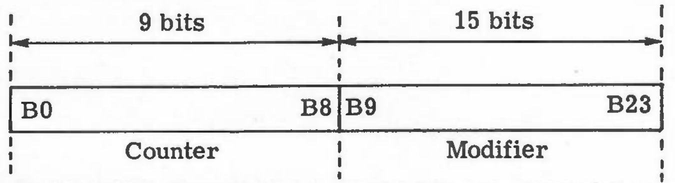
Signed numbers may be integers, fractions, mixed, fixed- or floating-point, and may be single or multiple length, i.e. may be held in one or more words. The interpretation of the form in which data is held depends on the instruction operating on the data. For instance, two of the divide instructions (044 and 045) will always interpret the dividend as a double length number, and a floating-point instruction will always assume data to be held in floating-point form.

Counter modifiers

A counter modifier, or index, word can be used in two forms: as a word counter modifier or as a character counter modifier. A counter modifier word can be used to hold both count and modifier only in 15AM (see Chapter 5), or if held in lower data store in 22AM.

WORD COUNTER MODIFIER

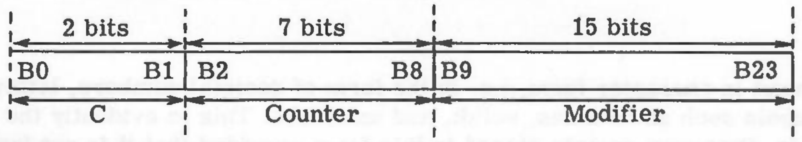
A word counter modifier has the following format:



The counter (Bit 0 to Bit 8) contains a count of the number of times an operation is to be performed. The counter must lie in the range 0 to 511; a count of zero will be treated as a count of 512 by the appropriate instruction. The modifier (Bit 9 to Bit 23) contains the word address involved in the operation. For each repeat of the operation the counter is decreased by 1 and the modifier is increased by either 1 or 2 according to the instruction used.

CHARACTER COUNTER MODIFIER

A character counter modifier has the following format:



The character modifier C (Bit 0 and Bit 1) contains the character positions 0, 1, 2 or 3 within a word. The counter (Bit 2 to Bit 8) contains a count of the number of characters involved in the operation. The counter must lie in the range 0 to 127; a count of zero will be treated as a count of 128 by the appropriate instruction.

The modifier (Bit 9 to Bit 23) contains the word address involved in the operation. After each operation 1 is subtracted from the counter and added to the character number (Bit 0 and Bit 1). If the resultant character number is 4, the character modifier is zeroized and 1 is added to the word modifier (Bit 15 to Bit 23).

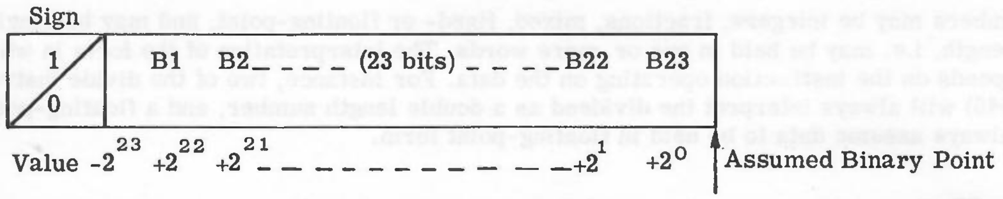
COUNTER MODIFIER WORDS IN 22 AM OR EBM

If a counter modifier word is held in lower data store it may use the above formats irrespective of the addressing mode in which the program is running. Otherwise, in 22AM or EBM two words must be used, one word to hold the count and the other to hold the modifier. The least significant 22 bits of the modifier word are used to hold the modifying value for the word address, the remaining two bits being available to modify the character address. The count is held as a binary number at the less significant end of the count word.

Fixed-point numbers

SINGLE-LENGTH INTEGER

For a single-length integer the binary point is assumed to be immediately to the right of Bit 23. Single length integers lie in the range -2^{23} to $+2^{23}-1$ inclusive.


Sign
1
0
B1
B2_ _ _ _ (23 bits) _ _ _B22
B23
Value -2^{23} $+2^{22}$ $+2^{21}$ _ _ _ _ _ _ _ _ _ _ _ $+2^1$ $+2^0$
Assumed Binary Point

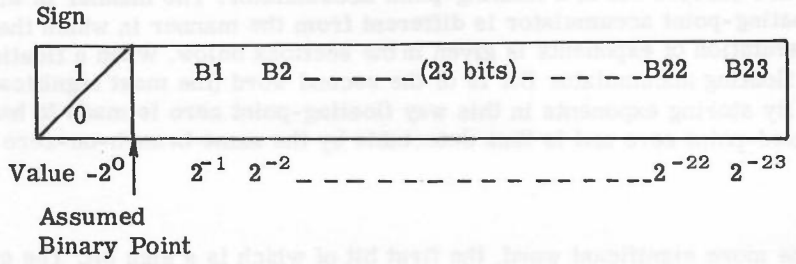
A negative integer is stored as its complement with respect to 2^{24}, i.e. $-n$ is stored as $2^{24}-n$.

MULTI-LENGTH INTEGER

For a multi-length integer the binary point is assumed to be immediately to the right of the least significant word. Bit 0 of all words other than the most significant must be set to zero and is irrelevant to the value of the number. Multi-length integers lie in the range of -2^{23n} to $+2^{23n}-1$ where n is the number of words in which the number can be held.

SINGLE-LENGTH FRACTION

For a fixed-point fraction the binary point is assumed to be between Bit 0 and Bit 1, i.e. immediately to the right of the sign bit. Fixed-point fractions lie in the range -1.0 to +1.0 -2^{-23} inclusive, a negative fraction being stored as its complement with respect to 2^0.

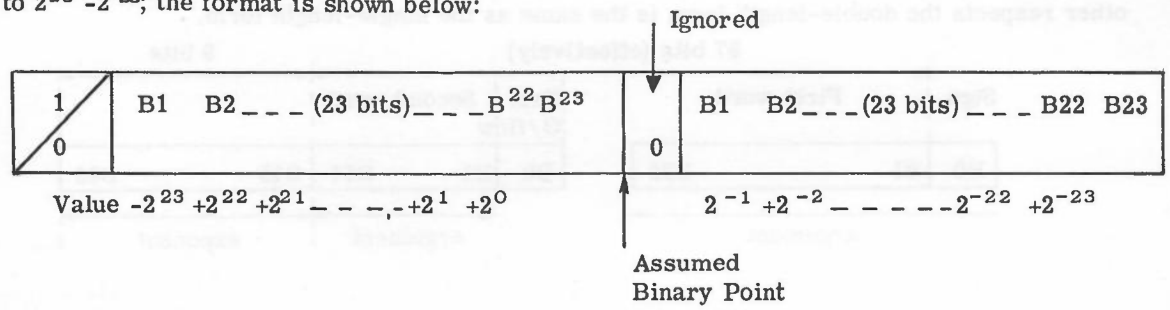


MULTI-LENGTH FRACTION

For a multi-length fraction the binary point is assumed to be between Bit 0 and Bit 1 of the most significant word. Bit 0 of all words other than the most significant must be set to zero and is irrelevant to the value of the number. Multi-length fractions lie in the range of -1.0 to +1.0 -2^{-23n} where n is the number of words in which the number can be held.

MIXED NUMBER

For a mixed number the binary point can be assumed to lie between any two bits of the one or more words holding the number. However, the normal 1900 Series convention is to use two words to store a mixed number: one word for the integral part and one word for the fractional part. In this case the binary point is assumed to lie between Bit 23 of the more significant word and Bit 1 of the less significant word. Bit 0 of the less significant word must be zero and is irrelevant to the value of the number. The term used to refer to a number of this type is *mid-point number.* The range for mid-point numbers is -2^{23} to 2^{23} -2^{-23}; the format is shown below:



Ignored
1
0
B1 B2 _ _ _ (23 bits) _ _ _ B^{22} B^{23}
0
B1 B2 _ _ _ (23 bits) _ _ _ B22 B23
Value -2^{23} $+2^{22}$ $+2^{21}$ _ _ _ , $-+2^{1}$ $+2^{0}$
2^{-1} $+2^{-2}$ _ _ _ _ _ 2^{-22} $+2^{-23}$
Assumed Binary Point

Floating-point numbers

The use of floating-point arithmetic greatly extends the numerical range of the central processor and at the same time relieves the programmer of the responsibility of correctly positioning the binary point during protracted mathematical operations.

A number n, in floating-point form, consists of an argument (or mantissa) r and an exponent e such that $n = r . 2^e$ where

r is a signed fractional argument in the range $1 > r \geqslant \frac{1}{2}$,

or $-\frac{1}{2} > r \geqslant -1$ or, in exceptional cases, $r = 0$

e is a signed integral exponent in the range $-256 \leqslant e \leqslant 255$.

These ranges correspond approximately to a decimal range for n of $-10^{76} < n < 10^{76}$. For smaller numbers in the range $-10^{-77} < n < 10^{-77}$, n is considered as zero.

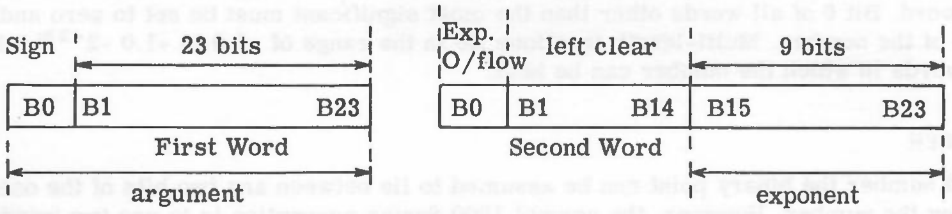
Before floating-point instructions can be used, the data concerned must be converted to floating-point form. The conversion is achieved by scaling a number to a convenient fractional size and storing the scaling factor as the exponent. Scaling is performed by a shift operation, the number of places shifted being the scaling factor.

Floating-point numbers can be held in single-, double-, or quadruple-length form. The double-length form is standard. The single-length form is used only with the 114 (NORM) instruction and the quadruple-length form is permissible only on certain central processors or with certain languages, e.g. FORTRAN.

Floating-point operations are carried out in a floating-point accumulator. The manner in which exponents are held in the floating-point accumulator is different from the manner in which they are held in store. The store representation of exponents is given in the sections below. When a floating-point number is loaded into the floating accumulator Bit 15 of the second word (the most significant bit of the exponent) is inverted. By storing exponents in this way floating-point zero is made to have the same representation as fixed-point zero and is thus detectable by the same branch-on-zero instructions.

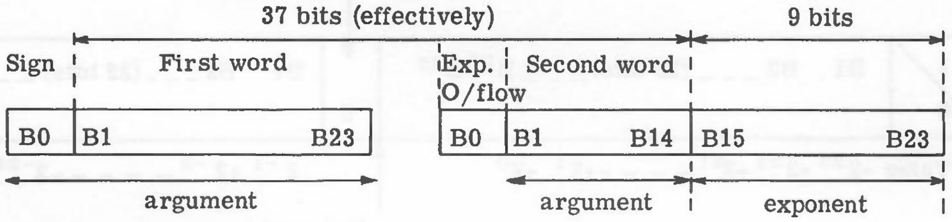
SINGLE-LENGTH

The argument is held in the more significant word, the first bit of which is a sign bit. The exponent is held in the least significant nine bits of the less significant word. The exponent is held in the form $e + 256$; thus any value below 256 indicates a negative exponent. Bit 0 of the less significant word is used to indicate exponent overflow; Bits 1 to 14 inclusive are left clear.



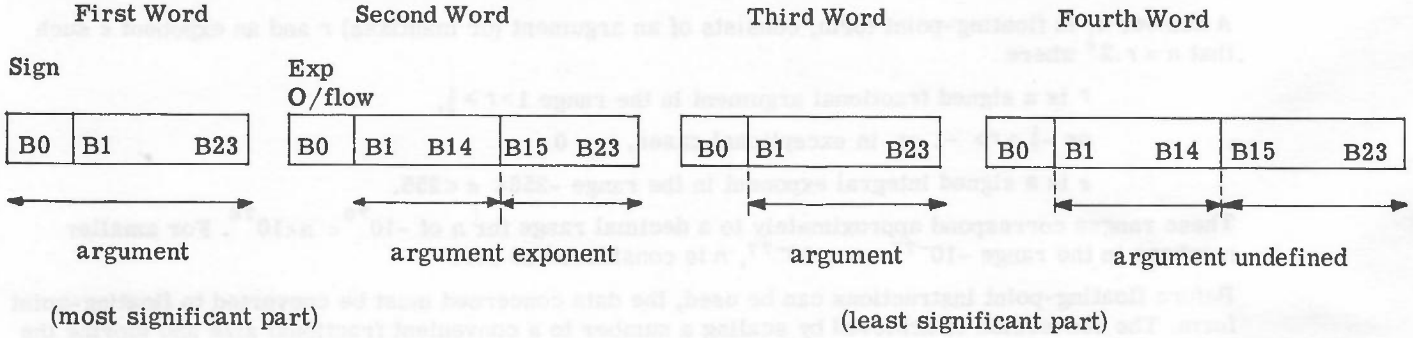
DOUBLE-LENGTH

The argument is held in the more significant word and Bits 1 to 14 of the less significant word. In all other respects the double-length form is the same as the single-length form.



QUADRUPLE-LENGTH (EXTENDED PRECISION)

The first two words of an extended precision floating-point number have the same contents as a double-length floating-point number. The third word and the fourteen most significant bits of the fourth word hold the least significant extension of the argument. The remainder of the fourth word is undefined.



OVERFLOW AND CARRY

One-bit registers associated with, but stored separately from, a program are used to record the occurrence of overflow and carry conditions.

Carry

Carry occurs in multi-length working when a quantity originally stored in one of the less significant words can no longer be held in 23 bits. As a consequence, the sign bit, which will have been set to zero originally, becomes set equal to one. When this occurs, it is cleared and the carry register (*C*) is set equal to one.

The carry register can be utilized whenever carry is likely to occur: any appropriate instruction will clear the condition, by adding one to the next more significant word, and re-setting the carry register to zero.

Overflow

FIXED-POINT

Overflow occurs when a single word working area, or the most significant word of a multi-length working area, becomes too small to hold a quantity. As a result, the value of the sign bit is changed and the overflow register (*V*) is set equal to one. The condition can be detected by testing the overflow register whenever overflow is likely to occur.

If overflow has been caused by a single addition, subtraction or multiplication (or by a factor of less than two as a result of a division) then the error in the answer will be minus two if the answer appears positive and plus two if the answer appears negative. Recovery can therefore be programmed.

112 and 113 (SRAV) and 114 and 115 (NORM) instructions are especially useful in this respect.

FLOATING-POINT

If, during any operation in the floating-point accumulator, the exponent attempts to exceed 255, exponent overflow is said to occur and Bit 0 of the second word is set to 1, the state of the other bits being indeterminate. If a floating-point number with Bit 0 of the second word set to 1 is loaded into the floating-point accumulator, this bit remains set.

The setting of overflow can be tested directly only on processors whose order code includes the 076 (BFP) instruction. However, if the exponent overflow bit is set and the contents of the floating-point accumulator are transferred to store (instruction 137, SFP) or converted to mid-point form and stored (instruction 131, FIX) then *V* will be set. In the case of instruction 137, Bit 0 of the second word of the result will also be set to 1.

If the exponent of a floating-point number attempts to become less than −256, the value of the floating-point number is regarded as zero and the argument and exponent are set accordingly. This may also occur when the exponent becomes equal to −256. Floating-point zero thus has the same representation as fixed-point zero. However, it is important to note that if underflow occurs during a sequence of operations in which overflow has previously occurred, exponent overflow will continue to be indicated by the setting to one of the exponent overflow bit. In this case floating-point zero will not have the same representation as fixed-point zero.

FIELD DESCRIPTIONS

There is a need to clarify words that, when applied to fields read or written by an object program, have specialised meanings. This terminology is explained below.

Defined

Full details of every option are recorded.

Ignored

The object program may store any value in an ignored field but cannot rely on its being preserved.

Reserved

This term indicates a field set aside for future use. When reading a reserved field, the object program should ignore the field as its contents cannot be relied upon. When writing a reserved field, the object program should zeroise the field.

Chapter 5 Format of instructions

GENERAL FORMAT

The general word format of a program instruction as stored in the computer is represented symbolically as:

X F M N

where

X is the accumulator field and specifies one of eight accumulators (0 to 7). These are the first eight words of the object program area and are used to store one of the operands to be used by the instruction.

F is the operation field and specifies the function the instruction is to perform.

M is the modifier field and is zero or the address of an accumulator (1, 2 or 3) whose content, if any, is to be used to modify the N field.

N is the operand field and is the core store address containing the other operand on which the instruction acts:

Certain program instructions use the accumulator and/or the operand fields for special purposes. These special cases are explained under the applicable types of program instructions.

There are four basic formats for instructions in the 1900 Series order code. An instruction is always held in a single word. The basic formats are:

1 Normal instructions

2 Branch instructions

3 Shift instructions

4 Floating-point instructions

Normal instructions

This format covers such instructions as add, multiply, divide, subtract and store.

Field Symbol	*X*	*F*	*M*	*N*
Number of Bits	3	7	2	12
Bit Positions	0 to 2	3 to 9	10 and 11	12 to 23

Branch instructions

There are two sets of conditional branch instructions and two unconditional branch instructions in the 1900 Series order code. The first set of conditional instructions branch according to the contents of the accumulator whose number is stored in the *X* field. The second set of conditional instructions branch according to the state of *V*, *C*, or the floating-point accumulator. This second set of instructions has a single function code which is modified by the contents of *X*.

The two unconditional branch instructions, 070 and 074 with *X* = 0, are similar except that the former stores the link setting in *X*.

All branch instructions have the same basic format. There is no *M* field, since they cannot be modified, although the 023 instruction exists to give the same effect as would be achieved by modifying a branch instruction.

Field Symbol	*X*	*F*	*N*
Number of Bits	3	6	15
Bit Positions	0 to 2	3 to 8	9 to 23

Shift instructions

The N field of a shift instruction is subdivided into N_t and N_s. Certain groups of shift instructions have the same function code which is modified by the contents of Nt.

Field Symbol	X	F	M	Nt	Ns
Number of Bits	3	7	2	2	10
Bit Positions	0 to 2	3 to 9	10 and 11	12 and 13	14 to 23

Nt specifies the type of shift and is considered as part of

Ns specifies the number of places of shift.

Floating-point instructions

The value in the X field of a floating-point instruction does not represent an accumulator but qualifies the function code. The value of X is given by the programmer.

Field Symbol	X	F	M	N
Number of Bits	3	7	2	12
Bit Positions	0 to 2	3 to 9	10 and 11	12 to 23

The group 16 instructions have a similar format.

ADDRESSING AND BRANCH MODES

Concepts

There are two addressing modes and two branch modes in the 1900 Series order code. The essential difference between the modes in both cases is the size of the address field.

The most frequently used modes are *15-bit address mode* (15AM) and *direct branch mode* (DBM). Both these modes allow a maximum of 15 bits to hold an address, which gives a limit of addressability within any program of 32K words. Thus the program and its data areas must not be larger than 32K words.

To allow larger areas of store to be addressed *22-bit address mode* (22AM) and *extended branch mode* (EBM) are provided. Facilities for extending the address to 22 bits allow an area up to 4M words of store to be occupied by any program and its data areas.

These modes indicate the instruction code and addressing features assumed by the program and form part of the information stored by Executive for each program. When a program is activated the central processor is switched into the appropriate addressing and branch mode.

Setting and switching of modes

MODE WORD

The mode word holds the mode setting of a program in Bits 21 and 23 as follows:

Bit 21 = 0 indicates direct branch mode

= 1 indicates extended branch mode

Bit 23 = 0 indicates 15-bit address mode

= 1 indicates 22-bit address mode

A program may be run in any combination of these modes in a suitable environment. The remaining bits of the mode word are undefined; i.e. they should be set to zero when a mode word is produced, but should not be assumed to be zero when a mode word is examined.

MODE SETTING ON PROGRAM LOADING

The mode word is Word 1 of the supplementary request block. This block is produced by the compiling process, or when a program is dumped, for any program not in 15AM or DBM. The decision as to whether a program is to be run in one mode or another is therefore taken by the compiler although this decision may be influenced by the programmer's use of compiler directives. A supplementary request block is acceptable only to Executives with 22AM or EBM capabilities; an attempt to run a program compiled in 22AM or EBM on an Executive with only 15AM and DBM capabilities will therefore result in the rejection of the program. For further details of the supplementary request block, see page 86.

MODE SWITCHING DURING PROGRAM EXECUTION

The 165 (GIVE) instruction is provided for enquiring the current mode setting and for changing the mode setting, as follows:

165 $N(M)$ = 8 Enquire mode setting. *X* contains the current mode word.

165 $N(M)$ = 9 Change mode setting. A mode word with the required new setting must be in *X*. A reply is given in *X* in the form of the mode setting actually achieved.

On processors without 22AM or EBM capability, the reply to these instructions will always be zero. The action to be taken in the event of a change being unsuccessful is left to the program.

MULTI-MEMBER PROGRAMS

The members of a multi-member program are treated independently and may run in different operating modes. The instruction to enquire and change mode settings (165) applies only to the member that issues it.

The initial mode setting of Member 0 is determined by the supplementary request block. Members other than Member 0 obtain their initial mode setting from the mode of the members that activate them. When a multi-member program is dumped, the mode setting of Member 0 only is recorded. On subsequent reloading, all members will again take their initial setting from the members that activate them.

22AM considerations

MODIFICATION

All modification, whether by supplementary (117, SMO) orders or using an index register quoted in the instruction, is carried out using 22 bits from the modifier word. Character modifiers in an index register additionally use the two most significant bits to give the character address.

COUNT INSTRUCTIONS

Because Bits 2 to 8 form part of the address in 22AM working, the 9 bit/15 bit division of a counter modifier word cannot be used if the word is required as a modifier. The action of the count instructions 060 (BUX), 062 (BDX) and 064 (BCHX) is therefore different in 22AM. These instructions increment the address in *X* and branch unconditionally to *N*.

Instruction 066 may be used in conjunction with these *increment only* instructions, the counting being carried out in a separate accumulator.

SUBROUTINE ENTRY AND EXIT

The format of the link in *X* must allow for a 22-bit address. The setting of the zero suppression mode is stored on entry in Bit 1, instead of in Bit 8 as in 15AM, and the remaining contents of the mode setting are lost. On exit the zero suppression mode setting is restored from Bit 1.

It is therefore inadvisable, unless it is known that the zero suppression mode is not set, to enter a subroutine in 22AM and leave it in 15AM, since the zero suppression mode will be restored incorrectly. If entry is in 15AM and exit in 22AM the zero suppression mode setting will be treated as part of the link address.

CONSOLE DISPLAYS

In 15AM the address and length of the message to be output by means of a 160 instruction is held in a control word having a counter modifier layout. In 22AM the same format may be used; alternatively, a word pair may be used, the first containing the number of characters in the message and the second the start address.

EBM considerations

Extended branch mode introduces two new forms of branching, relative and replaced. In the former, jumps forwards or backwards of limited extent are carried out relative to the address of the branch instruction. If the extent of the jump is beyond the scope of the relative address, the replaced form must be used; this allows reference to a location containing the full 22-bit address of the destination. The destination address in effect replaces the address given in the instruction. The latter technique is sometimes known as indirect addressing.

A section of code containing only relative branches, and which does not amend itself, may be obeyed in any part of the store. It may in fact be obeyed in different locations on different occasions. This is possible because the branch address is relative to the location holding the branch instruction and not an absolute address that might hold another instruction if the section of program were moved. Relative branches are more efficient in terms of time and space, and are therefore produced from source branch instructions whenever possible.

Bit 9 of the instruction is used to distinguish between relative and replaced branches as follows;

Bit 9 = 0 Bits 10 to 23 are interpreted as a signed relative address

Bit 9 = 1 The address in bits 10 to 23 is that of the location containing the destination address.

THE 023 (OBEY) AND BRANCH INSTRUCTIONS

If a 023 instruction addresses a relative branch instruction, the branch is performed relative to the location of the 023 instruction, not that of the branch instruction. Thus a branch to a labelled location, for example, will not be correctly performed.

It is necessary for the programmer to be aware of this fact if PLAN is used, since the compilation process cannot take account of it.

SUBROUTINE LINKS

In EBM working the subroutine link format is as for 22AM working, and similar considerations apply (see above).

SUMMARY OF FUNCTION CODES

A summary of the functions in the 1900 Series order code is given below. Each main function is described by a three-digit octal number. The numbering of functions is arranged so that similar functions are grouped, each group being referenced by the two most significant digits of the numbers in the group. Thus functions 041 to 047, which are the multiplication and division instructions, are referred to as group 04, and the 041 function is described as group 04, function 1.

The symbols used to describe the action of functions are defined below.

Definition of notation

Note: A prime (apostrophe) after a symbol indicates a value resulting from the relevant operation; e.g. a' = the contents of the floating-point accumulator after the instruction has been performed.

Symbol	Meaning
A	the floating point accumulator
a	the contents of A
Bi	the i^{th} bit of a word
BN	the branch address (see below)
C	the carry register
c	the contents of c (0 or 1)
DN	the operand address (see below)
e	the exponent of a floating-point number
E	$e + 256$
F	a function
$FOVR$	the floating-point overflow register.
M	a modifier (registers 1 to 3)
m	the contents of the modifier register (zero if $M = 0$)
N	a core store address or 12-bit number
$N(M)$	the modified core store address or 15-bit number

Symbol	Meaning
n	the contents of N, after modification if applicable
n'	the contents of N (after modification if necessary) after an instruction has been obeyed
$n:$	the double-length contents of N and $N + 1$
N_e	the least significant 9 bits of N
N_{em}	a 22-bit address
N_m	a 15-bit address
N_r	a 14-bit address
N_t	the most significant 2 bits of the 12-bit N address
N_s	the least significant 10 bits of the 12-bit N address
R	the branch-specifying bit in extended branch mode
RI	the address of the instruction to which control is transferred
S	the sign bit
V	the overflow register
X	an accumulator (registers 0 to 7)
X^*	the accumulator adjacent to X. $X^* = X+1$ except that $X7^* = X0$
x	the contents of X
x^*	the contents of X^*
x'	the contents of X after an instruction has been obeyed
$x:$	the double-length contents of two consecutive accumulators
$x:'$	the double-length contents of two consecutive accumulators after an instruction has been obeyed
x_a or n_a	the least significant twelve bits of x or n
x_c or n_c	the 9-bit counter at the more significant end of x or n
x_d or n_d	the least significant seven bits of x_c or n_c
x_e or n_e	the least significant nine bits of x or n. The exponent of a floating-point number occupies this portion of the second word
x_{em}	the least significant 22 bits of x
x_j or n_j	any one of x_0, x_1, x_2, x_3 the four 6-bit characters of x or n
x_k or n_k	the most significant two bits of x or n
x_m or n_m	the least significant fifteen bits of x or n

DERIVATION OF DN AND BN

Definition of DN

DN is defined according to the value of M as follows:

When $M = 0$ then $DN = (N + p)q$

When $M = 1, 2,$ or 3 then $DN = (N + m + p)q$

Where m = the contents of the modifier register

p = the supplementary value specified by a preceding 117 (SMO) instruction (zero if none).

q = the number of bits in which the address is contained. q is dependent upon the addressing mode in which the program is running, i.e. q is 15 in 15AM and 22 in 22AM.

Note: If the most significant bit of $(m)q$ or $(p)q$ is equal to 1, the effect will be as though the value were negative. If the resultant DN is negative, then when used as an address it may cause a reservation violation, or when the datum has been added may lie beyond the lower end of store.

Definition of BN

The definition of *BN* depends upon the current setting of the branch mode, the value of *q* and, in EBM, on the value of *R* (Bit 9) of the instruction.

When in DBM, $BN = (N_m+p)q$

Where Nm = a 15-bit address

When in EBM and $R = 0$, $BN = (I+N_{r\rightarrow 22}+(p)_{q\rightarrow 22})22$

Where Nr = a 14-bit address

I = the instruction address

$\rightarrow 22$ = implies extension to 22 bits, if in 22AM, by propagating the most significant bit of the value concerned.

Note: Extension is to ensure correct subtraction effects if p or N_r are negative. The 14-bit value in N_r is regarded as a signed integer.

When in EBM and $R = 1$, $BN = (n_{em}+(p)_{q\rightarrow 22})22$

Where n_{em} = the 22-bit content of address N_r

Definition of functions

GROUP 00

Functions 000 to 003 clear *C* but may set *V* on exit. Functions 004 to 007 cannot set *V*; the sign of the result is always positive and *C* is set if appropriate.

Function	000	(LDX)
Definition	$x' = n + c$	Any mode
Description	Write $n + c$ into *X*	
Function	001	(ADX)
Definition	$x' = x + n + c$	Any mode
Description	Add $n + c$ to x	
Function	002	(NGX)
Definition	$x' = -n - c$	Any mode
Description	Write $n + c$ negatively into *X*	
Function	003	(SBX)
Definition	$x' = x - n - c$	Any mode
Description	Subtract $n + c$ from x	
Function	004	(LDXC)
Definition	$x' = n + c$	Any mode
Description	Write $n + c$ into *X*	
Function	005	(ADXC)
Definition	$x' = x + n + c$	Any mode
Description	Add $n + c$ to x	
Function	006	(NGXC)
Definition	$x' = -n - c$	Any mode
Description	Write $n + c$ negatively into *X*	
Function	007	(SBXC)
Definition	$x' = x - n - c$	Any mode
Description	Subtract $n + c$ from x	

GROUP 01

Functions 001 to 003 clear C but may set V on exit. Functions 014 to 017 cannot set V; the sign of the result is always positive and C is set if appropriate. These instructions are similar to Group 00 but with n and x interchanged.

Function	010	(STO)	
Definition	$n' = x + c$		Any mode
Description	Write $x + c$ into N		
Function	011	(ADS)	
Definition	$n' = n + x + c$		Any mode
Description	Add $x + c$ to n		
Function	012	(NGS)	
Definition	$n' = -x -c$		Any mode
Description	Write $x + c$ negatively into N		
Function	013	(SBS)	
Definition	$n' = n - x - c$		Any mode
Description	Subtract $x + c$ from n		
Function	014	(STOC)	
Definition	$n' = x + c$		Any mode
Description	Write $x + c$ into N		
Function	015	(ADSC)	
Definition	$n' = n + x + c$		Any mode
Description	Add $x + c$ to n		
Function	016	(NGSC)	
Definition	$n' = -x -c$		Any mode
Description	Write $x + c$ negatively into N		
Function	017	(SBSC)	
Definition	$n' = n -x -c$		Any mode
Description	Subtract $x + c$ from n		

GROUP 02

Instructions 020 to 022, 024 and 025 clear C. None of this group can set V.

Function	020	(ANDX)	
Definition	$x' = x$ and n		Any mode
Description	Logical *and* of x and n		
Function	021	(ORX)	
Definition	$x' = x_{v}\ n$		Any mode
Description	Logical *inclusive* or of x and n		
Function	022	(ERX)	
Definition	$x' = x \not\equiv n$		Any mode
Description	Logical *exclusive or* of x and n		
Function	023	(OBEY)	
Definition	(See Description)		Any mode
Description	Obey the instruction in location $N + m$ as if it were in this instruction address.		

Function 024 (LDCH)

Definition $x' = n_j$ Any mode

Description Write into x_3 the character n_j (Extract character)

Function 025 (LDEX)

Definition $x' = n_e$ Any mode

Description Write into x_e the least significant nine bits of x_n (Extract exponent)

Function 026 (TXU)

Definition (See Description) Any mode

Description Set C if $n \neq x$ or $c = 1$ (Test equality); otherwise clear C

Function 027 (TXL)

Definition (See Description) Any mode

Description Set C if $n + c > x$; otherwise clear C

GROUP 03

All these instructions clear C; none can set V.

Function 030 (ANDS)

Definition $n' = n$ and x Any mode

Description Logical *and* of n and x

Function 031 (ORS)

Definition $n' = n_v\ x$ Any mode

Description Logical *inclusive or* of n and x

Function 032 (ERS)

Definition $n' = n \not\equiv x$ Any mode

Description Logical *exclusive or* of n and x

Function 033 (STOZ)

Definition $n' = 0$ Any mode

Description Clear n

Function 034 (DCH)

Definition $n'_j = x_3$ Any mode

Description Write x_3 into character position j of N leaving the rest unchanged (Insert character)

Function 035 (DEX)

Definition $n'_e = x_e$ Any mode

Description Write x_e into the least significant nine bits of N leaving the rest unchanged (Insert exponent)

Function 036 (DSA)

Definition $n'_a = x_a$ Any mode

Description Write x_a into the least significant twelve bits of N leaving the rest unchanged

Function 037 (DLA)

Definition $n'_m = x_m$ Any mode

Description Write x_m into the least significant fifteen bits of N leaving the rest unchanged.

GROUP 04

All these instructions clear C and may set V.

Function	040	(MPY)	
Definition	$x{:}' = n.x$		Any mode
Description	Unrounded multiplication		
Function	041	(MPR)	
Definition	$x{:}' = n.x + 2^{-24}$		Any mode
Description	Rounded multiplication		
Function	042	(MPA)	
Definition	$x{:}' = n.x + x^*$		Any mode
Description	Semi-cumulative multiplication		
Function	043	(CDB)	
Definition	$x{:}' = 10.x{:} + n_j$		Any mode
Description	Decimal to binary conversion		
Function	044	(DVD)	
Definition	$x^{*\prime} = x{:}/n, x' =$ remainder		Any mode
Description	Unrounded double-length division		
Function	045	(DVR)	
Definition	$x^{*\prime} = x{:}/n + 2^{-24}, x' =$ remainder		Any mode
Description	Rounded double-length division		
Function	046	(DVS)	
Definition	$x^{*\prime} = x^*/n, x' =$ remainder		Any mode
Description	Single-length integral division		
Function	047	(CBD)	
Definition	$x{:}' = 10.x{:}, n_j' =$ character		Any mode
Description	Binary to decimal conversion		

GROUP 05

All these instructions clear C; none can set V.

Function	050	(BZE)	
Definition	(See Description)		Any mode
Description	Branch to BN if $x = 0$		
Function	052	(BNZ)	
Definition	(See Description)		Any mode
Description	Branch to BN if $x \neq 0$		
Function	054	(BPZ)	
Definition	(See Description)		Any mode
Description	Branch to BN if $x \geqslant 0$		
Function	056	(BNG)	
Definition	(See Description)		Any mode
Description	Branch to BN if $x < 0$		

GROUP 06

All these instructions clear *C*; none can set *V*.

Function	060 (BUX)	
Definition	$x'm = x_m + 1;\ x'_c = x_c - 1$	15AM
Description	Branch to *BN* if $x'_c \neq 0$ (Single word modify)	
Definition	$x'_{em} = x_{em} + 1;\ x'_k = x_k$	22AM
Description	Branch unconditionally to *BN*	
Function	062 (BDX)	
Definition	$x'_m = x_m + 2;\ x'_c = x_c - 1$	15AM
Description	Branch to *BN* if $x'_c \neq 0$ (Double word modify)	
Definition	$x'_{em} = x_{em} + 2;\ x'_k = x_k$	22AM
Description	Branch unconditionally to *BN*	
Function	064 (BCHX)	
Definition	The 15AM definition depends on the value of x_k as follows:	
	If $x_k = 0$, 1, or 2, then $x'_k = x_k + 1;\ x'_m = x_m;\ x'_d = x_d - 1$	
	If $x_k = 3$, then $x'_k = 0;\ x'_m = x_m + 1;\ x'_d = x_d - 1$	
Description	Branch to *BN* if $x'_d \neq 0$ (Character modify)	
Definition	The 22AM definition depends on the value of x_k as follows:	
	If $x_k = 0$, 1, or 2, then $x'_k = x_k + 1;\ x'_{em} = x_{em}$	
	If $x_k = 3$, then $x'_k = 0;\ x'_{em} = x_{em} + 1$	
Description	Branch unconditionally to *BN*	
Function	066 (BCT)	
Definition	$x'_m = x_m - 1;\ x'_c = x_c$	15AM
Description	Branch to *BN* if $x'_m \neq 0$	
Definition	$x'_{em} = x_{em} - 1;\ x'_k = x_k$	22AM
Description	Branch to *BN* if $x'_{em} \neq 0$	

GROUP 07

Function	070 (CALL)	
Definition	(See Description)	Any mode
Description	Store in *X* the address of the next instruction and branch to *N*. Clear *V* and *C*. (Subroutine entry.)	

Note: The contents of *X* vary according to the mode setting as follows:

15AM and DBM	Bit 0	contains the setting of *V*
	Bit 8	contains the setting of the zero suppression mode
	Bits 9 to 23	contains the address of the instruction following the 070
15AM and EBM; 22AM	Bit 0	contains the setting of *V*
	Bit 1	contains the setting of the zero suppression mode
	Bits 2 to 23	contain the address of the instruction following the 070.

Function	072 (EXIT)	
Definition	(See Description)	Any mode

Description Restore control to the instruction whose address is in X or to $x + N$ if N is non-zero. Clear C; if V is set leave it set, otherwise restore it; restore the zero suppression mode. (Subroutine exit.)

Note: In 15AM and DBM the zero suppression mode setting is stored in Bit 8 of X; otherwise it is stored in Bit 1 of X. The address of the instruction to which control is transferred is defined according to the addressing and branch mode setting as follows:

15AM and DBM $RI = (N_m+p+x_m)_{15}$

15AM and EBM;
22AM $RI = ((N_m)_{\rightarrow 22}+(p)_{q\rightarrow 22}+x_{em})_{22}$

If the combination 15AM and EBM is used, the formula is true only if the processor is capable of operating in 22AM

Function 074 $X = 0$ (BRN)

Definition (See Description) Any mode

Description Unconditional branch to N

Function 074 $X = 1$ (BVS)

Definition (See Description) Any mode

Description Branch to N if V is set and leave V unaltered

Function 074 $X = 2$ (BVSR)

Definition (See Description) Any mode

Description Branch to N if V is set; clear V

Function 074 $X = 3$ (BVC)

Definition (See Description) Any mode

Description Branch to N if V is clear and leave V unaltered

Function 074 $X = 4$ (BVCR)

Definition (See Description) Any mode

Description Branch to N if V is clear, otherwise clear V

Function 074 $X = 5$ (BCS)

Definition (See Description) Any mode

Description Branch to N if C is set and clear C

Function 074 $X = 6$ (BCC)

Definition (See Description) Any mode

Description Branch to N if C is clear and leave C unaltered

Function 074 $X = 7$ (BVCI)

Definition (See Description) Any mode

Description Branch to N if V is clear and set V; otherwise clear V

Function 076 $X = 0$ (BFP)

Definition (See Description) Any mode

Description Branch to N if $a = 0$; V is set if FOVR is set; FOVR is unaltered

Function 076 $X = 1$ (BFP)

Definition (See Description) Any mode

Description Branch to N if $a \neq 0$; V is set if FOVR is set; FOVR is unaltered

Function 076 $X = 2$ (BFP)

Definition (See Description) Any mode

Description Branch to N if $a \geqslant 0$; V is set if FOVR is set; FOVR is unaltered

Function 076 $X = 3$ (BFP)

Definition (See Description) Any mode

Description Branch to N if $a < 0$; V is set if FOVR is set; FOVR is unaltered

Function 076 $X = 4$ (BFP)

Definition (See Description) Any Mode

Description Branch to N if floating-point overflow is clear

Function 076 $X = 5$ (BFP)

Definition (See Description) Any mode

Description Branch to N if floating-point overflow is set

GROUP 10

Instructions 100,102 and 104 to 107 cannot set V; instructions 100 to 104 cannot set C.

Function 100 (LDN)

Definition $x' = N(M) + c$ Any mode

Description Write $N(M) + c$ into x

Function 101 (ADN)

Definition $x' = N(M) + c$ Any mode

Description Add $N(M) + c$ to x

Function 102 (NGN)

Definition $x' = -N(M) - c$ Any mode

Description Write $N(M) + c$ negatively into X

Function 103 (SBN)

Definition $x' = x - N(M) - c$ Any mode

Description Subtract $N(M) + c$ from x

Function 104 (LDNC)

Definition $x' = N(M) + c$ Any mode

Description Write $N(M) + c$ into X

Function 105 (ADNC)

Definition $x' = x + N(M) + c$ Any mode

Description Add $N(M) + c$ to x

Function 106 (NGNC)

Definition $x' = -N(M) - c$ Any mode

Description Write $N(M) + c$ negatively into X

Function 107 (SBNC)

Definition $x' = x - N(M) - c$ Any mode

Description Subtract $N(M) + c$ from x

GROUP 11

All these instructions clear C.

Function 110 $N_t = 0$ (SLC)

Definition $x' = x$ shifted left N_s places Any mode

Description Single-length circular shift

Function	110 $N_t = 1$	(SLL)
Definition	$x' = x$ shifted left N_s places	Any mode
Description	Single-length logical shift	
Function	110 $N_t = 2$	(SLA)
Definition	$x' = x$ shifted left N_s places	Any mode
Description	Single-length arithmetic shift	
Function	110 $N_t = 3$	
Definition	$x' = x$ shifted left N_s places	Any mode
Description	Single-length special shift	
Function	111 $N_t = 0$	(SLC)
Definition	$x{:}' = x{:}$ shifted left N_s places	Any mode
Description	Double-length circular shift	
Function	111 $N_t = 1$	(SLL)
Definition	$x{:}' = x{:}$ shifted left N_s places	Any mode
Description	Double-length logical shift	
Function	111 $N_t = 2$	(SLA)
Definition	$x{:}' = x{:}$ shifted left N_s places	Any mode
Description	Double-length arithmetic shift	
Function	111 $N_t = 3$	
Definition	$x{:}' = x{:}$ shifted left N_s places	Any mode
Description	Double-length special shift	
Function	112 $N_t = 0$	(SRC)
Definition	$x' = x$ shifted right N_s places	Any mode
Description	Single-length circular shift	
Function	112 $N_t = 1$	(SRL)
Definition	$x' = x$ shifted right N_s places	Any mode
Description	Single-length logical shift	
Function	112 $N_t = 2$	(SRA)
Definition	$x' = x$ shifted right N_s places	Any mode
Description	Single-length arithmetic shift	
Function	112 $N_t = 3$	(SRAV)
Definition	$x' = x$ shifted right N_s places	Any mode
Description	Single-length special shift	
Function	113 $N_t = 0$	(SRC)
Definition	$x{:}' = x{:}$ shifted right N_s places	Any mode
Description	Double-length circular shift	
Function	113 $N_t = 1$	(SRL)
Definition	$x{:}' = x{:}$ shifted right N_s places	Any mode
Description	Double-length logical shift	
Function	113 $N_t = 2$	(SRA)
Definition	$x{:}' = x{:}$ shifted right N_s places	Any mode
Description	Double-length arithmetic shift	

Function	113 $N_t = 3$ (SRAV)	
Definition	$x:' = x:$ shifted right N_s places	Any mode
Description	Double-length special shift	
Function	114 (NORM)	
Definition	$x:' = x$ normalized with initial exponent $N(M)$; $x_e^{*\prime} = N_e'$	Any mode
Description	Normalize the number whose single-length argument is held in X with respect to $N(M)$.	
Function	115 (NORM)	
Definition	$x:' = x:$ normalized with initial exponent $N(M)$; $x_e^{*\prime} = N_e'$	Any mode
Function	116 (MVCH)	
Definition	(See Description)	Any mode
Description	Transfer $N(M)$ characters from the character address in X to the character address in $X + 1$	
Function	117 (SMO)	
Definition	Add $(n)q$ to the N address of the next instruction (for a definition of q, see page 29)	Any mode
Description	Supplementary modifier to next instruction	

GROUP 12

None of these instructions can set V or C. All apart from 123 clear C.

Function	120 (ANDN)	
Definition	$x' = x$ and $N(M)$	Any mode
Description	Logical *and* of x and $N(M)$	
Function	121 (ORN)	
Definition	$x' = x_v\ N(M)$	Any mode
Description	Logical *inclusive or* of x and $N(M)$	
Function	122 (ERN)	
Definition	$x' = x \not\equiv N(M)$	Any mode
Description	Logical *exclusive or* of x and $N(M)$	
Function	123 (NULL)	
Definition	(See Description)	Any mode
Description	Dummy instruction; no operation	
Function	124 (LDCT)	
Definition	$x_c' = N_e$; $x_m' = 0$	Any mode
Description	Set counter	
Function	125 (MODE)	
Definition	(See Description)	Any mode
Description	Set zero suppression mode in accordance with the state of $N(M)$	
Function	126 (MOVE)	
Definition	(See Description)	Any mode
Description	Transfer N words from address x to address x^* (Block transfer)	

Function 127 (SUM)

Definition (See Description) Any mode

Description x' = Sum of N words from address x^*

GROUP 13

The result of functions 132 to 135 is rounded and normalized. The 137 instruction will clear *FOVR* if it is set and set V instead.

Function 130 (FLOAT)

Definition $n{:}\rightarrow a$ Any mode

Description Convert $n{:}$ from fixed- to floating-point and store in A

Function 131 (FIX)

Definition $a \rightarrow n{:}$ Any mode

Description Convert a from floating- to fixed-point and store in $N(M)$ and $N(M) + 1$

Function 132 $X = 0$ or 4 (FAD)

Definition $a' = a + n{:}$ Any mode

Description Add $n{:}$ to a

Function 133 $X = 0$ (FSB)

Definition $a' = a - n{:}$ Any mode

Description Subtract $n{:}$ from a

Function 133 $X = 4$ (FSB)

Definition $a' = n{:} - a$ Any mode

Description Subtract a from $n{:}$

Function 134 $X = 0$ or 4 (FMPY)

Definition $a' = a.n{:}$ Any mode

Description Multiply a and $n{:}$ and store the result in A

Function 135 $X = 0$ (FDVD)

Definition $a' = a/n{:}$ Any mode

Description Divide a by $n{:}$

Function 135 $X = 4$ (FDVD)

Definition $a' = n{:}/a$ Any mode

Description Divide $n{:}$ by a

Function 136 $X = 0$ (LFP)

Definition $a' = n{:}$ Any mode

Description Load $n{:}$ into A

Function 136 $X = 1$ (LFPZ)

Definition $a' = 0$ Any mode

Description Clear A and *FOVR*

Function 137 $X = 0$ (SFP)

Definition $n{:}' = a$ Any mode

Description Store a in $N(M)$ and $N(M) + 1$ leaving a unchanged

Function 137 $X = 1$ (SFPZ)

Definition $n{:}' = a,\ a' = 0$ Any mode

Description	Store a in $N(M)$ and $N(M) + 1$, clear A and $FOVR$,		

GROUP 15

All these instructions clear C

Function	150	(SUSBY)	
Definition	(See Description)		Any mode
Description	Suspend program if peripheral type $N(M)$, unit X, is active		
Function	151	(REL)	
Definition	(See Description)		Any mode
Description	Release peripheral type $N(M)$, unit X		
Function	152	(DIS)	
Definition	(See Description)		Any mode
Description	Disengage peripheral type $N(M)$, unit X		
Function	154	(CONT)	
Definition	(See Description)		Any mode
Description	Read more program from peripheral type $N(M)$, unit X		
Function	155	(SUSDP)	
Definition	(See Description)		Any mode
Description	Dump program on peripheral type $N(M)$, unit X		
Function	156	(ALLOT)	
Definition	(See Description)		Any mode
Description	Allocate peripheral type $N(M)$, unit X, to the program		
Function	157	(PERI)	
Definition	(See Description)		Any mode
Description	Initiate peripheral operation on unit X according to the control area $N(M)$.		

GROUP 16

For full details of instructions 162 to 164 see page 65.

Function	160

This function uses a control word or words the format of which depends on the addressing mode. The 15AM format, given below, can be used in 22AM provided that N_a is in lower data store. The 22AM format consists of a count in the least significant six bits of Word $N(M)$ and the 22-bit start address of a message in Word $N(M) + 1$.

Function	160	$X = 0$	(SUSTY)
Definition	(See Description)		15AM or 22AM
Description	Type n_c characters from address n_a and suspend program		
Definition	(See Description)		22AM
Description	Type message defined in the control words and suspend the program		
Function	160	$X = 1$	(DISTY)
Definition	(See Description)		15AM or 22AM
Description	Type n_c characters from address n_a		
Definition	(See Description)		22AM
Description	Type message defined in the control words		

Function 160 $X = 2$ (DELTY)

Definition (See Description) 15AM or 22AM

Description Treat n_c characters from address n_a as a console directive and delete the program

Definition (See Description) 22AM

Description Treat message defined in the control words as a console directive and delete the program

Function 161 $X = 0$ (SUSWT)

Definition (See Description) Any mode

Description Type HALTED and N_a as two characters and suspend the program

Function 161 $X = 1$ (DISP)

Definition (See Description) Any mode

Description Type DISPLAY and N_a as two characters

Function 161 $X = 2$ (DEL)

Definition (See Description) Any mode

Description Type DELETED and N_a as two characters and delete the program

Function 162 (SUSMA)

Definition (See Description) Any mode

Description Conditional alteration of one word pair in common storage
If $n^* = 0$, set $n'=x$, set $n^{*'} \neq 0$ and omit next instruction

Function 163 (AUTO)

Definition (See Description) Any mode

Description Activate Member X at $N(M)$; if $N(M) = 0$ reactivate Member X or set 163 indicator

Function 164 $X = 1$ (SUSAR)

Definition (See Description) Any mode

Description De-activate the current member unless the 163 indicator is set.

Function 164 $X = 2$ (SUSIN)

Definition (See Description) Any mode

Description De-activate the current member unless the 163 indicator or the flag-setting interrupt indicator is set.

Function 164 $X = 3$ (SUSIN)

Definition (See Description) Any mode

Description De-activate the current member unless the 163 indicator, the flag-setting interrupt indicator or the priority member indicator is set

Function 164 $X = 4$ (SUSIN)

Definition (See Description) Any mode

Description As for the 164, $X = 3$ instruction, but also tell Executive to remove suspension from all members that can be re-activated currently by the Priority Member, and to remember the occurrence of this instruction for all other members

Function 165 $N(M) = 0$ (GIVE)

Definition (See Description) Any mode

Description x' = the number of days from 31/12/1899 to current day

Function 165 $N(M) = 1$ (GIVE)

Definition (See Description) Any mode

Description $x:'$ = the date in character form

Function 165 $N(M)= 2$ (GIVE)

Definition (See Description) Any mode

Description $x:'$ = the time in character form

Function 165 $N(M)= 3$ (GIVE)

Definition (See Description) Any mode

Description x' = the core store allocated to the program

Function 165 $N(M)= 4$ (GIVE)

Definition (See Description) Any mode

Description x = core store required; x' = core store allocated as a result of this instruction

Function 165 $N(M)= 5$ (GIVE)

Definition (See Description) Any mode

Description $x:'$ = details of Executive and the central processor

Function 165 $N(M)= 8$ (GIVE)

Definition (See Description) Any mode

Description x' = current setting of mode word

Function 165 $N(M)= 9$ (GIVE)

Definition (See Description) Any mode

Description x = required setting of mode word; x' = new setting of mode word

Function 166 $X= 0$ (RRQ)

Definition (See Description) Any mode

Description The program request slip is to be read into an object program area of 16 words beginning at $N(M)$

Function 166 $X= 1$ (RRQ)

Definition (See Description) Any mode

Description A new program request slip is to be taken from an object program area of 16 words beginning at $N(M)$

AVAILABILITY OF INSTRUCTIONS WITH CENTRAL PROCESSORS

In the table the following conventions are used:

H Instruction performed by hardware

E Instruction performed by extracode

* Instructions 040 to 042 and 044 to 046 performed by hardware; instructions 043 and 047 performed by extracode.

Blank Instruction not available.

The instruction set for basic processors and for those with extra facilities is given in the table. Where a processor has more than one of the facilities listed, the appropriate instruction set can be found by combining the relevant instruction sets and, where discrepancies occur, applying the following rules.

1 If the discrepancy is between hardware and extracode, then the instruction is performed by hardware.

2 If the discrepancy is between available and not available, then the instruction is available, but

(a) A processor supplied with hardswitches instead of a console typewriter will never have the 155 and 166 instructions.

(b) The 166 instruction on a 1902/03 with EX2L Executive is null.

Central Processor	Instructions																								
	000 to 037	040 to 047	050 to 056	060 to 064	066	070 to 074	076	100 to 107	110 and 112	111 and 113	114 and 115	116	117	120 to 125	126 and 127	130	131	132 to 137	150 to 154	155	156 and 157	160, 161, 165	162 to 164	166	22AM and EBM
Basic 1901 with handswitch (EX1H)	H	E	H	H		H		H	H	E				H	E				E		E	E			
Basic 1901 with typewriter	H	E	H	H		H		H	H	E				H	E				E	E	E	E		E	
1901 with E.M.U., fixed-point	H	*	H	H		H		H	H	H	H			H	E				E	E	E	E		E	
1901 with E.M.U., fixed- and floating-point	H	*	H	H		H		H	H	H	H			H	E	E	E	H	E	E	E	E		E	
1901 with floating-point extracodes	H	E	H	H		H		H	H	E	E			H	E	E	E	E	E	E	E	E		E	
Basic 1902/3	H	E	H	H		H		H	H	E				H	E				E	E	E	E		E	
Limited 1902/3 (EX2L)	H	E	H	H		H		H	H	E				H	E				E	E	E	E			
1902/3 with dualprogramming (EX2M)	H	E	H	H		H		H	H	E				H	E				E	E	E	E	E	E	
1902/3 with E.M.U., fixed-point	H	*	H	H		H		H	H	H	H			H	E				E	E	E	E		E	
1902/3 with E.M.U., fixed- and floating-point	H	*	H	H		H		H	H	H	H			H	E	E	E	H	E	E	E	E		E	
1902/3 with floating-point extracode	H	E	H	H		H		H	H	E	E			H	E	E	E	E	E	E	E	E		E	
Basic 1904	H	H	H	H		H		H	H	H	H			H	H				E	E	E	E	E	E	
1904 with floating-point extracodes	H	H	H	H		H		H	H	H	H			H	H	E	E	E	E	E	E	E	E	E	
1905	H	H	H	H		H		H	H	H	H			H	H	E	E	H	E	E	E	E	E	E	
1906	H	H	H	H	H	H		H	H	H	H	H	H	H	H				E	E	E	E	E	E	√
1906 with floating-point extracodes	H	H	H	H	H	H	E	H	H	H	H	H	H	H	H	E	E	E	E	E	E	E	E	E	√
1907	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	E	H	E	E	E	E	E	E	√
1904/6, E and F	H	H	H	H	H	H		H	H	H	H	H	H	H	H				E	E	E	E	E	E	√
1904/6, E and F with floating-point extracodes	H	H	H	H	H	H	E	H	H	H	H	H	H	H	H	E	E	E	E	E	E	E	E	E	√
1905/7, E	H	H	H	H	H	H	E	H	H	H	H	H	H	H	H	E	E	H	E	E	E	E	E	E	√
1905/7, F	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	E	H	E	E	E	E	E	E	√
Basic 1901A with handswitch (E1HS)	H	E	H	H		H		H	H	E				H	E				E		E	E			
Basic 1901A with typewriter	H	E	H	H		H		H	H	E				H	E				E	E	E	E		E	
1901A with commercial computing feature	H	H	H	H		H		H	H	H				H	E				E	E	E	E		E	
1901A with scientific computing feature	H	H	H	H		H		H	H	H	H			H	E	E	E	H	E	E	E	E		E	
1901A with floating-point extracodes	H	E	H	H		H		H	H	E	E			H	E	E	E	E	E	E	E	E		E	
Basic 1902/3, A	H	E	H	H	H	H		H	H	H		H	H	H	H				E	E	E	E		E	
1902/3, A with multiprogramming	H	E	H	H	H	H		H	H	H		H	H	H	H				E	E	E	E	E	E	
1902/3, A with commercial computing feature	H	H	H	H	H	H		H	H	H		H	H	H	H				E	E	E	E		E	
1902/3, A with scientific computing feature	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	E	E	H	E	E	E	E		E	
1902/3, A with floating-point extracodes	H	E	H	H	H	H	E	H	H	H	E	H	H	H	H	E	E	E	E	E	E	E		E	
1903A with GEORGE 3 (EXG3)	H	H	H	H	H	H		H	H	H		H	H	H	H				E	E	E	E	E	E	√
Basic 1904A	H	H	H	H	H	H		H	H	H	H	H	H	H	H				E	E	E	E	E	E	√
1904A with floating-point unit	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	E	H	E	E	E	E	E	E	√
1904A with floating-point extracodes	H	H	H	H	H	H	E	H	H	H	H	H	H	H	H	E	E	E	E	E	E	E	E	E	√
Basic 1906A	H	H	H	H	H	H	E	H	H	H	H	H	H	H	H				E	E	E	E	E	E	√
1906A with floating-point unit	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	E	H	E	E	E	E	E	E	√
1906A with floating-point extracodes	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	E	E	E	E	E	E	E	E	E	√

Figure 8 Availability of instructions with central processors

Chapter 6 Peripheral transfers

Note: The description of standard interface and hesitations given in this chapter is for the interest only of the reader. No inference should be drawn from it.

THE NATURE OF INFORMATION TRANSFERRED

Information that is transferred between the central processor and a peripheral unit is of a standard format, consisting of one or more six-bit characters that may be transferred singly or in groups of four. There are basically two kinds of information that can be transferred:

(a) control information, which enables the central processor to determine the status of the peripheral unit and to initiate desired actions.

(b) data or, in certain circumstances, program instructions that are treated as data, that form input to or output from the central processor.

The control information is not of immediate concern to the programmer, but Executive uses this information to control all peripheral units. This information is also used by Executive to detect peripheral failures; thus Executive is able to keep the operator informed of these failures either by a coded message on the switch/light panel or by a typed message on the console typewriter.

STANDARD INTERFACE

The purpose of standard interface

To the user of the 1900 Series, standard interface consists of a standard plug and socket by which a variety of peripherals can be connected to a range of central processors; but in fact it consists of a large amount of electronics in both the central processor and peripherals. The purpose of this complex of electronics is to ensure a standard method of control and transfer of information between a central processor and peripherals. To achieve this purpose several conditions must be fulfilled:

1 Since a number of peripherals connected to a central processor may be operating simultaneously, each character must be transferred to the correct peripheral.

2 Each time a transfer is requested by a peripheral, both the central processor and the peripheral concerned must know in which direction the transfer is to take place; that is, from the core store to the peripheral or from the peripheral to the core store.

3 When a transfer is to take place, the precise moment of transfer must be specified and this time must be convenient to both the central processor and the peripheral.

4 The core store address involved in a peripheral transfer must be known.

Standard interface lines

The I.C.T. Standard Interface has provision for 37 pairs of wires, each of which is called a line and given a unique name by which it is referenced. Figure 9 on page 46 is a schematic diagram of the standard interface lines and the functions of only those lines that concern this description of standard interface are given below.

THE Do AND Di LINES

All the I.C.T. Standard Interface peripherals in current use transfer six bits at a time across the interface. If it is required to transfer a 24-bit word, then four transfer operations are used. Each of these six bits has an individual line and these lines are termed the data lines. The lines for transferring data

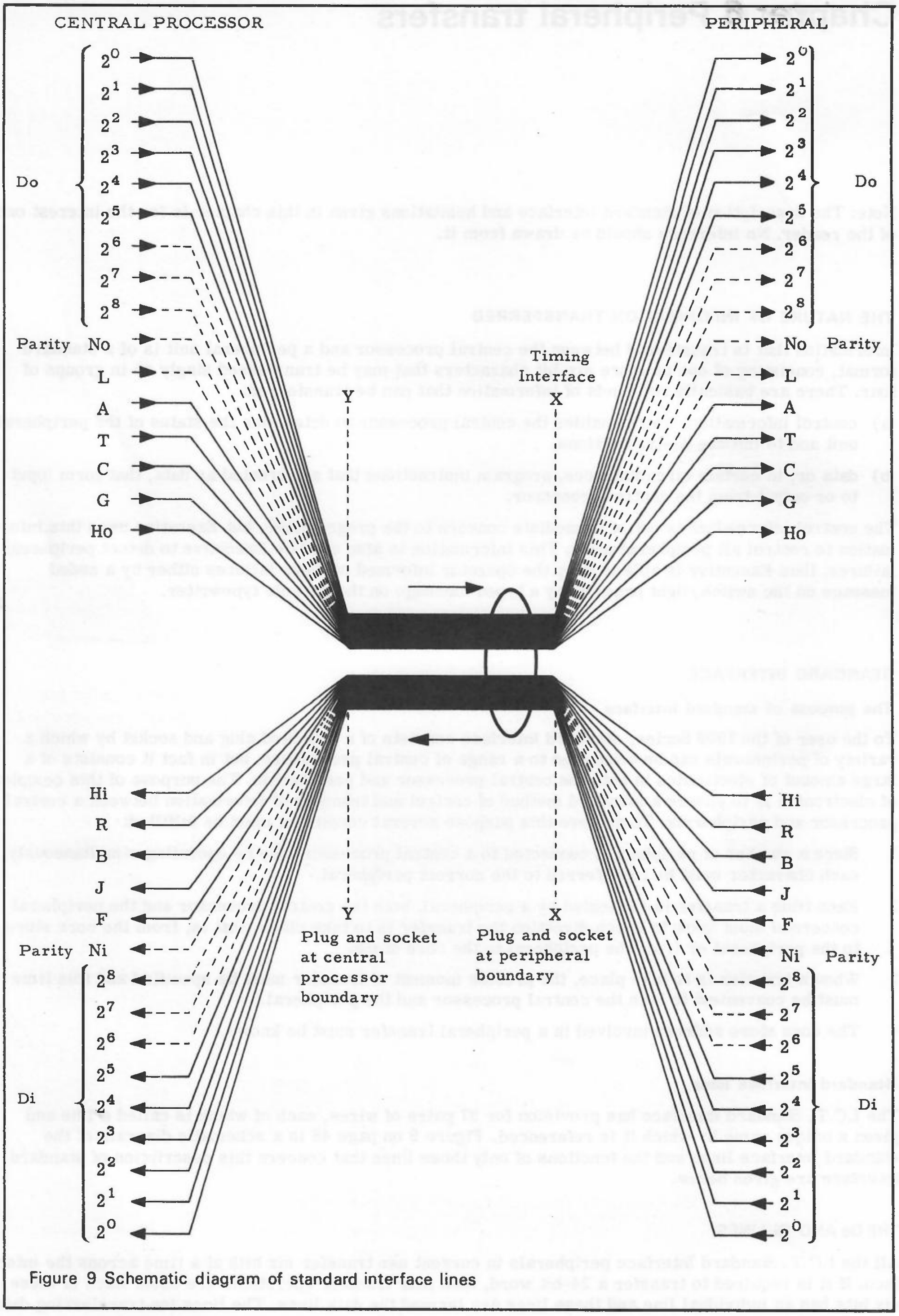


Figure 9 Schematic diagram of standard interface lines

from the central processor to the peripheral are called the *Do* (data out) lines. Similarly, the lines for transferring data from the peripheral to the central processor are called the *Di* (data in) lines.

THE C LINE

In addition to being used to transfer data, these same *Do* lines are used to transmit control commands from the central processor to the peripheral. To enable the peripheral to distinguish between a data character and a control command character, the *C* (control) line is used. When a character is transferred across the *Do* lines, the peripheral inspects the signal on the *C* line to determine whether the character is data or a control command. If the *C* line contains 0, the peripheral treats the signal on the *Do* lines as a data character; if it is 1 the peripheral treats the signal as a control command.

THE R LINE AND T LINE

The peripheral signals to the central processor that it is ready for a transfer by sending a request on the *R* (request) line. It is always the peripheral that requests a transfer, never the central processor. The actual moment that a transfer has to take place is specified by a timing pulse, which the central processor sends on the *T* (time) line. It is normally the central processor that specifies the precise moment the transfer has to take place.

THE B LINE

This line is used by the peripheral to signal to the central processor that a 'status' change has taken place that requires central processor action. The term 'status' is given to signals that indicate various states or conditions occurring in the peripheral. A signal on the *B* line is often called a *B*-interrupt. The *B* line is also used to indicate a peripheral incident that will cause an interrupt to Executive and may ultimately require operator attention. When such an interrupt occurs, Executive must ask for status information from the peripheral to determine the reason for the interrupt, for example, transfer completed, or error detected. The conditions causing a *B*-interrupt vary from peripheral to peripheral.

THE L LINE

The *L* (line) Line is used by the central processor to signal to the peripheral that the transfer has finished. It should be noted, however, that some peripherals do not make use of the function provided by the *L* line since they can be self terminating.

THE A LINE

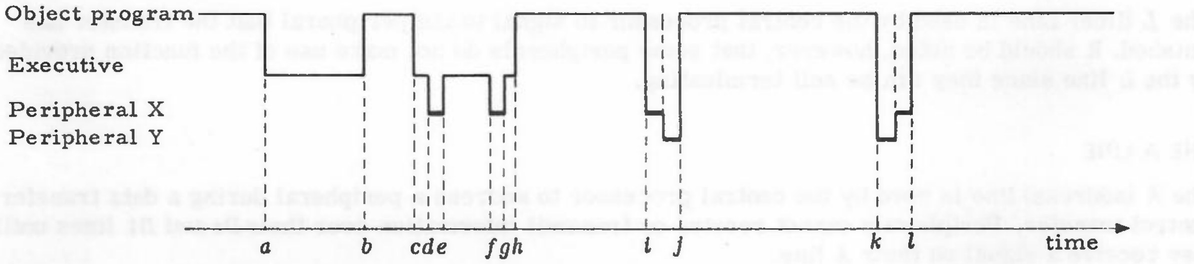
The *A* (address) line is used by the central processor to address a peripheral during a data transfer or a control transfer. Peripherals cannot receive or transmit information over their *Do* and *Di* lines until they receive a signal on their *A* line.

HESITATIONS

Certain central processors cannot deal with a peripheral transfer at the same time as they are executing program instructions. Therefore, when a peripheral requests a transfer by sending a signal on the *R* line, the central processor will hesitate, i.e. it will stop carrying out Executive or object program instructions at a convenient point and concentrate on servicing the transfer request. When the transfer of one or four characters to or from the peripheral is completed, the central processor will, providing there are no other hesitation requests, resume the execution of the program instructions. Typical hesitation times vary from 2 microseconds to 30 microseconds depending upon the central processor and type of hesitation. The control of hesitations, if there are no special peripheral connection devices, is performed by circuitry contained in the control unit.

Timesharing of the central processor for program instructions and hesitations

Figure 10 shows an example of how the central processor time is shared among the requirements demanded by two peripherals (i.e. initiation of the transfer and hesitations) and the running of an object program.



Key

1 up to *a*: The object program is running uninterrupted.

2 *a* to *b*: At point *a* the object program gives a PERI instruction for peripheral X and is suspended while Executive *initiates* the transfer for peripheral X.

3 *b* to *c*: The object program is allowed to continue.

4 *c* to *h*: At point *c* the object program gives a further PERI instruction this time for peripheral Y and is again suspended while Executive *initiates* this transfer; this initiation takes up to point *h*.

5 *d* to *e*: Executive is interrupted by a hesitation for the first transfer for peripheral X.

6 *f* to *g*: Executive is again interrupted by a hesitation for the second transfer for peripheral X.

7 *h* to *i*: The object program continues.

8 *i* to *j*: The third hesitation for peripheral X and the first for peripheral Y takes place.

9 *j* to *k*: The object program continues.

etc.

Figure 10 Example of sharing of the central processor for program instructions and hesitations

Operations performed during a hesitation

Essentially three sections of work must be performed during each hesitation.

1 The core store address to or from which information is to be transferred must be specified. This address is contained in the peripheral control word or words held on the processor side of the interface and associated with each channel. There may be one or two control words, depending on the central processor concerned. When the next hesitation occurs, a further address will be required, consequently the control word must be updated during each hesitation. Thus, the sequence is:

  (a) read the control word

  (b) note the address contained in the control word

  (c) update the control word

  (d) write the new control information back ready for the next hesitation.

2 The character or characters to be transferred between the peripheral and core store must be transmitted across the interface.

3 The information must be either read from or written to the core store.

Now that the relevant interface lines and the operations performed during a hesitation have been described, the various forms of hesitations will be considered. However, before they are, it is necessary to introduce a further component that is used during a hesitation.

Information stored in a central processor can be held either in the magnetic core store or in hardware registers. The principles of storing information in the core store are described in Chapter 4. A hardware register is an assembly of non-magnetic electronic components such as transistors, resistors and capacitors. Each bit of information is stored in a hardware register by letting current flow in one of two transistors to represent a 0-bit and letting current flow in the other of these two transistors to represent a 1-bit. Thus, a pair of transistors is required to represent the state of each bit position so that 48 transistors are required to store a 24-bit word. Registers can usually be read more quickly than a magnetic core store, but are more expensive. Consequently, in the 1900 Series central processors the majority of information is stored in the core store, although some hardware registers are used to hold words or part of words frequently addressed. Examples of such registers are the *A* and *D* registers, which are used during a high speed mode transfer.

Single-character hesitation

A peripheral requests a transfer to or from the central processor by sending a signal on the *R* line. When the central processor is ready to service this request, it sets the *A* line to 1; the *R* line is then set to 0. The character count and starting address contained in the control area associated with that peripheral are read, updated (i.e. the count is decreased and the address increased) and rewritten to the core store. During the updating of the control word information, the core store pauses; this operation is therefore called read-pause-write.

A second read-pause-write cycle is then initiated during which a six-bit data character is either read from the store during the read part of the cycle and transferred to the peripheral via the *B* register and *Do* lines, or transferred from the peripheral via the *Di* lines and *B* register and written to the store during the write part of the cycle.

Burst mode hesitation

To perform two read-pause-write operations for each six-bit character transferred across the interface, and waiting before the processor deals with the *R* request, obviously makes uneconomical use of the central processor time. Peripherals with a fast data transfer rate are therefore provided with a word buffer so that four characters may be transferred during a hesitation. The control word need then be updated only once for the transfer of four characters. However, since the data lines can transfer only six bits at a time, the four characters are transferred as four sets of six bits in a burst; consequently this type of transfer is called a burst mode transfer. Each character transferred required a *T* pulse; therefore four *T* pulses are signalled during a burst mode hesitation. The operation of this type of hesitation is as follows:

TRANSFER FROM THE CORE STORE TO A PERIPHERAL

When the buffer in the peripheral can accept four characters. the peripheral will signal on the *R* Line. When the central processor is free it will answer on the *A* line. The control word is read, updated, and rewritten to the core store. The content of the word specified by the address that was contained in the control word is then read from the core store and stored in the *B* register and also rewritten to the core store. The four characters contained in the *B* register are then transferred, one character at a time, across the *Do* lines to the peripheral.

TRANSFER FROM A PERIPHERAL TO THE CORE STORE

When the peripheral buffer contains four characters to be transferred to the central processor, the peripheral signals on the *R* line and the central processor responds in due course on the *A* line. The control word is read, updated, and rewritten to the core store. The core store word specified in the address part of the control word is read, thus zeroizing the core store word. The central processor then sends four *T* pulses, one at a time, to the peripheral; on each *T* pulse the peripheral transfers one character across the interface to the *B* register. When all the four characters are assembled in the *B* register, the content of the *B* register is transferred to the core store word specified in the control word.

On a 1904 or 1905 central processor a burst mode hesitation takes approximately 14 microseconds, whereas a single-character hesitation on these central processors takes approximately 6 microseconds. Thus, burst mode transfers reduce the central processor time used for hesitation.

High-speed mode

A burst mode hesitation on the 1902 takes 20 microseconds. Thus, if a peripheral connected to a 1902 requests a transfer at the rate of one word every 20 microseconds, then the central processor time will be completely devoted to hesitations and, consequently, there will be no time available for processing. Furthermore, if the data transfer rate exceeds one character every 5 microseconds, then the central processor will not be able to service all the requests. The 2801 and 2802 Exchangeable Disc Stores have a nominal data transfer rate of 208 kch/s and are examples of peripherals that require a transfer in excess of one character every 5 microseconds. The high-speed mode raises the maximum data rate that can be handled by the 1902 central processor, and by smaller processors in the 1900 Series.

It will be recalled that during a hesitation the core store is accessed twice, first to read, update, and rewrite the control word and secondly to access the core store word to or from which data is to be transferred. With the high-speed mode, the time taken by a hesitation is reduced because the control words* are kept in the central processor's hardware registers and not in the core store.

When the high-speed mode is used, in addition to the *B* register mentioned earlier, two further registers in the central processor, called the *A* register and the *D* register, are used. The main use of the *A* register, when the high-speed mode is not employed, is to retain an operand during instructions, and the normal use of the *D* register is to hold the current value of datum. In the high-speed mode the control words are held continuously in these two registers.

The *A* and *D* registers can hold the control words for only one peripheral at any one time; it is evident therefore, that there can be no true simultaneity between a high-speed mode transfer and any other transfer. It is also evident that, since the *D* register can no longer hold the core store datum value, there can be no simultaneity between high-speed mode and the processing of an object program. However, this restriction on simultaneity applies only during the actual transfer i.e. not during the initiation stages. Moreover, some basic peripherals need not have completed a transfer before a high-speed mode transfer can take place. Thus a buffered line printer, for instance, may be in the middle of a transfer when a high-speed mode transfer takes place. The printer transfer appears to continue uninterrupted, but in fact the printer will not be granted hesitations while the high-speed mode transfer is taking place. However, no data is lost.

Crisis times

The time between the moment a peripheral requests a hesitation and the moment the peripheral requires the hesitation to be granted, i.e. the time a peripheral can wait for a data transfer, is known as the *crisis time* of the peripheral. Peripherals that have a short crisis time are known as *time conscious.*

*Due to extended count, i.e. block lengths in excess of 128 characters, there are two control words associated with most peripherals.

With fast peripherals the crisis time is so short that the central processor may have insufficient time to complete the instruction on which it is engaged before the crisis time elapses. This will only be the case if the current instruction is one that takes a long time to execute. Accordingly, all such instructions are so organized that they may be temporarily halted while the central processor engages in a hesitation. It is essential that a peripheral be granted a hesitation within its crisis time, otherwise data may be overwritten.

Paper tape readers (and punches) have an infinite crisis time because the reader (or punch) mechanism can (does) stop the tape movement to wait for the arrival of the next character or until the central processor is ready to receive a character. Similarly, line printers that have a buffer to hold one line of print also have an infinite crisis time since a line will not be printed until the buffer is full.

Magnetic tapes, drums and disc stores have a finite crisis time because the devices cannot be stopped between characters owing to the high speed of movement of these devices.

Other peripherals, such as punched card equipment, have a medium crisis time. Although the data transfer rates to or from these devices is relatively slow compared to disc stores or fast magnetic tape systems, and the punch or read mechanisms can stop card movement momentarily, card movement cannot be stopped indefinitely in the way that paper tape movement can.

METHODS OF DECREASING HESITATION TIME

There are several devices by means of which the time required for hesitations can be reduced. These devices, described below, are available on certain 1900 central processors (see Chapters 10 to 12).

Peripheral control connector

The peripheral control connector may be specified for use on the 1904 to 1907 with burst mode peripherals only and is fitted between the core store and the standard interface connecting the peripheral concerned to the central processor.

The main advantage of the use of peripheral control connectors is that the *B* register can send or receive a word of four characters in parallel and is not held up as in normal burst mode when four characters are transferred one at a time. The store may therefore be accessed again sooner. Also, a peripheral control connector produces its own clock pulses and can therefore operate autonomously while assembling a word or sending characters to a peripheral.

A peripheral control connector is basically a further buffer or buffers, each of which can contain four six-bit characters, and is used to transfer information from or to the core store and a burst mode peripheral. Thus, the control word is updated only once for every four characters transferred. The operation of a word hesitation using a peripheral control connector is as follows:

TRANSFER FROM A PERIPHERAL TO THE CORE STORE

When the peripheral buffer contains four characters to be transferred, the peripheral signals *R* to the peripheral control connector. When the peripheral control connector is able to service this request, it sends four *T* pulses to the peripheral and on each *T* pulse transfers one character across the interface to the peripheral control connector. When the peripheral control connector has assembled these four characters into a 24-bit word, it signals the central processor that it is ready to transfer a word. The control word is then read, updated, and rewritten to the core store. The whole 24-bit word contained in the peripheral control connector is then transferred in parallel to the core store. This word hesitation takes approximately the same time as a single-character hesitation. It is important to realize that when a peripheral control connector is used, the hesitation takes place only after the peripheral control connector has signalled to the central processor that it is ready to transfer a word. Thus, while the peripheral control connector is receiving characters from a peripheral, the central processor can continue with other tasks such as servicing another peripheral or executing program instructions.

TRANSFER FROM THE CORE STORE TO A PERIPHERAL

The peripheral control connector will initially be empty. When the peripheral buffer is ready to receive a burst of four characters, the peripheral signals on the *R* line to the peripheral control connector, which then signals the request to the central processor. When the central processor is free, the word hesitation commences; the control word is read, updated, and rewritten, and a 24-bit word is transferred in parallel from the core store to the peripheral control connector.

The object program now recommences. So far as it is concerned the hesitation is over; however, the peripheral control connector still has to transfer the four characters across the interface to the peripheral buffer. On the fourth *T* pulse associated with this transfer to the peripheral, provided the count is not zero, the peripheral control connector requests a further hesitation from the central processor. This further hesitation is requested regardless of whether or not the peripheral has requested a transfer. The next word is then transferred to the peripheral control connector. However, these four characters will not be transferred to the peripheral unit the peripheral control connector receives a request on the *R* line from the peripheral. Consequently, a transfer request from a peripheral will usually be serviced immediately as the peripheral control connector will already contain the next four characters.

A burst mode hesitation takes approximately 14 microseconds on the 1904 whereas the same hesitation using a peripheral control connector takes only 6 microseconds. A peripheral control connector thus considerably reduces the central processor time used for hesitation in relation to the time used for burst mode hesitation.

In conclusion it should be noted that when a peripheral control connector is fitted, the peripheral is still connected to the central processor via the standard interface. Furthermore, it should be noted that although a peripheral control connector effectively alters a burst mode hesitation into a word hesitation, the transfer across the interface is still in burst mode in that four *T* pulses are sent to the peripheral for each four-character word transferred.

Store access control

The store access control is a hardware feature, available to handle burst mode peripherals only on the 1906 and 1907, which can be specified in conjunction with peripheral control connectors and provides a means of reducing the time that the central processor is occupied by peripheral transfers. A peripheral control connector must be fitted for each standard interface peripheral to be connected to the central processor via the store access control. There are two important aspects of the store access control.

1 When a store access control is not employed, data transferred from the central processor passes from the core store, through the central processing unit, i.e. the registers and arithmetic unit, and then to the peripheral. With a core access control the flow of data is from the core store to the peripheral without going through the central processing unit. Thus, a peripheral has direct access to the core store via a store access control. The effective advantage of this is that a peripheral can be writing to the core store via store access control, while at the same time an object program can be performing, for example, an arithmetic operation in the arithmetic unit. Thus, the central processor will hesitate only when core store access requests coincide with those of the store access control.

2 The store access control is provided with one or more registers which are used for the sole purpose of holding the peripheral control word. Consequently it is not necessary to access the core store in order to read the control word for each peripheral transfer. The store access control is provided with its own arithmetic unit for updating of the control words. A store access control can be provided with up to six of these control word registers; therefore, up to six peripherals can be handled simultaneously by one store access control. A store access control transfers 24 bits at a time and it is therefore necessary to have a 24-bit buffer, i.e. a peripheral control connector, between the store access control and each of the standard interface peripherals it is to handle.

Peripheral autonomous control

| The peripheral autonomous control is a hardware feature available on the 1904/5/6/7E, F, 1903A and 1904A processors to handle burst mode peripherals only. It is similar to the store access control used in conjunction with peripheral control connectors on the 1906/7 processors, and serves the same purpose as both these devices together. Fast magnetic tape systems, fast drums and all disc stores must be connected via a peripheral autonomous control to the processors with which this device is available. Use of the peripheral autonomous control is optional with some other peripherals.

A peripheral autonomous control contains its own mill, buffer registers to hold control words during peripheral transfers, and up to six data buffers, each containing one or two data words. One or more peripheral channels may be connected to each data buffer depending on the space available within the cabinet and the transfer rate of the peripherals concerned.

With peripheral autonomous control the flow of data is between the core store and the peripheral autonomous control without going through the central processor registers or arithmetic control unit. A peripheral therefore has direct access to core store, and hesitations occur only if core store accesses coincide, i.e. if the central processor wishes to access core store at the same time as peripheral autonomous control wishes to access store to transfer a data word.

Store access manager

The store access manager is an autonomous device contained within the central processing unit of the 1902,3/A central processors. It handles all peripheral transfers. Its function is to control the priority of access to the store so that peripherals that require access to the store urgently have it at the expense of peripherals that can afford to wait. The store access manager allows all peripherals priority over the central processing unit in access to the store.

In order that the fastest peripherals may not have to wait too long for access to the store, they may use a facility known as *early warning*. On one of its earlier *T* pulses, prior to the peripheral requesting a transfer to the store, early warning is set for that peripheral. This condition prevents any lower priority peripheral gaining access to the store before the peripheral for which early warning was set. On the 1902A the setting of early warning prevents the central processing unit from gaining access to the store for approximately three microseconds before the peripheral that set early warning requests store access. However, other peripherals may finish transfers that are currently being executed so that normally a proportion of the three microseconds will be used. Early warning will normally be used on peripherals with a data transfer rate of over 120K characters per second.

The store access manager may handle up to 8 or 12 standard interface connections on the 1902A and 1903A respectively. Each connection has a data buffer capable of holding a single character or word. The buffers are used to hold data in transit between a peripheral and the core store. A mill within the store access manager carries out the updating of the control word.

Peripheral processing unit

All peripheral transfers on the 1906A must be made via an autonomous unit known as the peripheral processing unit. The P.P.U. is similar in principle to the peripheral autonomous control but significantly faster. It has buffers and a mill to hold and update control words.

The peripheral processing unit contains a single slow peripheral control allowing a maximum of 30 standard interface channels for single character peripherals. Seven fast controls, of which up to three may be high speed controls, allow up to 19 burst mode peripheral controls to be connected.

STANDARD INTERFACE SWITCHING UNIT

The standard interface switching unit can be used in conjunction with one or more central processors to provide increased flexibility in the connection of peripherals. The principal uses of the S.I.S.U. may be summarised as follows:

1 To allow one of two similar peripherals to be connected to a single standard interface, and thus increase the available number of peripherals beyond the limit imposed by the number of standard interfaces provided.

2 Similarly, to decrease the on-line peripheral configuration to avert a crisis time problem.

3 To allow a peripheral to be switched between two processors and thereby redistribute peripherals in a dual processor system during maintenance periods or in the event of a breakdown.

4 Similarly, to redistribute peripherals in a dual processor system where the processor/peripheral requirements vary between shifts.

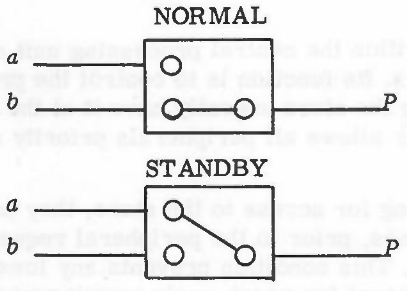
A standard interface switching unit consists of a cabinet containing up to four switching modules and a control panel. Each switching module, of which there are three types, is basically a self-contained two position switch that is operated manually. The three types of module have the numbers 7204/1, 2 or 3. The control panel allows for local individual control of each module or remote control of selected modules by a single master switch.

Types of module

MODULE TYPE 7204/1 (Y TYPE)

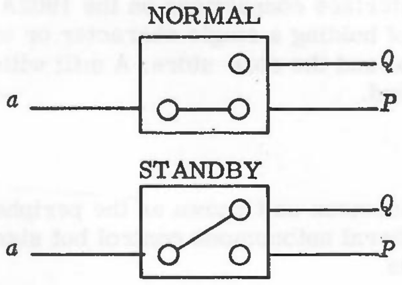
The following are four uses of module type 7204/1.

1 It is possible to switch one peripheral *P* between two processors *a* and *b*. The two processors need not be identical.



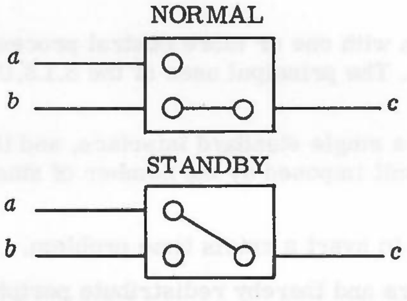
This is of use when processor *a* is used as a standby for processor *b*.

2 It is possible to switch one processor *a* between two similar peripherals *P* and *Q*. The similarity of peripherals in this context depends on the type of central processor and the interface channels fitted. For details in particular configurations reference should be made to the I.C.T. sales representative or other appropriate I.C.T. source.

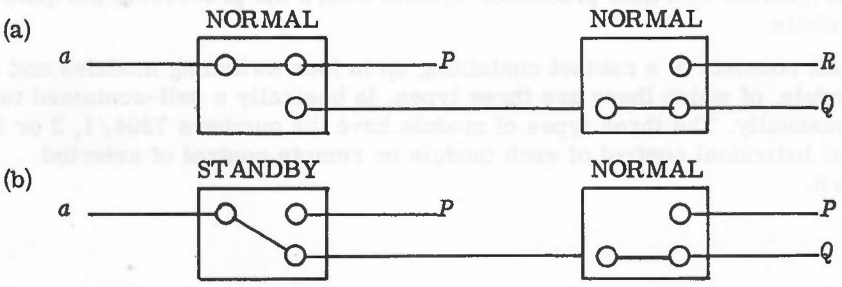


This is of use when it is required to have, say, seven peripherals with a processor which has only six standard interface sockets and where it is known that two particular peripherals must never be used together for reasons of crisis time.

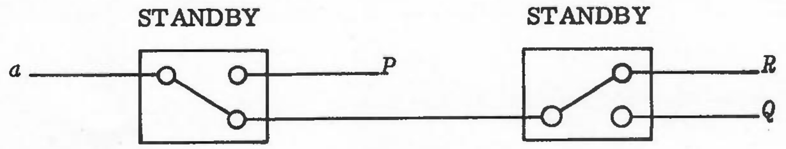
3 Where a means of connecting two processors via standard interface exists, it is possible to switch one of the two identical processors *a* and *b* to a third processor *c*.



4 It is possible to connect not more than two modules in series so that more than two similar peripherals may be made available to one interface.

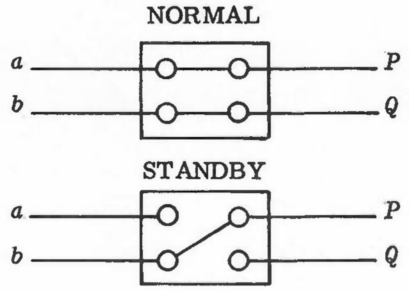


(c)



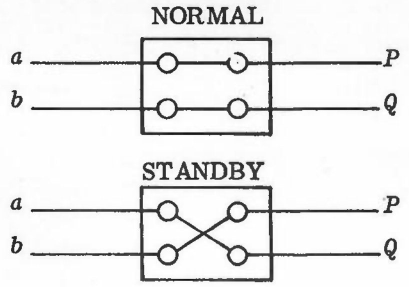
MODULE TYPE 7204/2 (N TYPE)

It is possible to switch a peripheral *P* from one processor *a* to a processor *b*, switching out at the same time a peripheral *Q* already connected to processor *b*. As far as processor *b* is concerned, peripherals *P* and *Q* must be similar.



MODULE TYPE 7204/3 (X TYPE)

It is possible to achieve interchangeability between two processors *a* and *b* and two peripherals *P* and *Q*. Peripherals *P* and *Q* must be similar to both the processors.



Chapter 7 Executive

THE NATURE OF EXECUTIVE

Executive is a supervisory program which, for all practical purposes, may be considered part of the hardware. It uses a mode of processor operation containing functions not available to object programs. These functions are not range-compatible but they always include the ability to send control signals to activate peripherals or obtain information about their status and to access any part of store unimpeded by datum and checks. Executive consists of a number of modules that can be assembled in variable packages to meet the requirements of a particular 1900 configuration. All computers in the 1900 Series use an Executive program which is always present in a protected area of store. The size of Executive varies from one installation to another depending upon the type and number of modules of which the particular Executive is composed.

THE PURPOSE AND FUNCTIONS OF EXECUTIVE

The general purpose of Executive is to take over from both programmer and operator the execution of a number of routine tasks and to organize the running of each program in the most efficient manner possible.

Executive's principal functions are:

1 Control of multiprogramming, dualprogramming, and subprogramming on the processors that have these facilities.

2 Control of peripheral devices and execution of peripheral transfer requests.

3 Loading and dumping of programs.

4 Provision of extracode facilities.

5 Communication with the operator and execution of operator directives.

THE COMPOSITION OF EXECUTIVE

The composition of the Executive used with any given configuration depends both on the central processor and the overall configuration. There are a number of different types of Executive, as listed in the section below, one or more of which is available with each central processor. Each type of Executive is available in a number of different versions according to the configuration with which it is to be used. This modularity can be further explained by reference to the functions of Executive listed above.

Not all Executives include multiprogramming facilities but all versions of a given type of Executive will be constant in this respect. That is, all versions of E4BM include multiprogramming facilities, no version of EX1H does. All Executives that include multiprogramming facilities also include subprogramming facilities.

All versions of an Executives will have routines for the control of peripheral activity, since all configurations have peripherals. However, the particular routines incorporated will depend on the peripherals used and may vary from one type of Executive to another and from one version to another.

All types of Executive and all versions include routines to control the loading of programs; routines to control dumping are included if there is a dump peripheral in the configuration. The routines may vary from one version of Executive to another depending on the peripherals that may be concerned in loading and dumping.

All types of Executive and all versions include some extracodes, since some instructions are always carried out by extracode. The number of extracodes incorporated depends upon the central processor and any optional feature that it has. For example the 165 (GIVE) instruction will always be carried out by extracode; a 1901 without an E.M.U. will have an Executive that includes an extracode to perform the 042 (MPA) instruction, but the Executive of a 1901 with an E.M.U. will not include this extracode.

All versions of any type of Executive will include some routines for operator communication. The precise nature of these routines depends on the type of Executive concerned. All versions of one type of Executive will have the same routines.

TYPES OF EXECUTIVE

The following table gives a list of the types of Executive, a brief description of each and details of availability.

Executive type code	Central processor(s)	Description of Executive	Configuration restrictions
EX1H	1901	Single program, handswitch controlled.	Limited range of peripherals allowed.
EX1T	1901	Single program, typewriter controlled; Automatic Operator feature optional.	Minimum 8K store, console typewriter.
EX1V	1901	Single program, typewriter controlled, partially stored on E.D.S.; Automatic Operator feature standard.	Minimum 8K store, console typewriter, 2 E.D.S. drives.
E1HS	1901A	Single program, handswitch controlled.	Limited range of peripherals allowed.
E1TS	1901A	Single program, typewriter controlled; Automatic Operator feature optional.	Minimum 8K store, console typewriter.
E1DS	1901A	Single program, typewriter controlled, partially stored on E.D.S.; Automatic Operator feature standard.	Minimum 8K store, console typewriter, 2 E.D.S. drives.
E1MS	1901A	Single program, typewriter controlled, partially stored on T.E.D.S.; Automatic Operator feature standard.	Minimum 8K store, console typewriter, 2 T.E.D.S. drives.
EX2L	1902	Single program, simplified facilities.	Only 4K stores, basic peripherals.
EX2M	1902/3	Dualprogram, trusted program feature optional.	Minimum 16K store.
EX2S	1902/3	Single program; Automatic Operator feature optional.	Minimum 8K store.
EX2V	1902/3	Single program, partially stored on E.D.S.; Automatic Operator feature standard.	Minimum 8K store, at least 2 E.D.S. drives or 1 drive and industry compatible magnetic tape.
E3TS	1902A	Single program; Automatic Operator feature standard.	Minimum 8K store, 4 tape decks.
E3TE	1902A/3A	Single program, trusted program feature standard.	Minimum 16K store, 4 tape decks.
E3TM	1902A/3A	Multiprogram, trusted program feature standard.	Minimum 16K store, 6 tape decks, C.C.F.

Executive type code	Central processor(s)	Description of Executive	Configuration restrictions
E3DS	1902A/3A	Single program, partially stored on E.D.S.; Automatic Operator feature standard, trusted program feature included amongst overlays stored on E.D.S.	Minimum 8K store.
E3DM	1902A/3A	Multiprogram, trusted program feature standard.	Minimum 16K store, C.C.F., disc systems.
E3DG	1903A	For use with GEORGE 3.	As determined by GEORGE 3.
E4BM	1904/5	Multiprogram, manually operated.	
E4G3	1904/5	For use with GEORGE 3.	As determined by GEORGE 3.
E6BM	1904E/F 1905E/F 1906/7 1904A	Multiprogram, manually operated.	
EDG3	1906E/F 1907E/F	For use with GEORGE 3D.	As determined by GEORGE 3D.
E6G3	1904E/F 1905E/F 1906/7 1904A 1906A	For use with GEORGE 3.	As determined by GEORGE 3.
E6G4	1906A	For use with GEORGE 4.	As determined by GEORGE 4.

ENTRY TO EXECUTIVE

An event which causes an object program to be left and Executive entered is termed an entry or interrupt, of which there are two principal types, voluntary and involuntary. A voluntary entry is one which occurs at a set predictable point in the program; an example is a program instruction that is actually executed by an extracode. An involuntary entry is one which is caused by an occurrence that cannot be readily predicted by the programmer; an involuntary entry may be caused by, for instance, operator action or the end of a peripheral transfer.

MULTIPROGRAMMING

Multiprogramming is the term given to the concurrent processing of more than one program that is made possible by facilities provided with some Executives. When a program being run under the control of a multiprogramming Executive causes an interrupt, for instance to request a peripheral transfer, Executive activates the program with the highest priority that is waiting to use the central processing unit. Executive does this each time it is interrupted, leaving all other programs in the central processor in a state of temporary suspension.

Program protection

Executive holds datum and limit values for each program held in store. Before each store access is made a check to ensure that Datum ≤ Location Addressed < Limit is effected dynamically by hardware. Thus complete protection of each program is guaranteed.

Program priorities

Each program is loaded with a priority in the range 01 to 99*, 01 being the lowest priority. Executive tries to enter the highest priority program whenever possible; if two programs have the same priority the choice depends on the circumstances in which the program were set up.

It is usual to allot higher priorities to programs that are peripheral limited. Such programs will continually interrupt Executive and thus allow other programs use of the central processing unit. If a processor limited program is given a high priority it may monopolise use of the central processing unit to the exclusion of other programs held in store. The programs would then be run consecutively rather than concurrently.

The indiscriminate use of program priorities 50 to 99 may, with certain Executives, lead to inessential programs taking precedence over some of the slower Executive and operating system actions, with a consequent loss in efficiency. In a real time system, it is essential that priority 99 is not used by any program or program member loaded except the real time member of the on-line program.

Program priorities are disregarded when Executive has formed a queue of instructions waiting to be given to a multi-unit channel such as a magnetic tape cluster. The operations are initiated in the sequence in which the instructions are received.

Two console messages are relevant to program priorities. The REvise message allows alteration of program priorities while programs are running on the machine. The PRint message allows information on the priorities of programs already loaded to be typed out on the console typewriter.

Program deletion

When a program is deleted, either by the operator or by program instruction, the peripherals become available for another program. In order to ensure that all the spare core store is available as one continuous area when a program is deleted, Executive moves all programs in higher locations down to fill the gap, and changes the values for datum and limit accordingly. Before moving the programs, Executive completes any transfers that are in progress. This process can take several seconds.

Multiprogramming under GEORGE 3 or GEORGE 4 control

The above information applies to manually operated Executives, i.e. all except those used with the GEORGE 3 and GEORGE 4 operating systems. For details of the concurrent running of programs in GEORGE 3 and GEORGE 4 environments reference should be made to the relevant GEORGE manuals.

DUALPROGRAMMING

Dualprogramming is the term given to the limited multiprogramming facilities available on 1902/3 central processors under the control of the EX2M Executive. Two programs may be run concurrently under EX2M; subprogramming facilities are also provided. Dualprogramming is similar to multiprogramming except in the respects described below.

Program protection

Executive stores a datum value for each program but no limit value. The limit, common to both programs, is the last word in store. Before each store access is made a check to ensure that Datum ≤ Location Addressed < Common Limit is effected dynamically by hardware. In normal circumstances adequate protection is thus provided. However, since it is possible for the first program loaded to access locations in the other program's area, it would be unwise to load an unproved program first. When two programs are in store, one of these being unproved, the unproved program should always occupy the higher numbered locations to prevent the possibility of its interfering with the other program.

Complete protection is provided for store accesses resulting from a peripheral transfer however. Before initiating a peripheral operation, Executive will ensure that the transfer lies within the area of store allocated to the program initiating the transfer.

*Some programs have been issued with a priority higher than 99. These programs are not range-compatible and may be run only in the environment specified.

Program deletion

There is no program relocation either as a result of deleting the lower program in store or as a result of the 165, N(M) = 4 (GIVE) instruction. A program may be loaded into the vacant area of store between Executive and the higher program when the lower program has been deleted provided that the new program is as small as or smaller than the program in the lower part of store when the program in the higher part was loaded. If the higher program is deleted, a program can be loaded into the area of store between the last word of the lower program and the store limit. However, it should be noted that if the combined size of Executive and the first program loaded is greater than 16,128 words it will be impossible to load a second program: as 16K is the maximum value that the datum register in 1902 and 1903 processor can hold.

When a program is deleted. Executive will not disengage any slow peripheral. Such peripherals must be disengaged by a 152 (DIS) order in the program. Nor will Executive rewind any tapes; this must be done by Unload Tape or Close Tape orders issued by the program.

Console typing of the output messages arising from 160, X = 1 (DISTY) and 161, X = 1 (DISP) will not be timeshared with the initiating program. The other program will not be held up.

SUBPROGRAMMING

Subprogramming is the term given to the facility whereby parts of a program, called *members*, can time-share with each other. In many ways subprogramming is similar to multiprogramming, except that whereas programs occupy discrete areas of store protected by hardware lock-outs (but see Dualprogramming) subprogramming allows order numbers to be given to members of the program which share the program's store but follow their own sequence of instructions. The subprogramming facility is available with all Executives that have multiprogramming or dualprogramming facilities.

The purpose of subprogramming

The purpose of subprogramming, in general terms, is to enable programs to run more efficiently. One of the more obvious uses is to allow calculation to be time shared with input/output in a fairly straightforward commercial type of program where the degree of time-sharing provided by the use of double-buffering techniques and by the basic autonomy of peripheral transfers on the 1900 Series is not adequate. In such cases the use of subprogramming allows input/output routines of a more complex nature to be written. This particular use is usually relevant only in fairly modest operating environments, as the problem of obtaining the best performance from such a program is usually overcome by the off-lining of peripherals by systems such as GEORGE.

An extension of the example is the use of subprogramming in connection with real-time equipment where it is essential to answer a request for acceptance of incoming data promptly to avoid possible loss of data. Subprogramming provides the means by which one part of the program can get incoming data safely into the processor whilst another part of the program is processing previously received data. Were it not for subprogramming, the processing routine would have to look at the real-time devices very frequently and, even if it did so, the access time to service a request would be much greater than with subprogramming. An example of a program which makes use of subprogramming for such reasons is the Multiplexer Housekeeping Package.

Members of a program

A program may normally consist of a maximum of three or four members, depending on the environment concerned; however, if the priority interrupt feature (see Chapter 9) is available, there will be a member, called the Priority Member, that may be additional to this maximum.

Each member has its own priority and operates autonomously with respect to other members except in the cases of loading or dumping. In these cases Member 0 acts as a master member. At all other times any member may issue control instructions with respect to itself or any other member; i.e. any member may de-activate itself and activate any other member.

It is necessary to distinguish between two forms of suspension that may occur. A member may suspend itself by means of the appropriate subprogram control instruction or it may be suspended for some reason that is not relevant to subprogram control e.g. while awaiting the termination of a peripheral transfer. The former case is described hereafter as de-activation to avoid confusion with the latter case, which is the generally accepted meaning of the term suspension in programming 1900 Series central processors.

Information associated with each member

Note: The following information does not apply to the Priority Member.

Each member of a program has certain information that is permanently associated with it and that is stored each time the member is suspended. This information includes the contents of the floating point accumulator and its overflow indicator (FOVR), the normal overflow indicator (V), the carry indicator (C), the address of the next instruction to be obeyed, the object program modes that are under the control of a program, i.e. addressing mode, branch mode and zero suppression mode, and the accumulators. This information is stored in the first 16 words of the member's area.

The 16 words referred to above contain for each member the same information as is normally held for a program in the first sixteen words of store (see page 11). The first 16 words of each member's area are distinct and reserved for use by that member. These words are stored consecutively for each member from location 32 onwards of the program area. Thus:

Words 32 to 47	Words 0 to 15 of Member 0
Words 48 to 63	Words 0 to 15 of Member 1
Words 64 to 79	Words 0 to 15 of Member 2
Words 80 to 95	Words 0 to 15 of Member 3

It should be noted that the directive GO AT renders the contents of Words 0 to 15 indeterminate for all members. The rest of the program area is available to all members, so that each member's area consists of its own first 16 words plus the rest of the program area minus the other members' first 16 words and any other reserved areas.

MEMORY INDICATORS

Each member other than the Priority Member has associated with it two memory indicators, M and P, and where a Priority Member exists, a further indicator E. The Priority Member has only a P indicator. These indicators are used to remember attempts to activate a member while it is already active. The indicators each consist of a single bit that is set if an attempted activation is to be remembered; subsequent attempts to activate the member before the memory indicator has been cleared will be forgotten.

M is the indicator set by a 163, N(M) = 0 (AUTO) instruction issued in respect of a member that is active. It must be remembered that a member may be active but suspended. The M indicator remains set until cleared by a 164 (SUSAR or SUSIN) instruction.

P is the indicator set when an event occurs, while the member is active, on a direct response device that the member controls. An event is said to occur on a flag-setting direct response device if the reply word for the device changes from "transfer in progress" to "transfer completed" or if a condition requiring object program action occurs. An event is said to occur on a suspension device operating in direct response mode if the former condition above occurs or if a device that was disengaged becomes engaged. A member is said to control a device if it is the member from which the most recent non-discrete instruction was accepted for that device. A non-discrete instruction is one that cannot be assumed to have been completed at the time that execution of the following instruction begins. If there was no such instruction, the controlling member is the one that established the device's flag area or caused the device to be switched to direct response mode.

The P indicator remains set until cleared by a 164, X = 2 or 3 (SUSIN) instruction.

E is the indicator set for all members that are active when the Priority Member issues a 164, X = 4 (SUSIN) instruction. The E indicator remains set until cleared by a 164, X = 3 (SUSIN) instruction.

Information associated with the Priority Member

Executive stores none of the information mentioned above for the Priority Member apart from the P memory indicator.

The Priority Member always operates in 15AM and DBM so there is no need to store its address and branch mode setting. The Priority Member may use the 125 (MODE) instruction but on initial activation and after a 164 (SUSIN) instruction the state of its zero suppression mode is undefined until it has issued a 125 instruction or suitable 047 (CBD) instruction.

In no circumstances may the Priority Member use Words 8 to 15. It may use the accumulators provided that the contents of those used are preserved before use and restored after use: i.e. effectively, preserved immediately after the first activation and every 164 instruction and restored immediately before each 164 instruction.

Violation of this rule may cause corruption of other members of the program but not of another program or Executive.

Further notes on the Priority Member

PRIORITY

The Priority Member, which must always be Member 5 of a program, has absolute priority; i.e. higher even than Executive. The program request block must show Member 5 as having a priority of octal 7777, i.e. Word 12 of the request block must contain octal 77777705.

TIME-OUT FEATURE

Because of the absolute priority of the Priority Member there is a danger that this member might monopolise use of the central processor. To avoid this danger, a time-out feature is used. The time-out feature causes the Priority Member to be regarded as illegal if it is continuously active for longer than a certain period of time, the exact period varying from one central processor to another.

ORDER CODE RESTRICTIONS

Certain order code restrictions apply to the Priority Member. The following instructions may be used in accordance with their range-compatible defintions (see Chapter 5): 000 to 037, 050 to 064, 070 to 074, 100, 110, 112, 120 to 125. The following instructions may be used in accordance with their range-compatible definitions provided they are available to other members in the environment concerned and provided they are performed by hardware and not by extracode: 040 to 047, 066, 111, 113, 116, 126, 127. The only other instructions available to the Priority Member are certain subprogram control instructions defined in the relevant section below, and its peripheral control instructions which will be peculiar to the processor and/or peripheral device.

Note: Only one Priority Member may be present in the central processor at any one time.

Further notes on other members

PRIORITIES

The priority assigned to each member is that supplied in the request block; the number of each member in no way affects the member's priority. It should be noted that the priorities of members of a program can be changed by means external to the program without the program being aware of the change; accordingly, a program must not be logically dependent upon the priorities of its members.

ADDRESS AND BRANCH MODES

The initial mode setting of Member 0 is determined by the supplementary request block. Members other than Member 0 obtain their initial mode setting from the mode of the member that first activates them. When a multi-member program is dumped, the mode setting of Member 0 only is recorded. On subsequent reloading, all members will again take their initial setting from the member that activates then initially.

Loading and dumping

LOADING

When a program is first loaded Member 0 will be active and suspended awaiting operator action. All other members will be inactive awaiting activation by a 163, N(M) ≠ 0 (AUTO) instruction.

If a Priority Member exists, before any servicing of priority devices can take place the Priority Member must be activated by one of the other members of the program by means of a 163, N(M) ≠ (AUTO) instruction. This initial activation causes interrupts from priority devices to be significant and they will continue to be so until either the program is deleted or the Priority Member times out. On the first activation the Priority Member should perform any appropriate initialization procedures and then suspend itself awaiting an interrupt from a priority device or re-activation by another member.

Member should perform any appropriate initialization procedures and then suspend itself awaiting an interrupt from a priority device or re-activation by another member.

DUMPING

Orders 154 (CONT) and 155 (SUSDP) are illegal if issued by any member other than Member 0. All members of a program are suspended while either of these two instructions is being carried out so that Words 0 to 15 of all members other than the Priority Member will be in their respective storage areas and thus will be dumped correctly for subsequent reload.

In the case of an operator initiated dump, it is the responsibility of the operator to ensure that all members are suspended before the dump is initiated. The members may be suspended by the use of a SUspend directive in respect of each member.

Reference to common storage areas

PROGRAMS WITHOUT A PRIORITY MEMBER

As stated earlier, Words 0 to 15 of each member are protected from corruption; the rest of the program area is common to all members and it is therefore possible for one member to corrupt another. This means that where an area of store is to be used by more than one member the program must include appropriate lock-out routines. In simple cases lock-out can be performed by using 163 and 164 instructions; in more complex cases it has to be performed by means of indicators that record the state of the common areas. In the latter cases no assumption can be made as to the relative priorities of members, since these are not under members' control. It is always possible that another member's instructions may be obeyed between the obeying of any pair of successive instructions in a member's routine, so that the programmer must ensure that data being processed by one member is always protected from interference by other subprograms.

A consequence of this last point is that the alteration of an indicator that is also altered by another member, and whose altered value is in some way dependent upon its original value, must be carried out by a single instruction that alters the quantity directly in its commonly accessed location. If more than one instruction is used to alter an indicator, it is possible for two members to be altering the same indicator at the same time, with indeterminate results. The 162 (SUSMA) instruction is provided to help overcome this problem, since SUSMA causes Executive to be entered, so that no other member can interrupt.

Two further important consequences are as follows:

1 If, under certain conditions, an indicator is set by a member and is nowhere re-set by that member, then, if another member determines that the indicator is set, it can assume that the setting conditions existed in the first member. However, if it detects that the indicator is not set, it cannot assume that the setting conditions did not exist.

2 Only when a routine in a program is pure may it be obeyed by more than one member of the program; this applies particularly to subroutines. In this context, a routine is deemed to be pure if the only words of the program area that it attempts to change lie in the range 0 to 15 inclusive or are reserved for that member and are accessed by use of a modifier. The area used for dumping Words 0 to 15 of the member's area must not be used by a pure routine.

PROGRAMS WITH A PRIORITY MEMBER

The above information applies equally to programs with a Priority Member except where contradicted below.

Non-priority members passing information such as data addresses that may be used as modifiers to the Priority Member must remember that the Priority Member always operates in 15AM and DBM and must ensure that such data is compatible with this restriction. Provided that correct use is made of the 162 (SUSMA) instruction for the manipulation of flags between non-priority members, there is no restriction on the number of the non-priority members that may share one area with the Priority Member.

Despite the fact that a 162 instruction issued by a non-priority member can be interrupted by the activation of the Priority Member, the 162 instruction may still be used by a non-priority member to manipulate flags between itself and the Priority Member. The Priority Member does not need, and indeed cannot use, the 162 instruction. Because of the absolute priority of the Priority Member, it can safely alter a flag or common area, without any risk of interference, by means of a sequence of several instructions provided that the sequence does not include any 164 (SUSIN) instructions.

Subprogramming control instructions

The instructions provided for the control of communication between members of a program are the 162 (SUSMA), 163 (AUTO) and 164 (SUSAR or SUSIN) instructions. The effect of each is defined below and the use of these instructions is described in a later section. The states of members mentioned below may be clarified by reference to the States of members in the next section.

THE 162 (SUSMA) INSTRUCTION

The action of this instruction depends on the contents of Word N(M)+1, as follows:

If the contents of Word N(M)+1 are non-zero, the program continues at the instruction in the word following that which contains the 162 instruction.

If the contents of Word N(M)+1 are zero, they are made non-zero and the contents of X are copied into Word N(M). The program then continues at the instruction contained in the second word after that which contains the 162 instruction.

Restrictions

1 N(M) must not be in a reserved area of store.

2 This instruction cannot be used by the Priority Member.

THE 163 (AUTO) INSTRUCTION

This instruction takes two forms. The first, in which N(M) is non-zero, is provided for the initial activation of a member after the program is loaded; the second, in which N(M) is zero, is provided for subsequent reactivation of a member.

Initial activation

X contains the number of the member to be activated, the first instruction to be obeyed being that in word N(M). Activation will cause Member X to assume the same address and branch modes are are applicable to the member that issues the 163 instruction at the time that the instruction is issued. The state of the zero suppression mode in Member X will be indeterminate.

Restrictions applicable to this form of the instruction are:

1 N(M) must not be in a reserved area of store

2 The instruction may be obeyed only when Member X is inactive in state SL: i.e. in the state assumed by members other than Member 0 as a result of initial loading or the GO$_\nabla$AT directive.

3 Member X can never be Member 0 or the member that issues the 163 instruction.

Subsequent re-activations

X contains the number of the member to be re-activated and word N(M) is always zero. If Member X is currently inactive due to a 164 instructions, it is re-activated at the instruction following that 164, with the state of address, branch and zero suppression modes the same as when the 164 instruction was issued. If Member X is currently active, the memory indicator M of Member X will be set and will remain so until Member X issues a 164 instruction which will then clear the M indicator but not de-activate Member X.

Restrictions applicable to this form of the instruction are:

1 Word N(M) must be zero.

2 The instruction must not refer to a member that is inactive in state SL.

THE 164 (SUSAR OR SUSIN) INSTRUCTION

This instruction provides the means by which a member can de-activate itself until a specified type of event occurs provided that no such event has occurred since the previous equivalent instruction was issued by the member. It will be noted that this instruction has variants dependant upon the value of X (N(M) is always zero), and that successive variants include all preceding variants. The definition of all variants is such that spurious re-activation, i.e. re-activation of a member that should not be re-activated may occur; all programs must be coded to allow for spurious re-activation.

The 164, X=1 (SUSAR) variant

Unless the M indicator of the member issuing this instruction is set, the member is de-activated until either a 163 instruction referring to this member is issued. If the M indicator is set, it is cleared and the member proceeds at the next instruction.

This variant is not available to the Priority Member.

The 164, X=2 (SUSIN) variant

Unless the M or P indicators of the member issuing this instruction are set, the member is de-activated until either a 163 instruction referring to this member or an event on a direct response device controlled by this member occurs. If either or both of the M and P indicators are set, they are then cleared and the member proceeds at the next instruction.

This variant is not available to the Priority Member.

The 164, X=3 (SUSIN) variant

This variant is available only to a program that includes a Priority Member.

Unless the M, P, or E indicators of the member issuing this instruction are set, the member is de-activated until one of the following occurrences:

1 A 163 instruction referring to this member is issued.

2 An event on a direct response device controlled by this member occurs.

3 A 164, X = 4 instruction is issued.

If any or all of the indicators are set, they are then cleared and the member proceeds at the next instruction.

The 164, X=4 (SUSIN) variant

This variant is available only to the Priority Member. If there have been no interrupts from priority devices since the Priority Member was last activated, the Priority Member is de-activated until either such an interrupt occurs or a 163 instruction referring to the Priority Member is issued. If an interrupt from a priority device has occurred, the Priority Member proceeds at the next instruction.
Regardless of whether the Priority Member de-activates itself, all other members that are inactive in such a state that they can be re-activated by the Priority Member are re-activated, and the E memory indicator of any other members is set.

States of members

When a program is loaded Member 0 is deemed to be active, although it will probably be suspended awaiting some operator message, e.g. GO. Member 0 may then activate some other member and de-activate itself. It is important to realise that a member may be active, i.e. have current use of the central processor *as far as that program is concerned* , and yet be suspended awaiting operator action or an event such as the terminator of a peripheral transfer.

A program may be in any one of a number of inactive states. These inactive states are considered below.

STATE TRANSITION TABLES

The two diagrams below summarize the effect of various events under all possible valid conditions. The explanatory notes that follow should be read in conjunction with the diagrams.

1 The word "invalid" indicates that a restriction has been violated.

2 At any time a member is in one of the following states.

NS Active, but may be suspended

Instruction or event	Applying to member	Member being in state				
		SL	NS	SM	SMP	SMPE
163, N(M) ≠ 0	Y	Y becomes active	Invalid	Invalid	Invalid	Invalid
163, N(M) = 0	Y	Invalid	M indicator of Y set	Y becomes active	Y becomes active	Y becomes active
Direct Response Peripheral Event	Y or all members	Invalid	P indicator of Y set	P indicator of Y set	Y becomes active	Y becomes active
164, X = 4	All members	Invalid	E indicator of Y set	E indicator of Y set	E indicator of Y set	Y becomes active

Figure 11 The effect of the 163 instruction, direct response peripheral events and the 164, X=4 instruction on members' states

Memory indicators of Y that are set	Effect of member Y issuing a 164 instruction		
	164, X=1	164, X=2	164, X=3
None	Y assumes state SM	Y assumes state SMP	Y assumes state SMPE
E only	Y assumes state SM	Y assumes state SMP	E cleared, Y remains in state NS
P only	Y assumes state SM	P cleared, Y remains in state NS	P cleared, Y remains in state NS
E and P only	Y assumes state SM	P cleared, Y remains in state NS	E and P cleared, Y remains in state NS
M only	M cleared, Y remains in state NS	M cleared, Y remains in state NS	M cleared, Y remains in state NS
E and M only	M cleared, Y remains in state NS	M cleared, Y remains in state NS	E and M cleared, Y remains in state NS
M and P only	M cleared, Y remains in state NS	M and P cleared, Y remains in state NS	M and P cleared, Y remains in state NS
E, M, and P	M cleared, Y remains in state NS	M and P cleared, Y remains in state NS	E, M and P cleared, Y remains in state NS

Figure 12 The effect of the 164 instruction on memory indicators

SL In an inactive state because it has not had an initial activation since the program was loaded or since a GO AT directive.

SM Inactive because it has issued a 164, X = 1 instruction

SMP Inactive because it has issued a 164, X = 2 instruction

SMPE Inactive because it has issued a 164, X = 3 instruction

3 All references to memory indicator E and associated conditions apply only to a program that includes a Priority Member.

Examples

The following examples are provided to clarify some of the points made in the preceding description of the subprogramming system and to show the use of the subprogramming control instructions. In the examples it is assumed that subprogramming is being employed to gain efficient time-sharing of processing with input/output functions. One member is therefore engaged in processing normal records and in stacking exception records that have to be printed. Another member takes exception records from the stack and prints them. The former member will de-activate itself only when the area holding records to be printed is full; the latter member will de-activate itself when there is no record to be printed.

The notation used in the examples is as follows:

p A cyclic pointer to the area of the buffer into which to read the next record; p is local to the member reading and processing normal records

q A cyclic pointer to the area of the buffer from which to print the next record; q is local to the member printing exception records

r The number of exception records outstanding to be printed; r is common to both members.

t The capacity of the stacking buffer

The reason for interrupts occurring is not specified in the examples as they are not important. It should be noted that the sequence in which members run bears no relation to any possible priority they may have. It is likely that this apparent disregard of member's priorities would occur in an environment including GEORGE or where direct response devices are being serviced. It is for this reason that it has been stressed that a program must not be logically dependent on member's priorities.

EXAMPLE 1: WHY THERE MUST BE M INDICATORS

It might be thought that a program would use 163 and 164 instructions only where necessary and that therefore there is no need for an *M* indicator for the 163 instruction to set and the 164 instructions to test. The following example shows why the *M* indicator is necessary.

Member	Sequence of actions
0	Reads a record, sets $r = 1$, issues 163 1 0. An interrupt occurs
1	Prints the record previously read by Member 0, sets $r = 0$, tests r and finds $r = 0$. An interrupt occurs before a 164 instruction can be issued.
0	Reads next record, sets $r = 1$, issues 163 1 0. Member 1 is still active, so its *M* indicator is set. An interrupt occurs.
1	Issues a 164 instruction. The *M* indicator is set, so this member carries on.

The 163 instruction really means that if the member referred to is active its *M* indicator must be set; otherwise Executive must be informed that the next time there is an interrupt the member that was the subject of the 163 instruction must be considered for running. If the 163 instruction did not set an indicator and the 164 instruction did not test it, Member 1 in the example above would be de-activated and Member 0 would carry on and de-activate itself on filling the stacking area. This example illustrates the constant problem of subprogramming of how to make a test and act on the result without being interrupted, or if an interrupt occurs, for it not to matter. A similar condition could arise if Member 0 were interrupted just before issuing a 164 instruction. In both cases the lack of an *M* indicator could result in complete paralysis, each member being inactive awaiting activation by the other. (See also Example 3.)

EXAMPLE 2: A 163 INSTRUCTION ISSUED WHEN M IS SET

It might seem that the definition of the 163 instruction should allow for finding the M indicator set, or that this condition should never be allowed to arise. The following example illustrates not only that the tests do not prevent this happening but that the condition is irrelevant.

Member	Sequence of actions
0	Reads a record, sets $r = 1$, issues 163 1 0. An interrupt occurs.
1	Prints the record previously read by member 0, sets $r = 0$. An interrupt occurs before the member can test r.
0	Reads next record, sets $r = 1$, issues 163 1 0. Member 1 is still active so its M indicator is set. An interrupt occurs.
1	Tests r and finds $r = 1$, prints record previously read by Member 0, sets $r = 0$. An interrupt occurs before the member can test r.
0	Reads next record, sets $r = 1$, issues 163 1 0. Member 1 is still active so its M indicator is set again.

Thus the tests have permitted a spurious 163 instruction because they could not detect that Member 1 was active. The first example illustrated activating a member that was virtually inactive since it had tested r and was about to de-activate itself. This example illustrates activating a truly active member, but no error arises provided that spurious re-activation is not caused.

EXAMPLE 3: COMPLETE PARALYSIS

Example 1 showed that complete paralysis could occur if M indicators were not provided. However, even with the provision of these indicators, complete paralysis can still occur if care is not taken. Programmers using subprogramming for the first time tend to think that since M indicators cannot be tested explicitly to see if a member is active, switches should be defined corresponding to the M indicator of each member. In the example below Member 0 has a switch S and Member 1 a switch T, these switches being set when the members are active and unset before the members de-activate themselves. Each member tests the other's switch and issues a 163 instruction only if the other member's switch is set, i.e. only if the other member is inactive. In this way unnecessary 163 instructions can supposedly be avoided.

Member	Sequence of actions
0	Reads a record, sets $r = 1$, tests T and finds it set, issues 163 1 0. An interrupt occurs
1	Sets T, prints the record previously read by Member 0, sets $r = 0$, tests r and finds $r = 0$. An interrupt occurs before the member can unset T
0	Reads next record, sets $r = 1$, reads another record, sets $r = 2$ and so on until $r = t$. Unsets S and issues 164 1 0.
1	Unsets T, and issues 164 1 0.

Both members are now inactive awaiting re-activation by the other. This example again illustrates the problem of how to make a test and act on it without being interrupted, or if an interrupt occurs, for it not to matter. The complete paralysis obtained above could be avoided if S and T were unset by their respective members before instead of after testing r. S and T are in fact superfluous and can be replaced by further tests on r.

EXAMPLE 4: SPURIOUS RE-ACTIVATION

Spurious re-activation, it will be remembered, occurs where a member is re-activated when it should not be (as against need not be). This condition is illustrated below.

Member	Sequence of actions
0	Reads a record, sets $r = 1$, issues 163 1 0. An interrupt occurs
1	Prints the record previously read by Member 0, sets $r = 0$. An interrupt occurs before r can be tested.

Member Sequence actions

0 Reads next record, sets $r = 1$, issues 163 1 0. Member 1 is active so its M indicator is set. An interrupt occurs

1 Since $r = 1$, prints the record previously read by Member 0, sets $r = 0$, issues 164 1 0. M is set, so the member carries on.

Since $r = 0$, Member 1 should not carry on. To avoid such cases of spurious re-activation it is necessary to loop back and test r again after the 164 instruction before going on to read or print the next record.

EXAMPLE 5: THE USE OF THE 162 INSTRUCTION

The previous examples have used only 163 and 164 instructions. It is necessary to use the 162 instruction only:

1 When it is required to guard against all future machine contingencies

2 To update a parameter in a way that cannot be achieved in one interruptable instruction

3 In a program in which two or more members are updating a parameter to prime another member that both call.

The example below illustrates the last case.

In this example it is assumed that two files have to be processed and that in each case exception records have to be printed out. It is further assumed that the printed records will be dealt with individually and can therefore be printed in no particular sequence, records from one file being intermingled with those from the other. In the illustration, Members 0 and 2 are reading into buffer areas $R1$ and $R2$ respectively and stacking exception records to be printed by a third member, Member 1. The symbols used are as previously defined.

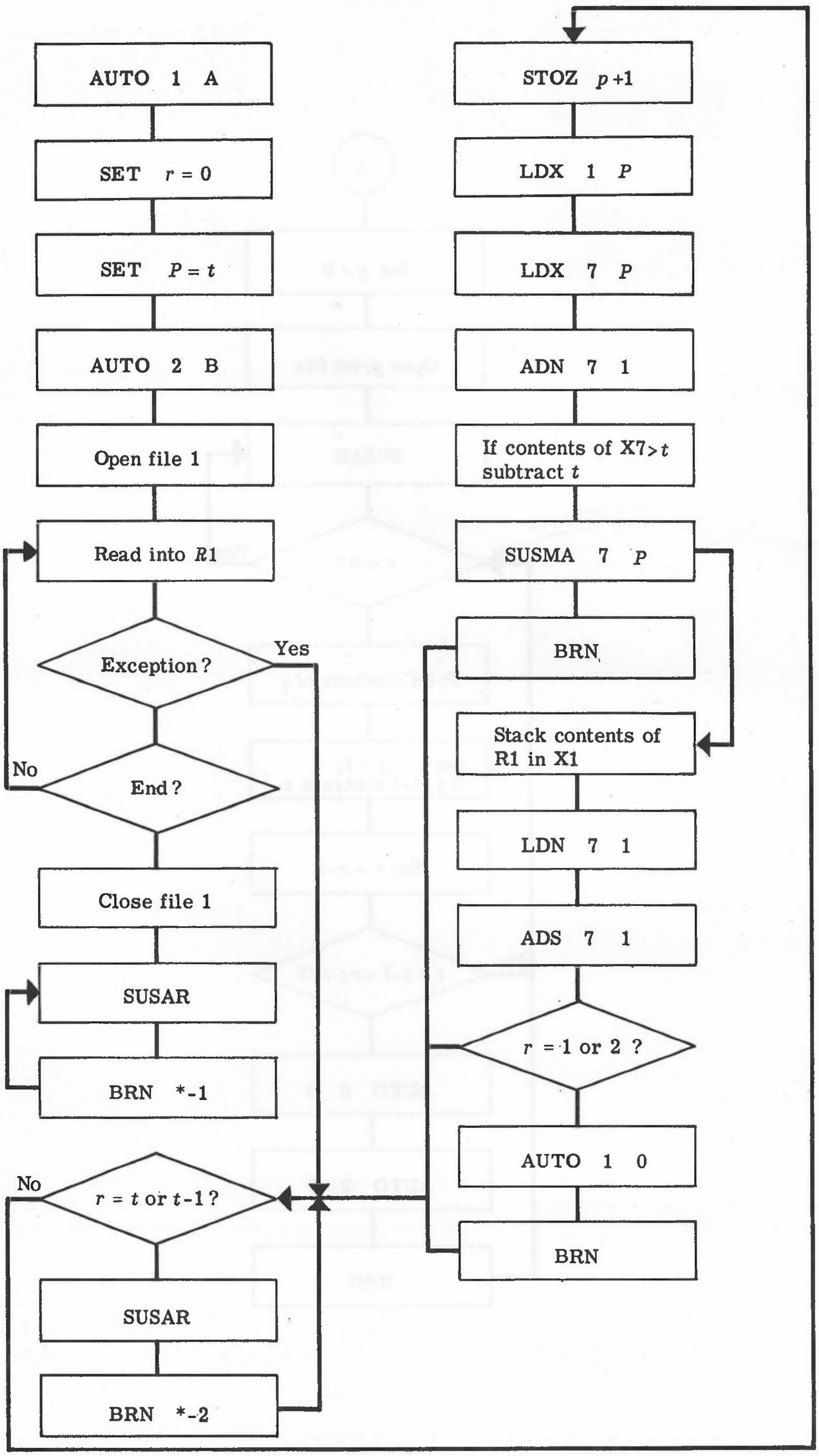
Having two stacking members considerably alters the programming method adopted. To obtain the most efficient utilization of the store the stacking area should be shared by Members 0 and 2; p therefore becomes a common parameter and a little thought will show that it is not viable to simply read into p and set $p = p + 1$ in both members.

There is the possibility of one member reading into p and then being interrupted, whereupon the other stacking member might also read into p. To avoid this possibility p must be updated by means of the 162 instruction. It would be possible to use the 162 instruction to lock out the stack whilst any given member was accessing it. However, this procedure would give rise to unnecessary 163 and 164 instructions. By using the 162 instruction as in the example given, it is possible to increment p before reading into the stack and thus safely reserve an area and then use it.

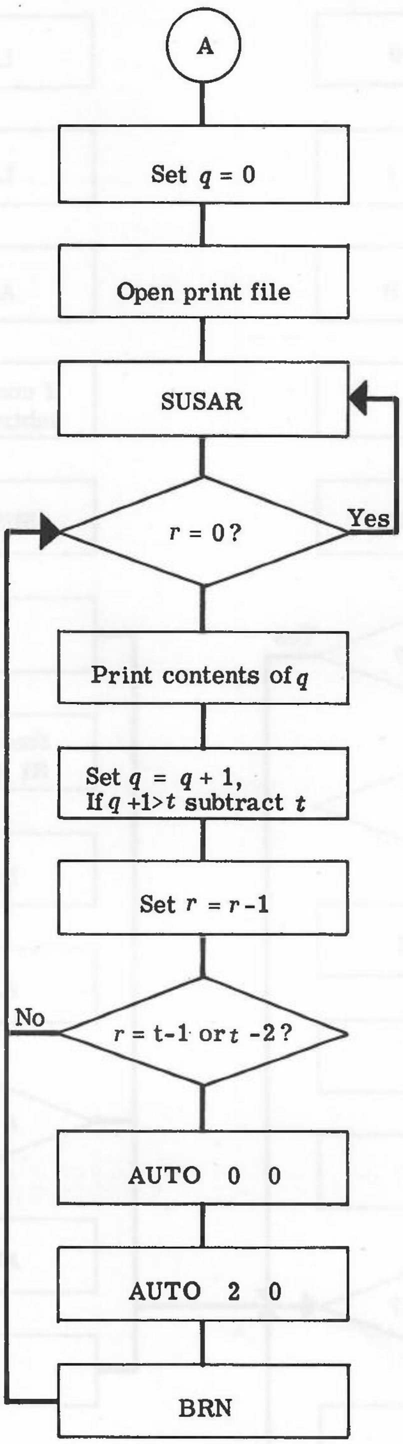
Although r is a common parameter in the examples previously given, problems arise when there is more than one member incrementing r; in previous examples one member incremented r and the other reduced it. Unlike p, r can be updated in one interruptable instruction, the 011 (ADS) instruction. The problem is how to test whether the stack is full and whether Member 1, the printing member, might need a 163 instruction. Considering the former half of the problem, if a test for $r = t$ is made and both members update r virtually in parallel the stack could overflow. The test must therefore be for $r = t$ or $t = 1$; if the result is positive, the testing member must de-activate itself. Unnecessary de-activation could result but this is the price to be paid for ensuring against the overflow condition arising. It is not possible to avoid all unnecessary 163 and 164 instructions, although every effort should be made to reduce their occurrence because of the time taken to perform extracodes. The general rule is that whenever a situation requiring a 163 or 164 instruction may occur, then one must obey the instruction as it is never possible to determine whether or not the instruction is really necessary. With even more members sharing r, correspondingly more tests would have to be made: if a third member shared r, the test would be for $r = t$ or $t - 1$ or $t - 2$.

The second half of the problem, to determine whether the printing member might need a 163 instruction, is resolved by testing for $r = 1$ or 2. To appreciate why it is necessary to test for $r = 2$, suppose that one stacking member reads a record into the stack, sets $r = 1$ and is then interrupted before testing r. The other stacking member could then also read a record into the stack and set $r = 2$. If the test on r was simply for $r = 1$ the printing member would never be activated; the stacking members would continue until the stack was full and then de-activate themselves, thus leading to complete paralysis.

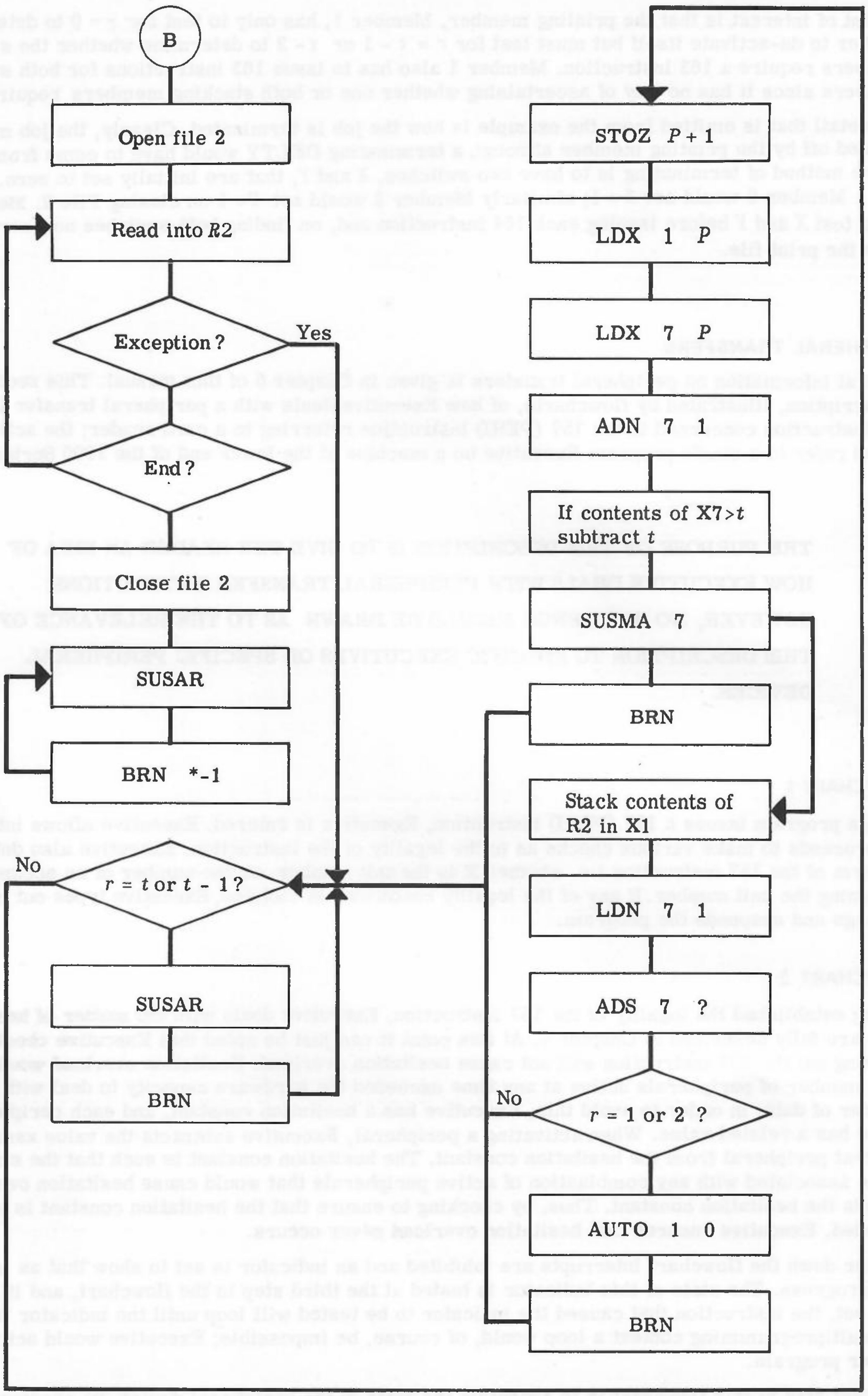
The spurious re-activation condition mentioned in previous examples is combatted by following any 164 instruction by a loop back to the test that gave rise to the 164 instruction.


AUTO 1 A
SET r = 0
SET P = t
AUTO 2 B
Open file 1
Read into R1
Exception?
Yes
No
End?
Close file 1
SUSAR
BRN *-1
No
r = t or t-1?
SUSAR
BRN *-2
STOZ p+1
LDX 1 P
LDX 7 P
ADN 7 1
If contents of X7>t subtract t
SUSMA 7 P
BRN
Stack contents of R1 in X1
LDN 7 1
ADS 7 1
r = 1 or 2 ?
AUTO 1 0
BRN

MEMBER 0


A
Set q = 0
Open print file
SUSAR
r = 0?
Yes
Print contents of q
Set q = q + 1,
If q +1>t subtract t
Set r = r -1
No
r = t-1 or t -2?
AUTO 0 0
AUTO 2 0
BRN

MEMBER 1


B
Open file 2
Read into R2
Exception ?
Yes
End ?
Close file 2
SUSAR
BRN *-1
No
r = t or t - 1 ?
SUSAR
BRN
STOZ P + 1
LDX 1 P
LDX 7 P
ADN 7 1
If contents of X7>t subtract t
SUSMA 7
BRN
Stack contents of R2 in X1
LDN 7 1
ADS 7 ?
No
r = 1 or 2 ?
AUTO 1 0
BRN

MEMBER 2

A point of interest is that the printing member, Member 1, has only to test for $r = 0$ to determine whether to de-activate itself but must test for $r = t - 1$ or $t - 2$ to determine whether the stacking members require a 163 instruction. Member 1 also has to issue 163 instructions for both stacking members since it has no way of ascertaining whether one or both stacking members require activation.

One detail that is omitted from the example is how the job is terminated. Clearly, the job must be rounded off by the printing member although a terminating DELTY would have to come from Member 0. One method of terminating is to have two switches, *X* and *Y*, that are initially set to zero. On closing File 1 Member 0 would set $X = 1$; similarly Member 2 would set $Y = 1$ on closing File 2. Member 1 would test *X* and *Y* before issuing each 164 instruction and, on finding both switches non-zero, would close the print file.

PERIPHERAL TRANSFERS

General information on peripheral transfers is given in Chapter 6 of this manual. This section gives a description, illustrated by flowcharts, of how Executive deals with a peripheral transfer instruction. The instruction concerned is the 157 (PERI) instruction referring to a card reader; the actions described refer to a single program Executive on a machine at the lower end of the 1900 Series.

THE PURPOSE OF THIS DESCRIPTION IS TO GIVE THE READER AN IDEA OF HOW EXECUTIVE DEALS WITH PERIPHERAL TRANSFER INSTRUCTIONS; HOWEVER, NO INFERENCE SHOULD BE DRAWN AS TO THE RELEVANCE OF THIS DESCRIPTION TO SPECIFIC EXECUTIVES OR SPECIFIC PERIPHERAL DEVICES.

FLOWCHART 1

When a program issues a 157 (PERI) instruction, Executive is entered. Executive allows interrupts, and proceeds to make various checks as to the legality of the instruction. Executive also determines the form of the 157 instruction i.e. whether X is the unit number, or the number of an accumulator containing the unit number. If any of the legality conditions is violated, Executive types out an ILLEGAL message and suspends the program.

FLOWCHART 2

Having established the legality of the 157 instruction, Executive deals with the matter of hesitations; these are fully described in Chapter 6. At this point it can just be noted that Executive checks that carrying out the 157 instruction will not cause hesitation overload. Hesitation overload would occur if the number of peripherals active at any time exceeded the hardware capacity to deal with all the transfer of data. In order to avoid this, Executive has a hesitation constant, and each peripheral device has a related value. When activating a peripheral, Executive subtracts the value associated with that peripheral from the hesitation constant. The hesitation constant is such that the sum of the values associated with any combination of active peripherals that would cause hesitation overload exceeds the hesitation constant. Thus, by checking to ensure that the hesitation constant is never exceeded, Executive ensures that hesitation overload never occurs.

Further down the flowchart interrupts are inhibited and an indicator is set to show that an instruction is in progress. The state of this indicator is tested at the third step in the flowchart, and if it is found to be set, the instruction that caused the indicator to be tested will loop until the indicator is unset. In a multiprogramming context a loop would, of course, be impossible; Executive would activate another program.

The final step in this flowchart is to ascertain the state of the peripheral device, by sending a 'command' to it to perform the required transfer. The reply, sent back immediately from the peripheral device, may be "Accepted", "Rejected", or "Inoperable".

FLOWCHART 3

If the command is "Accepted", Executive sets the reply word negative, subtracts the value of the peripheral from the hesitation constant, and then tests to see if this transfer is a repeat of a previous transfer. If this is the case, a branch is made to E; otherwise, Executive returns control to the object program.

If the command is not accepted, nor a repeat, a test is made to determine whether it is rejected or inoperable. If it is rejected, Executive will allow interrupts, and loop back to the main line of flow at B in Flowchart 2. If the command reply is "Inoperable", a FIX message is typed if this has not already been done, and a loop made as with the rejected command. Until a peripheral or typewriter interrupt occurs, the command is repeated indefinitely (once every 150 microseconds on a 1902).

FLOWCHART 4

If the 157 instruction is a repeat, a branch is made from the main line of flow (in Flowchart 3) to E. A test is made to see if the command has been accepted. If this is the case, Executive transfers control back to the object program. Otherwise, an error message is typed and the program that issued the 157 instruction is suspended.

FLOWCHART 5

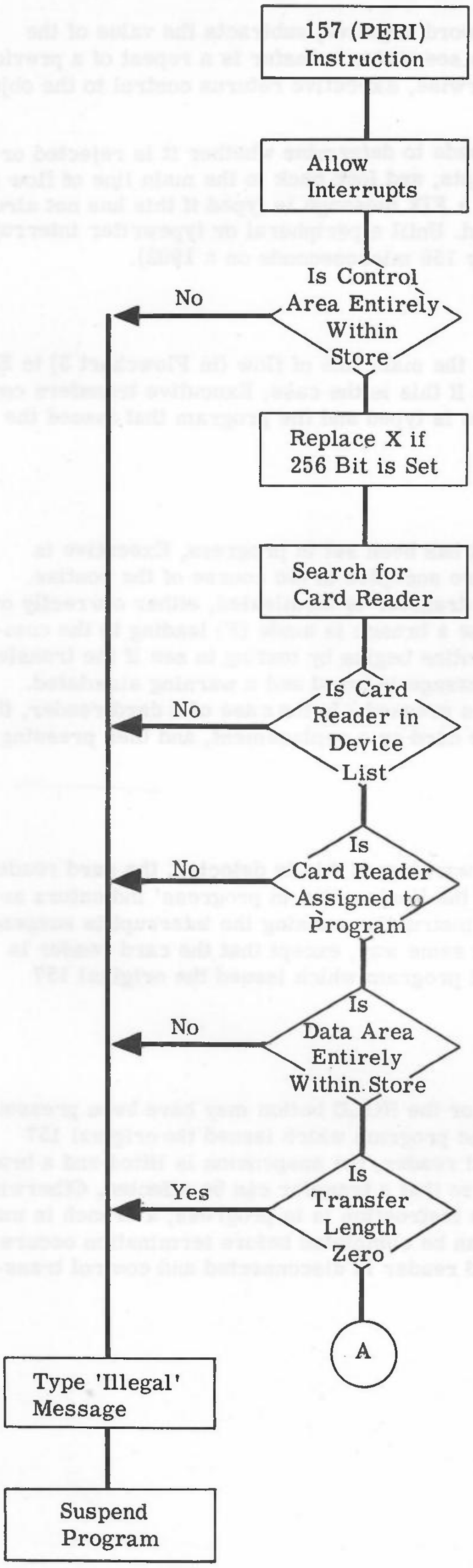
If a peripheral transfer is stopped for any reason once it has been set in progress, Executive is entered (top of Flowchart 5). In this case no interrupts are accepted in the course of the routine. Executive establishes the reason for the interrupt. If the transfer is terminated, either correctly or in error, an end-of-transfer routine is entered; otherwise a branch is made (F) leading to the completion or repeat of the transfer. The end-of-transfer routine begins by testing to see if the transfer was terminated in error. If this is the case, an error message is typed and a warning simulated. (A warning is said to have occurred if the HOLD button is pressed.) In the c ase of a card reader, the repeat process involves the operator in repositioning the card or a replacement, and then pressing the ALLOCATE button.

FLOWCHART 6

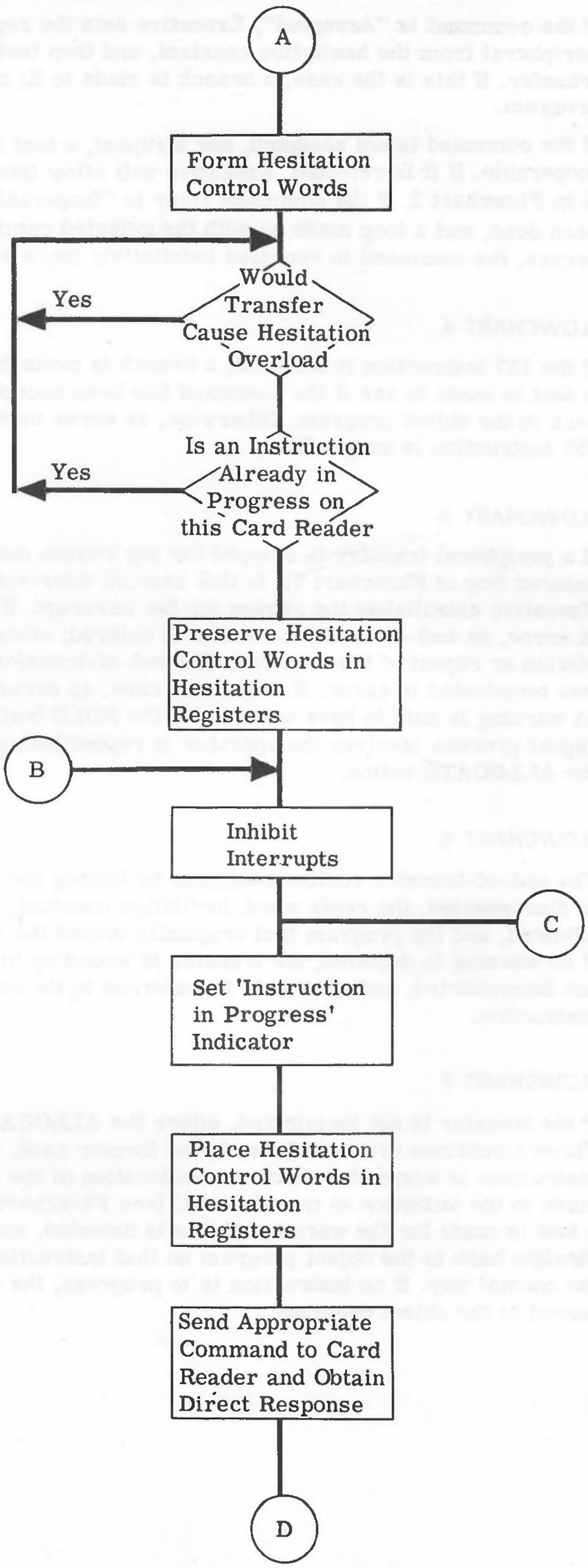
The end-of-transfer routine continues by testing for the warning; if this is detected, the card reader is disconnected, the reply word, hesitation constant, and the 'instruction in progress' indicators are adjusted, and the program that originally issued the 157 instruction causing the interrupt is suspended. If no warning is detected, the transfer is wound up in the same way, except that the card reader is not disconnected, and control is transferred to the object program which issued the original 157 instruction.

FLOWCHART 7

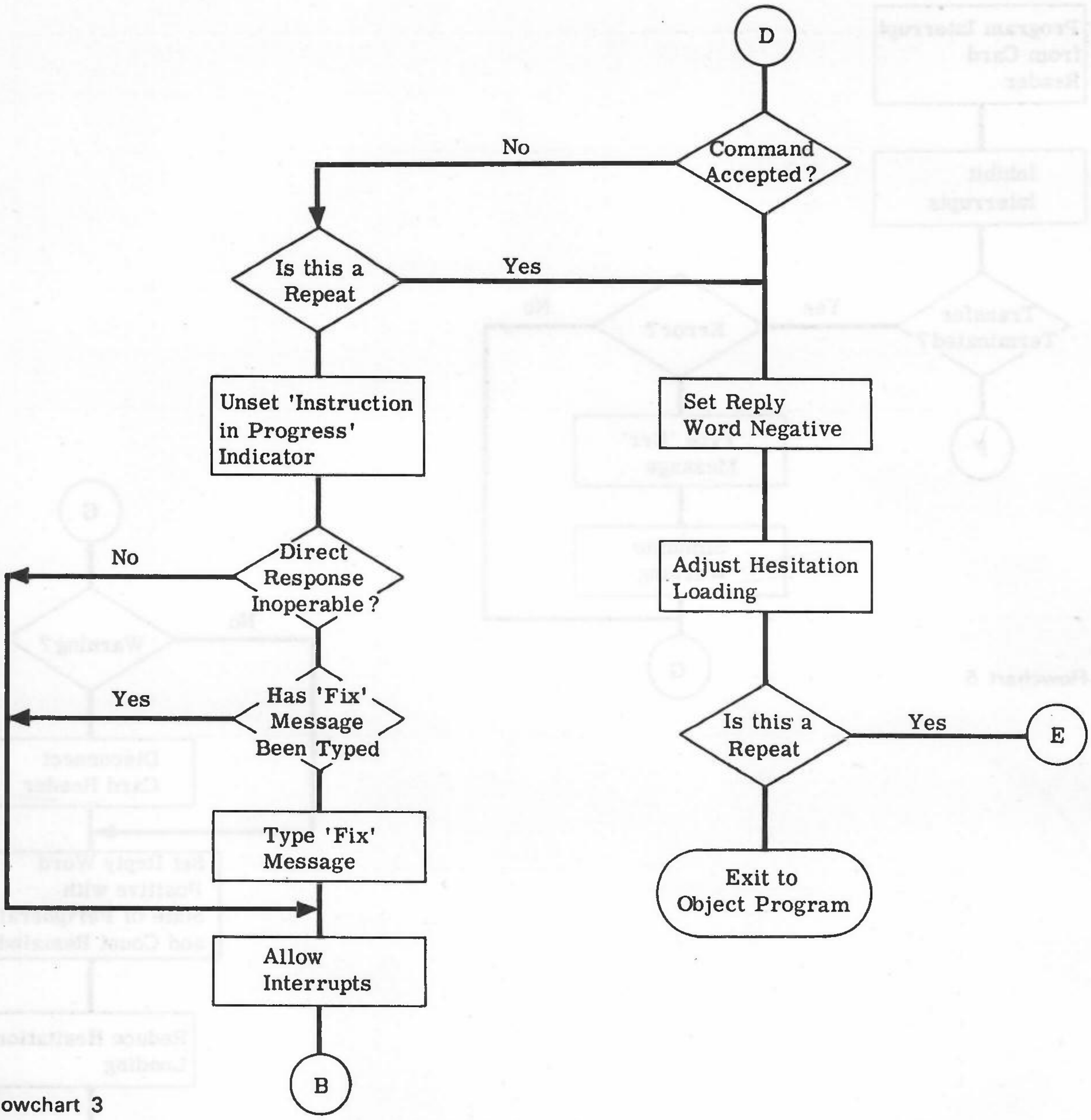
If the transfer is not terminated, either the ALLOCATE or the HOLD button may have been pressed. These conditions are tested for. In the former case, if the program which issued the original 157 instruction is suspended awaiting reallocation of the card reader, the suspension is lifted and a branch made to the initiation of transfer at C (see Flowchart 2) so that a transfer can be effected. Otherwise, a test is made for the warning. If this is detected, and an instruction is in progress, a branch is made straight back to the object program so that instruction can be completed before termination occurs in the normal way. If no instruction is in progress, the card reader is disconnected and control transferred to the object program.


157 (PERI) Instruction
Allow Interrupts
Is Control Area Entirely Within Store
No
Replace X if 256 Bit is Set
Search for Card Reader
Is Card Reader in Device List
No
Is Card Reader Assigned to Program
No
Is Data Area Entirely Within Store
No
Is Transfer Length Zero
Yes
A
Type 'Illegal' Message
Suspend Program

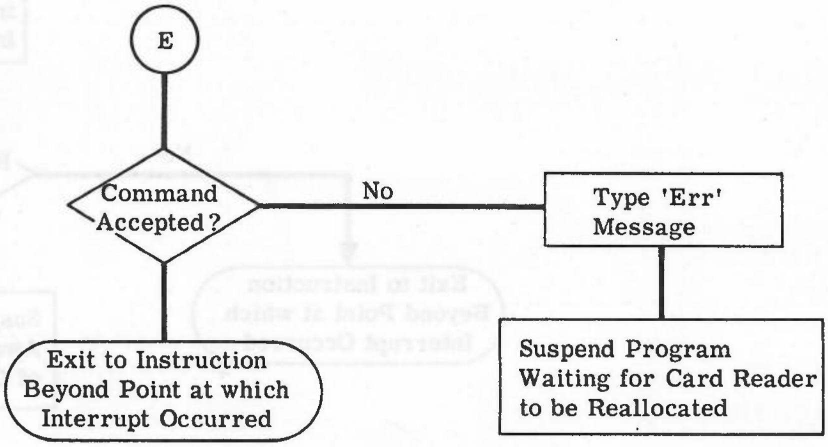
Flowchart 1


A
Form Hesitation Control Words
Would Transfer Cause Hesitation Overload
Yes
Is an Instruction Already in Progress on this Card Reader
Yes
Preserve Hesitation Control Words in Hesitation Registers
B
Inhibit Interrupts
C
Set 'Instruction in Progress' Indicator
Place Hesitation Control Words in Hesitation Registers
Send Appropriate Command to Card Reader and Obtain Direct Response
D

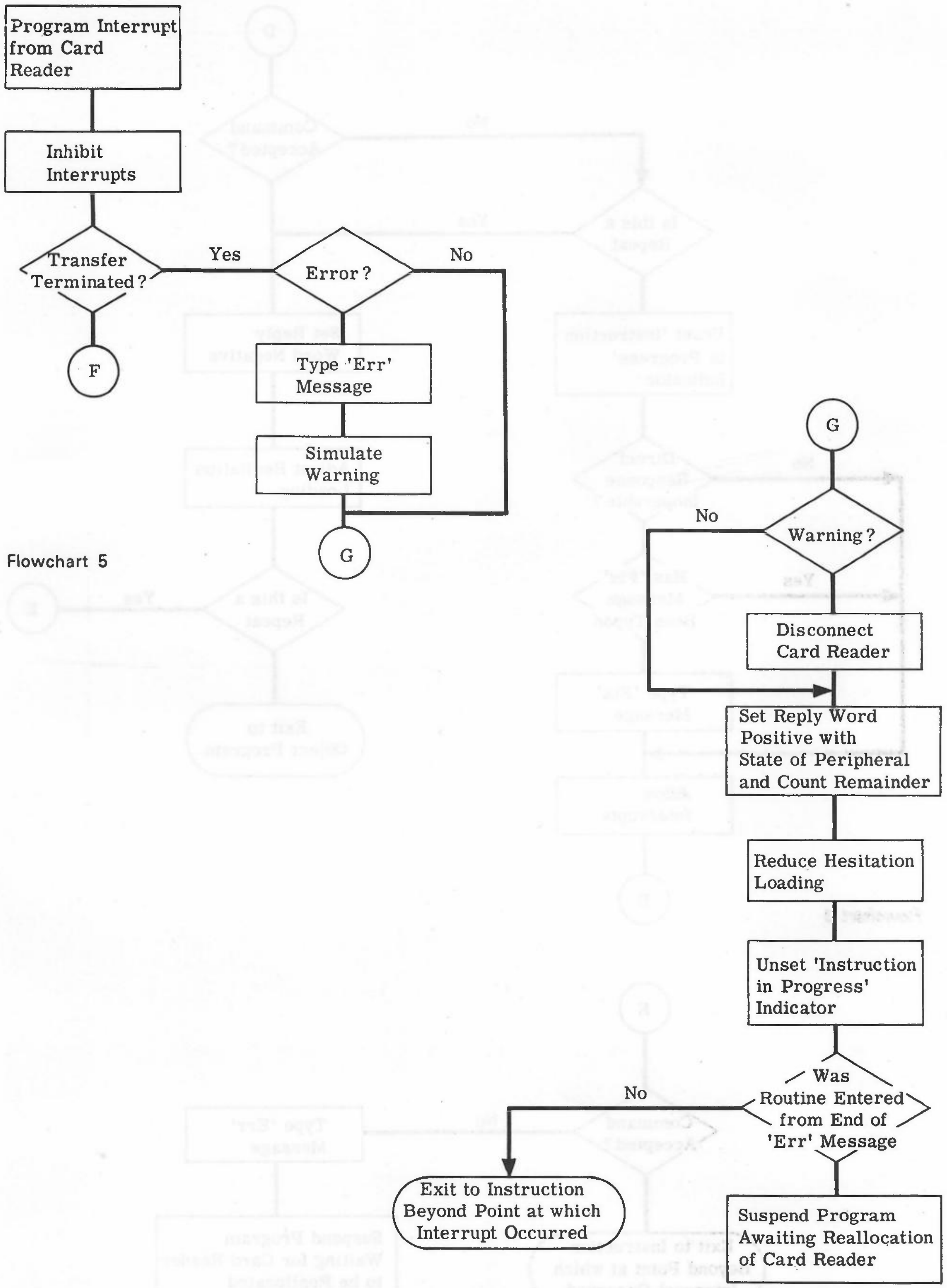
Flowchart 2



Flowchart 3

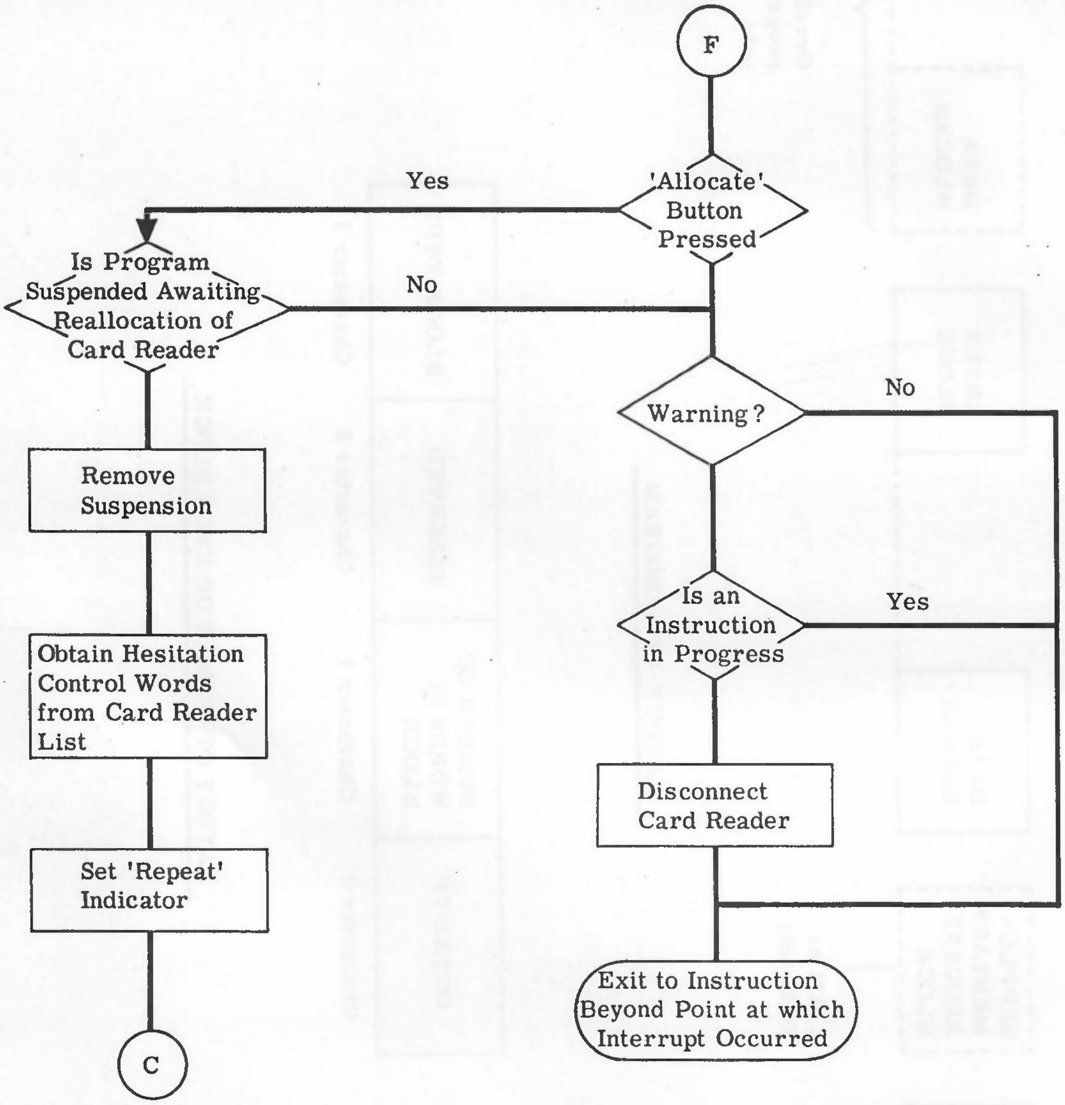

E
Command Accepted?
No
Type 'Err' Message
Exit to Instruction Beyond Point at which Interrupt Occurred
Suspend Program Waiting for Card Reader to be Reallocated

Flowchart 4


Program Interrupt from Card Reader
Inhibit Interrupts
Transfer Terminated?
Yes
Error?
No
F
Type 'Err' Message
Simulate Warning
G
G
No
Warning?
Yes
Disconnect Card Reader
Set Reply Word Positive with State of Peripheral and Count Remainder
Reduce Hesitation Loading
Unset 'Instruction in Progress' Indicator
No
Was Routine Entered from End of 'Err' Message
Exit to Instruction Beyond Point at which Interrupt Occurred
Suspend Program Awaiting Reallocation of Card Reader

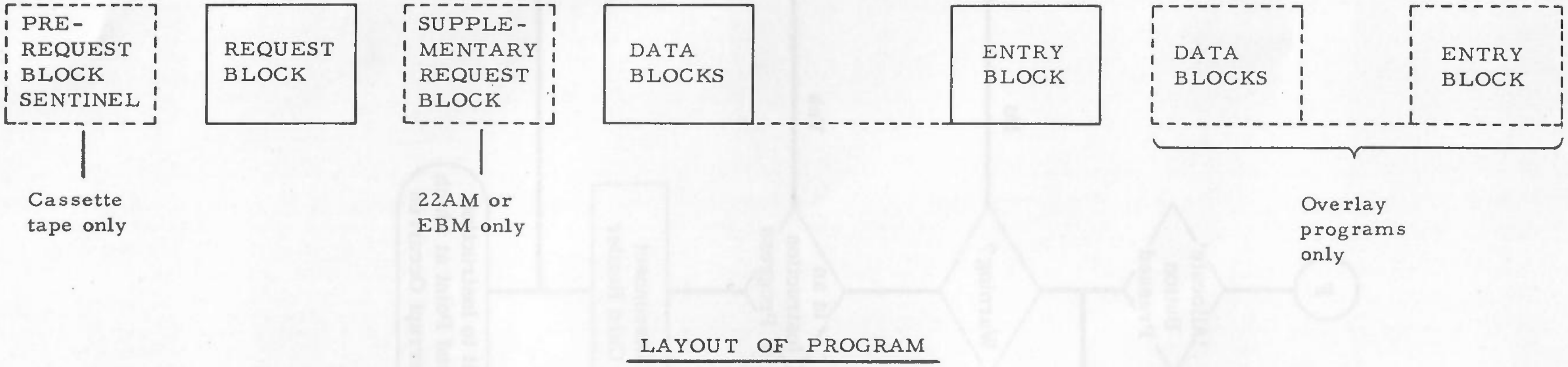
Flowchart 5

Flowchart 6


F
'Allocate' Button Pressed
Yes
No
Is Program Suspended Awaiting Reallocation of Card Reader
Remove Suspension
Obtain Hesitation Control Words from Card Reader List
Set 'Repeat' Indicator
C
Warning?
No
Is an Instruction in Progress
Yes
Disconnect Card Reader
Exit to Instruction Beyond Point at which Interrupt Occurred

Flowchart 7

Figure 13 Format of a binary program



OCTAL 73	NUMBER OF WORDS IN BLOCK	RESERVED	BLOCK TYPE
Character 0	Character 1	Character 2	Character 3

LAYOUT OF WORD 0 OF EACH BLOCK

BINARY PROGRAMS

Although programs are normally written in a convenient language e.g. PLAN, FORTRAN, COBOL, the central processor and Executive can handle programs only in machine code. This form is known as *binary program:* It is a function of the compilers and associated routines, such as the General Purpose Loader, to convert the source language program into binary program.

Some compilation processes convert the source program into binary program and output it on a suitable medium for preservation and loading into store. Others leave the binary program in the processor store, ready for use. In the latter case it is possible to make Executive output the binary program onto a suitable medium, such a process being known as *dumping*.

This part of the manual describes the formats used on various media for binary programs and the means by which loading and dumping can be carried out.

Binary program formats

All binary programs input or output by Executive have the following standard format:

1 A request block indicating to Executive such information as the program's size, peripheral requirements and program/subprogram priorities where appropriate. This may be followed by a supplementary request block.

2 The binary program data blocks.

3 An entry block that indicates to Executive where to enter a program and the required action on the program, i.e. GO, SUspend, or COntinue with next program instruction.

Block types

All blocks consist of an integral number of 24-bit words, the first of which indicates the block type and size. The types and their numbers are as follows:

Type 1	Request block, also known as request slip
Type 0 or 5	Data blocks
Type 2, 3 or 4	Entry blocks
Type 6	Supplementary request block
Type 62	Pre-request block sentinel

The first word of each block contains the following information:

Char: 0	Octal 73
Char: 1	Number of words in block (except pre-request block sentinel)
Char: 2	Reserved
Char: 3	Block type

A supplementary request block is present only if the program is to be run in 22AM or EBM and, if present, follows the request block. The pre-request block sentinel applies only to programs stored on cassette tape and, if present, immediately precedes the request block. In overlay programs there is an entry block for every overlay in addition to the normal one for the initial entry.

The above information is illustrated in Figure 13.

Block type 1: request block (request slip)

A request block has the following format:

Word 0:	Block specifying word
Word 1:	Program name
Word 2:	Peripheral request and trusted program status word
Word 3:	Core store request word
Words 4 and 6:	Reserved

Word 5:	Overlay directory word
Word 6:	Self-monitoring address
Word 7:	Priority of Member 0, which must exist
Words 8 to 12:	Priorities of other members, which need not exist
Word 13:	Negative check sum
Words 14, 15:	Optional characters

The contents of each word are further defined below.

WORD 0

As standard, with character three set equal to one.

WORD 1

Four character program name composed of letters and digits of which the first character must be a letter. The name EXEC is reserved. The four characters are used to identify the program within the system and in Executive/operator communication. Optionally up to 8 further program name characters may be placed in Words 14 and 15 for use as an accounting code, see page 84.

WORD 2

Bits 0 and 1

If either Bit 0 or Bit 1 is set to 1, the peripheral from which the program is loaded will be assigned to the program as unit 0 of its type. If one or more peripherals of the same type are requested in Bits 6 to 8 or Bits 15 to 17, the setting of Bit 0 or Bit 1 will not result in an additional unit being assigned; it will, however, ensure that the load peripheral is the one retained as unit 0.

The distinction between Bit 0 and Bit 1 is that when the program is dumped Bit 0 is always zero, but Bit 1 is in the same state as on loading.

Bit 0 is used with programs in G.P.L. form to ensure that the loader reads the semi-compiled program from unit 0. It is set automatically by compilers and consolidators.

Bits 2 to 5

The table below gives the significance of each of these bits if set.

Bit	Status	Significance
2	Q	Writes to direct access system/directory files
3	R	trusted program facilities used
4	S	trusted program facilities used; always set to zero on program dumping
5	T	Requires use of GEORGE 3 Executive

Bits 6 to 20

These bits specify the number of basic peripherals required by the program as follows:

Bits 6 to 8	Number of paper tape readers
Bits 9 to 11	Number of paper tape punches
Bits 12 to 14	Number of line printers
Bits 15 to 17	Number of card readers
Bits 18 to 20	Number of card punches

Bits 21 to 23

Reserved

WORD 3

This word specifies the core store requirement of the program as shown below.

Bits 0 to 8

The number of units of 64 words, held as a binary number.

Bits 9 to 16

Must be zero.

Bits 17 to 23

The number of additional units of 32,768 words, held as a binary number.

WORD 5

If this word is zero, the bootstrap will take no action. If the word is non-zero, Bits 0 and 1 are reserved and the remaining bits give the address of the first word of the overlay directory. In this case the program's subfile description is scanned and a directory giving the layout of the program's overlay and permanent units is built up starting at the word indicated by the value of Bits 2 to 23.

WORD 6

Address for self-monitoring of illegal orders or floating-point overflow. Zero if no self-monitoring.

WORD 7

Bits 0 to 11

The priority of Member 0 held as two characters.

Bits 12 to 17

Octal 77.

Bits 18 to 23

The number of the member, in this case 0.

WORD 8

The priority of Member 3 held in the same format as Word 7. If this or other members below do not exist, their priority word must be set to zero.

WORD 9

The priority of Member 1 held in the same format as word 7.

WORD 10

Must be set to zero but must not subsequently be assumed to contain zero.

WORD 11

The priority of Member 2 held in the same format as Word 7.

WORD 12

The priority of Member 5 held in the same format as Word 7.

WORD 13

This word holds the check sum. The number of words in the request record is given in Character 1 of Word 0. The value of the check sum word is such that if the words in the record (starting with Word 0) are summed using the 127 order the result is zero.

WORDS 14 and 15

Optional program name extension, comprising up to 8 letters or digits, left-justified. These words are used by the Log Analysis program in accordance with its specification. They are included in the check sum word count character of Word 0 and are output on dumping.

Block type 0: data block

This block contains between 1 and 16 words of program plus 3 or 4 further words as described below. This type of data block is usually used with media other than magnetic tape or direct access media since considerable economies result from the use of type 5 data blocks with these media.

WORD 0

As standard with character three set equal to zero. The word count is always the number of program words plus three.

WORD 1

The destination address, relative to the datum, of the first word of program in the block i.e. Word 2.

WORD 2

The first word of program data in the block.

WORDS 3 to 17

These words are optional on paper tape but always present on punched cards and magnetic or cassette tape. If present, they contain further words of program or zero.

NEXT WORD

Negative check sum.

WORD 19

Optional block sequence number. If present, this word is ignored by Executive and is not included in the word count in Word 0. This word applies to cards only.

Block type 5: data block

This is a data block pair and consists of a five-word specifying block and a further block that consists entirely of words of program and has a maximum length of 512 words. The block pair is regarded as a single block for the purpose of block counts. The format of the specifying block is given below.

WORD 0

As standard, with character three set equal to five.

WORD 1

The destination address, relative to the datum, of the first word of the following data block.

WORD 2

The length of the data block (maximum 512 words).

WORD 3

Negative check sum of data block.

WORD 4

Negative check sum of the specifying block.

Block type 2: entry block

This is a three or four-word block (see Block Counts) that instructs Executive to GO at the specified starting address of the program.

WORD 0

As standard, with character three set equal to two.

WORD 1

The address at which the program is to be entered.

WORD 2

Negative check sum or block count.

WORD 3

Negative check sum if there is a block count in Word 2.

Block type 3: entry block

This is a three or four-word block (see Block Counts) that instructs Executive to SUSPEND the program after loading and await operator action. If the operator types GO without specifying an address the program is entered at the address specified in Word 1 of this block.

WORD 0

As standard, with character three set equal to three.

WORD 1

The address at which the program is to be entered if the operator types GO.

WORD 2

Negative check sum or block count.

WORD 3

Negative check sum if there is a block count in Word 2.

Block type 4: entry block

This is a three- or four-word block (see Block Counts) normally used in conjunction with the 154 (CONT) order to instruct Executive to read in more program. A program using this type of block will be overlaid. This type of block is used to terminate the overlay and the program will be re-entered at the instruction following the 154 instruction.

WORD 0

As standard, with character three set equal to four.

WORD 1

Zero

WORD 2

Negative check sum or block count.

WORD 3

Negative check sum if there is a block count in Word 2.

Block type 6: supplementary request block

The supplementary request block is used with programs to be run in 22AM or EBM. If present, it normally follows immediately after the standard request block. If the presence of a supplementary request block is detected by an Executive without 22AM or EBM facilities, then the program will be rejected, even if Words 0 to 6 of the supplementary request block are zero.

WORD 0

As standard, with character three set equal to six.

WORD 1

This word contains the mode setting of the program in Bits 21 and 23. The remaining bits should be set to zero but must not be assumed to be zero when the mode word is tested.

Bit 21

0 indicates DBM

1 indicates EBM

Bit 23

0 indicates 15AM

1 indicates 22AM

WORDS 2 to 6

Reserved.

WORD 7

Negative check sum of block.

Block type 62: pre-request block sentinel

This block is used only with cassette tape and, if present, immediately precedes the request block.

WORD 0

Octal 73000076

WORD 1

Zero

WORD 2

Reserved

WORD 3

The program name

Block counts and sequence numbers

When Executive dumps a program it produces a four-word entry block, putting in Word 2 the total number of blocks dumped, and using Word 3 for the negative check sum. Three word entry blocks, produced by some compilers, are acceptable as input and have the check sum in Word 2 and no block count. Executive also, on some media (e.g. punched cards), extends type 0 blocks to 20 words, using Word 19 to provide a block sequence number; but it does not amend the block specifying word, which always excludes Word 19 from the count.

On loading, Executive will, if the entry block is four words long, check that the total number of blocks read is correct. No check is performed on the sequence word of a type 0 block which is intended for external purposes only (e.g. for identifying a card).

When the program read in does not have a request block (i.e. in connection with a handswitch Executive or program read as a result of a 154 instruction) the block count in the entry block must be one greater than the actual number of blocks. The request block may be removed from the result of a dump, so that the latter can be overlaid.

Layout of binary program on various media

PAPER TAPE

Blocks are punched in eight-track tape in graphic set mode with even parity, each block being terminated with a Newline character and followed by a gap of at least three blank tape characters. No block sequence word is permitted.

80-COLUMN CARDS

Blocks are stored one per card, each column representing a six-bit pattern. The maximum number of columns required to hold a block will thus be 76 (in the case of a 19-word data block type 0). The last four columns of each card are therefore available and are used to contain a card count which allows card sequence checks to be made. The card count is not included as a word to be sum-checked and thus does not affect the previously detailed block check sum procedure. It should be noted that this sequence number is not punched in decimal form, but consists of the card code characters produced by the binary value of the s equence number.

MAGNETIC TAPE

Blocks of 20-word minimum length are written on magnetic tape in odd parity mode. Nothing may be assumed about data (which will be redundant to Executive) beyond the block length specified in Word 0. The format of a program tape that is requested by means of the FInd message containing one name is as follows:

1 A header label with the file name PROGRAM$_\nabla$*Name*0000, where *Name* is the four character stored in Word 1 of the request block and 0000 is the reel sequence number.

2 Tape mark.

3 Blocks as described at the beginning of this section.

When a program is loaded from magnetic tape either by the LOad message or the 154(CONT) order, the tape must be positioned immediately before the first block to be read; i.e. the request block if LOad is used, the first data block if the 154 (CONT) is used. Following a LOad message, the tape will not remain allocated to the program unless Bit 0 or Bit 1 in Word 2 of the request block called for the load peripheral to be retained.

The DUmp directive finds a scratch tape on an unallocated deck and constructs a tape with the format shown in items 1 to 3 above. The tape is then rewound and disengaged i.e. removed from Executive's list until re-engaged. On most processors, the data blocks dumped onto magnetic tape are of type 5, but 1901 to 1903 and 1901A to 1903A Executives always dump type 0 blocks.

A dumping action by means of the 155 (SUSDP) order produces a request block, data blocks and an entry block type 3, no end of file marks being written.

DIRECT ACCESS

The format of binary programs stored on direct access devices will be published later.

Executive treatment of request and supplementary request blocks

EX1H, E1HS

With this Executive all store and peripherals are regarded as being permanently assigned to the object program. Request blocks are ignored on program loading, though if one is present its check sum must be correct.

A supplementary request block is signalled as an error, and the RRQ (166) order is null.

There is no storage of request block details, and a program dump has no request block. Such a dump on re-loading is acceptable to EX1H and E1HS, but not to an Executive for a processor with a console typewriter unless a request block is added.

EX2L

As with EX1H, there is no assignment of peripherals, and all the store is made available to the program. The request block must sum check correctly and the name must agree with the name on the console LOad message. A check is then made that the core store requirement stated on the request block does not exceed the available storage. The request block is stored with Bit 0 of Word 2 zero.

A supplementary request block is signalled as an error. The RRQ (166) instruction is null and the stored request block therefore remains unchanged during the running of the program.

A dump by Executive of the object program is preceded by a copy of the stored request block.

EX1T, E1TS,EX1V, E1DS, E1MS, EX2S, E3TS, E3TE, EX2V, E3DS, E3DE, EX2M, E4BM, E3TM, E3DM

The first eleven of these are single program Executives in which peripherals are assigned to the program's unit numbers on demand. All available storage outside Executive is regarded as being available to the object program.

The remainder are dualprogramming and multiprogramming Executives in which core store and peripherals are assigned and released on demand.

Program loading

These Executives sum check all binary program records on reading them. The following checks and actions are then carried out on the request block:

1 Check that this is the first record of the program.

2 Check that the program name in the request block agrees with the name on the console LOad message.

3 Check that the core store requirement (from the request block unless overridden by the LOad message) is available.

4 Assign load peripheral if Bit 0 or 1 of Word 2 is set.

5 Check that the peripherals called for in the request block are available; these units are then assigned to the program.

In addition, dualprogramming and multiprogramming Executives establish the priorities of members; E4BM, E3TM and E3DM also monitor the program name extension. The request block is stored with Bit 0 of Word 2 zero.

The appearance of a supplementary request block during input of the program is signalled as an error.

Changes during execution

No alterations are made to the stored request block as a result of such events as assigning or releasing peripherals, or variations in core store used. Changes can arise as the result of 166, X = 1 instructions, which completely replace the old request block by a new one.

Program dumping

A dump by Executive of the object program is preceded by a copy of the latest stored version of the request slip.

E6BM

This Executive handles the request block in the same way as E4BM. The supplementary request block is recognized, and causes 22AM and/or EBM to be set as indicated in the mode word. If the program switches these modes during execution, the new setting is used in any program dumping operation. On dumping, a supplementary request block is produced immediately following the standard request block unless the contents of the supplementary request block would have been zero.

Dumping

The dumping of a program can be initiated by the operator, or by the program itself by means of a 155 (SUSDP) order. It should be noted that, when Executive is instructed to dump a program, it does so without checking the state of any peripheral devices other than the one to be used for the dump. As a result, should the program concerned have any peripheral active at dump time, the result of the dump may be indeterminate.

In all cases the request block output will be a copy of the original except that bit 0 of word 2 will always be zero and the check sum adjusted accordingly. The core-store figure will always be the original value, which may differ from the actual current size of the program due to a 165, N(M) = 4 (GIVE) order, or because the operator specified the amount of store to be used at load time. The entry block produced will always be of type 3.

DUMPING OF ZEROS

A dump that is in the form of type 0 data blocks (i.e. all except a magnetic tape dump on 1904 and above) will omit any blocks in which all program words are zero as, on loading, the store is initially zeroized.

DEVICES THAT MAY BE USED FOR DUMPING

These are paper tape punches, card punches, and magnetic tape.

PROGRAM INITIATED DUMPS

The 155 (SUSDP) instruction, causes the program to be suspended and then dumps the complete program area. Such a dump is written to the specified device without any prior repositioning of the output medium. On completion of the dumping the program continues with the instruction following the 155 (SUSDP) instruction, without operator intervention.

OPERATOR INITIATED DUMPS

Usually the program concerned will already be suspended when the operator calls for a dump. Should this not be the case, it will be suspended immediately, and, on completion of the dumping, operator action will be required to restart it.

If a particular device is specified by the operator the device specified may be only a paper tape or card punch, and it must be either a device currently assigned to the program to be dumped or a device that is not assigned to any program. If no device is specified by the operator, Executive will dump on magnetic, but not cassette, tape. Executive locates, on any tape deck that is not currently assigned to any program, a tape that is available for use as an output tape (i.e. retention period exceeded). It labels this tape PROGRAMγ*name*, giving it reel sequence and generation numbers of zero. In order to give reasonable protection to the dump a retention period of seven days is given. Should Executive not be able to locate a suitable tape, the operator will be notified. If there is no magnetic tape, or only cassette tape, connected to the processor, then a dump message that does not specify a device is unacceptable.

On completion of the dump, the tape is closed but the operator is not notified which tape was used; any retrieval procedure must use either the FInd directive or a program that opens the dump by name.

Loading

In general, the loading of a program is operator initiated, but under certain circumstances it may be program initiated. On loading, the store is initially zeroized.

Devices that may be used for program loading are paper tape readers, card readers and magnetic and cassette tape.

PROGRAM INITIATED LOADING

During the running of a program

The 154 (CONT) instruction provides the programmer with a simple type of overlay facility. It causes the program to be suspended immediately. Executive then reads binary program, which must not have a request block, into the program's area, the locations overwritten being determined by the data blocks read. The area read into is not zeroized before loading, hence care must be taken in respect of areas which were zero when the binary program was created. On completion of the loading, the action taken will depend on the type of entry block read. Usually, this will be of type 4, which will cause the program to continue with the instruction in the location following that which contained the 154 (CONT) instruction. (Either or both of these locations may have been overwritten by the program just read in.)

The input medium must be correctly positioned, immediately before the first data block, at the time that the 154 instruction is given.

Note: When a 154 instruction is initiated Executive starts the loading process without checking the state of any peripheral device, other than the one to be used in the loading process. As a result, should the program which issues the 154 instruction have any peripheral active at the time, the results of the loading may be indeterminate.

When a program deletes itself

If a program deletes itself by means of a 160 (DELTY) instruction, Executive obeys the message output by the program as if it has been typed by the operator. This enables one program to initiate the loading of another. The effect of such an instruction in no way differs from the effect of the operator typing the same message. (See below for further details.) However, if the output message is LOad specifying magnetic or cassette tape, the tape specified will not be rewound when the program issuing the 160 instruction deletes itself.

OPERATOR INITIATED LOADING

There are two operator directives which can initiate program loading.

LOad

This directive causes Executive to read binary program from the specified device, which is normally a paper tape reader or card reader. Such a program must be in the standard form commencing with a request block. The program bearing medium must be correctly positioned in the reading device, so that the first block read is the request block.

The LOad directive will be used in respect of magnetic or cassette tape only as a result of a 160 X = 2 (DELTY) order.

FInd

The FInd message is used to load a program from backing storage. In some cases Executive calls in a search program which locates and loads the actual program required. The information given below about the actions of search programs describes conventions and not rules; for details see the specifications of search programs.

The backing storage medium primarily used is specified when an Executive is compiled, and the action of the FInd message depends on the medium specified.

The FInd message on processors with console typewriters has two main forms, specifying either one or two program names. In either case there may follow a decimal integer, equal to 64 or more, that specifies a core store reservation that is to be used instead of that specified in the request block. On 1904 processors and upwards decimal integers less than 64 may follow that represent the absolute number of peripherals that are to be allocated to the program to be loaded, each as unit 0 of its type. Hence there must be not more than one of any type allocated by this method. Not more than three integers in total may be specified in a FInd directive.

Direct access peripherals cannot be allocated in this way.

Action of the FInd directive with magnetic tape (industry compatible) and cassette tape as primary backing storage medium

In response to FI #*Name* where #*Name* represents any legal program name, Executive searches all the currently unassigned tape decks or cassette stations for a tape with the file name PROGRAM$_\nabla$*Name* and reel sequence number zero (as if opening it in mode #100 as unit 0). If it finds one it loads the binary program from that tape and further action is determined by the program loaded. In the case of cassette tape, Executive searches for a block with the block sequence word negative (this should be a pre-request block) before loading the program.

If the name #*Name* is that of a search program, the latter on entry will conventionally halt and output the message HALTED SL. The operator should then type a GO AT directive to indicate whether a steering line to specify the program actually required is provided on cards or paper tape.

If Executive does not find a file called PROGRAM$_\nabla$*Name* it will instruct the operator to load one. Formerly, under these circumstances, after a 160, X = 2 (DELTY) instruction, Executive would search for a magnetic tape called PROGRAM$_\nabla$TAPE and use #TAPE as a search program to locate the load PROGRAM$_\nabla$*Name*. This can still be achieved, if, after the LDT PROGRAM$_\nabla$*Name* console message, the operator types FI #*Name* #TAPE followed by the integers of the original FInd directive.

On processors without console typewriters the FInd process (handswitch setting 20/0) reads from the cassette station currently allocated as the program's unit 0 until it finds a block with block sequence word negative, which should be a pre-request block, and then loads the program.

In response to FI #*Nam* #*Nam*2, where #*Nam*1 and #*Nam*2 represent any legal program names, Executive will search all the currently unassigned decks or cassette stations for a tape with file name PROGRAM$_\nabla$*Nam*2. If no such tape is found the message LDT PROGRAM$_\nabla$*Nam*2will result. If the tape is found Executive will load the first program on it, which must be called #*Nam*2 and is conventionally a search program, and will pass to it the name #*Nam*1, any core requests and the additional information in the FInd message as follows:

Word 10	Name of program required (in this example #*Nam*1)
Words 11 to 13	
Bits 2 to 23	Integers specified in the FInd message
Bits 0 and 1	Indeterminate

If less than three integers are specified, Bits 2 to 23 of the remainder of Words 11 to 13 may be set to zero or left as loaded (conventionally zero in any case).

Conventionally, the search program will locate the program specified, position the tape before the request block and obey a 160, X = 2 (DELTY) instruction which implements the message LO #*Nam*1 followed by the geographical address of the tape deck and any integers in the FInd directive.

Action of the FInd directive with E.D.S. or F.D.S

In response to any FInd directive, e.g. FI #*Nam*1#*Nam*2, where #*Nam*1 and #*Nam*2 represent any legal program names, Executive loads a special search program from the reserved area on the disc into the store, assigns it the first name in the FInd directive, and hands over the information in the FInd directive.

If the message is of the form FI #*Nam*1, the search program will conventionally try to open a file with name PORGRAM$_\nabla$*Nam*1 as unit 0 and load the program from it, taking the usual action on the integers.

If the message is of the form FI #*Nam*1#*Nam*2, the search program will conventionally try to open a file with name PROGRAM$_\nabla$*Nam*2 as unit 0, and load the program from its subfile named PROGRAM$_\nabla$*Nam*1, taking the usual action on the integers.

In both cases, the search program changes its request block in order to convert itself into the program #*Nam*1 and some low numbered locations may be corrupted in the loading process. These locations will be detailed in the specifications of the relevant search programs.

The action taken if either the file or the subfile is not found will depend on the search program; for example, some search programs will proceed to repeat the attempt on another medium e.g. magnetic tape. For details see the specification of the search program used.

OPERATOR/EXECUTIVE COMMUNICATION

A comprehensive description of the use of the console typewriter for communication between the operator and Executive is given in the appropriate console operating manual. This account therefore confines itself to a general description of the way in which an input message is dealt with by Executive.

Monitoring of input messages

There is an important difference in the treatment of typewriter messages between Executives in the range 1903 and below, and Executives higher in the 1900 Series. In the former case, the striking of each character key causes an interrupt to Executive, which then causes the corresponding character to be typed out on the log. Every character after the first is monitored individually for error conditions which, if encountered, will instantly be brought to the operator's notice. Thus, when an error message is typed out, the operator knows that either

(a) the error message refers to the first or second characters typed in, or both, or

(b) the error message refers to the last character typed.

In these circumstances, few error messages are needed since the source of error is precisely defined.

Executives in the range 1904 and upwards operate differently. The striking of each key on the console typewriter causes the corresponding character to be transferred into store by hesitations as with other peripherals, but an interrupt causing Executive to be entered occurs only when the ACCEPT key is pressed. Executive then monitors the whole message, an error condition at any point in the input message causing an error message to be output. Since the operator cannot know precisely at what point the error occurred, the error message must contain this information, and there must therefore be a larger number of error messages.

The typing of input and output messages on the 1902 and 1903 is timeshared with the running of the program, except in the case of messages from the program (SUSTY, DISTY, DELTY, SUSWT, DISP, and DEL). The 1901 processor, however, stays in Executive mode during the time that a message is input or output, so that the object program may be suspended for several consecutive seconds.

Description of executive treatment of input typewriter message

BELOW IS GIVEN A DESCRIPTION PROVIDED FOR INTEREST ONLY, OF THE WAY IN WHICH AN EXECUTIVE ON ONE OF THE SMALLER PROCESSORS IN THE 1900 RANGE DEALS WITH THE GO MESSAGE WHEN IT IS INPUT. NO ASSUMPTION SHOULD BE MADE AS TO THE RELEVANCE OF THIS DESCRIPTION TO ANY PARTICULAR EXECUTIVE.

The description is illustrated by a number of flowcharts.

FLOWCHART 1

When a console interrupt occurs, if the typewriter is not in the output state, Executive sets an indicator to show that a message is being interpreted. If the input character is 'Accept', a branch is made to avoid a subsequent test that would treat this character as illegal. A test is then made that the character was not input in error, and if this test is satisfied, the character is typed on the console log. 'Control A' is a facility that allows messages to Executive to be input from a card or paper tape reader instead of from the console typewriter. The printable set of characters in those that are in the code range 00 to 63; if an input character is not one of this set, or the Accept character, an indeterminate character will be output, and an error message typed out. This is the only error that is monitored for the first input character.

FLOWCHART 2

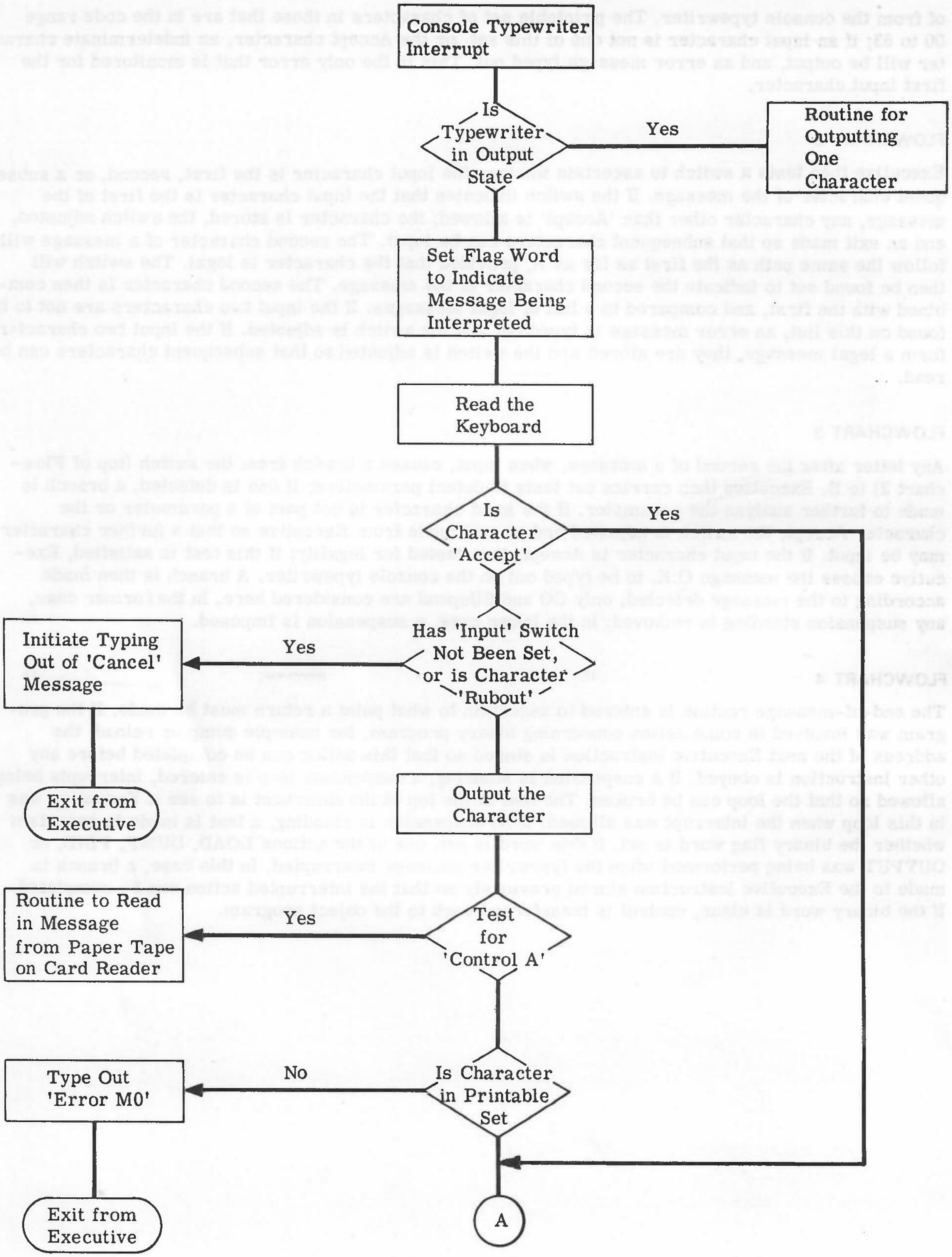
Executive then tests a switch to ascertain whether the input character is the first, second, or a subsequent character of the message. If the switch indicates that the input character is the first of the message, any character other than 'Accept' is allowed; the character is stored, the switch adjusted, and an exit made so that subsequent characters can be input. The second character of a message will follow the same path as the first as far as A, provided that the character is legal. The switch will then be found set to indicate the second character of the message. The second character is then combined with the first, and compared to a list of legal messages. If the input two characters are not to be found on this list, an error message is typed out and the switch is adjusted. If the input two characters form a legal message, they are stored and the switch is adjusted so that subsequent characters can be read.

FLOWCHART 3

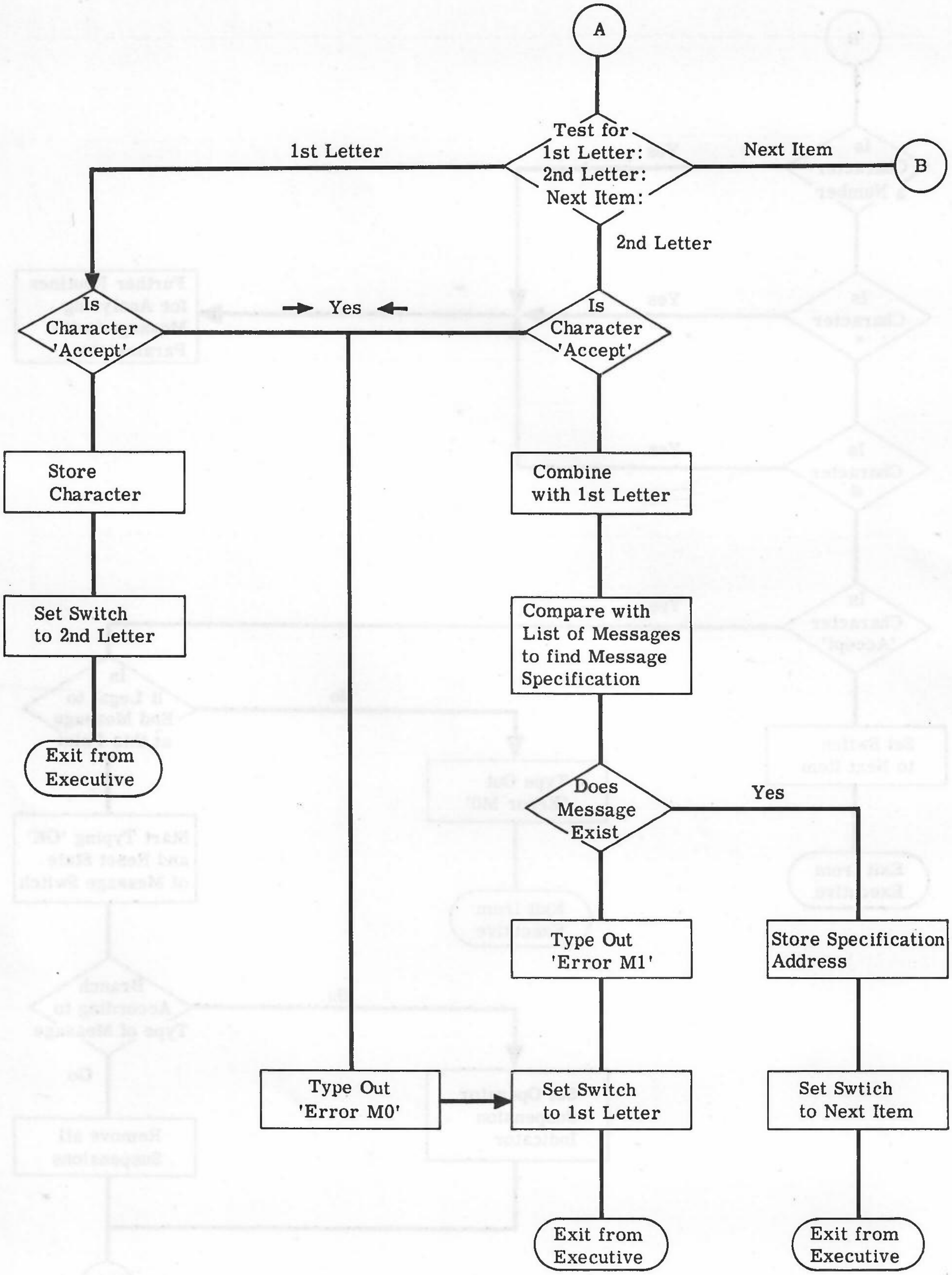
Any letter after the second of a message, when input, causes a branch from the switch (top of Flowchart 2) to B. Executive then carries out tests to detect parameters; if one is detected, a branch is made to further analyze the parameter. If the input character is not part of a parameter or the character Accept, the switch is adjusted and an exit made from Executive so that a further character may be input. If the input character is Accept, it is tested for legality; if this test is satisfied, Executive causes the message O.K. to be typed out on the console typewriter. A branch is then made according to the message detected; only GO and SUspend are considered here. In the former case, any suspension standing is removed; in the latter case, a suspension is imposed.

FLOWCHART 4

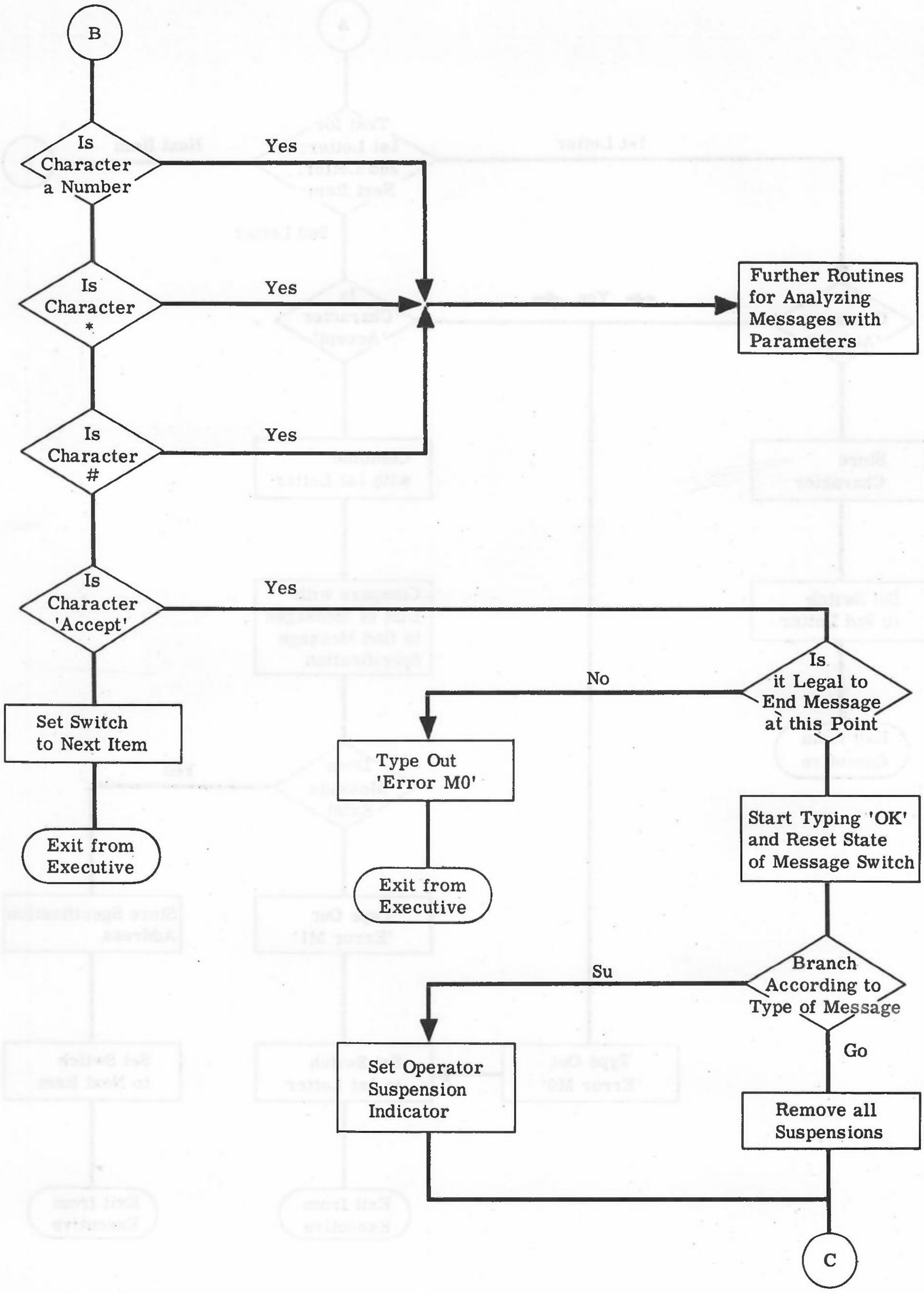
The end-of-message routine is entered to ascertain to what point a return must be made. If the program was involved in some action concerning binary program, for example dump or reload, the address of the next Executive instruction is stored so that this action can be completed before any other instruction is obeyed. If a suspension is standing, a suspension loop is entered, interrupts being allowed so that the loop can be broken. The test at the top of the flowchart is to see if Executive was in this loop when the interrupt was allowed. If no suspension is standing, a test is made to ascertain whether the binary flag word is set. If this word is set, one of the actions LOAD, DUMP, FIND, or OUTPUT was being performed when the typewriter message interrupted. In this case, a branch is made to the Executive instruction stored previously so that the interrupted action can be completed. If the binary word is clear, control is transferred back to the object program.


Console Typewriter Interrupt
Is Typewriter in Output State
Yes
Routine for Outputting One Character
Set Flag Word to Indicate Message Being Interpreted
Read the Keyboard
Is Character 'Accept'
Yes
Has 'Input' Switch Not Been Set, or is Character 'Rubout'
Yes
Initiate Typing Out of 'Cancel' Message
Exit from Executive
Output the Character
Test for 'Control A'
Yes
Routine to Read in Message from Paper Tape on Card Reader
Is Character in Printable Set
No
Type Out 'Error M0'
Exit from Executive
A

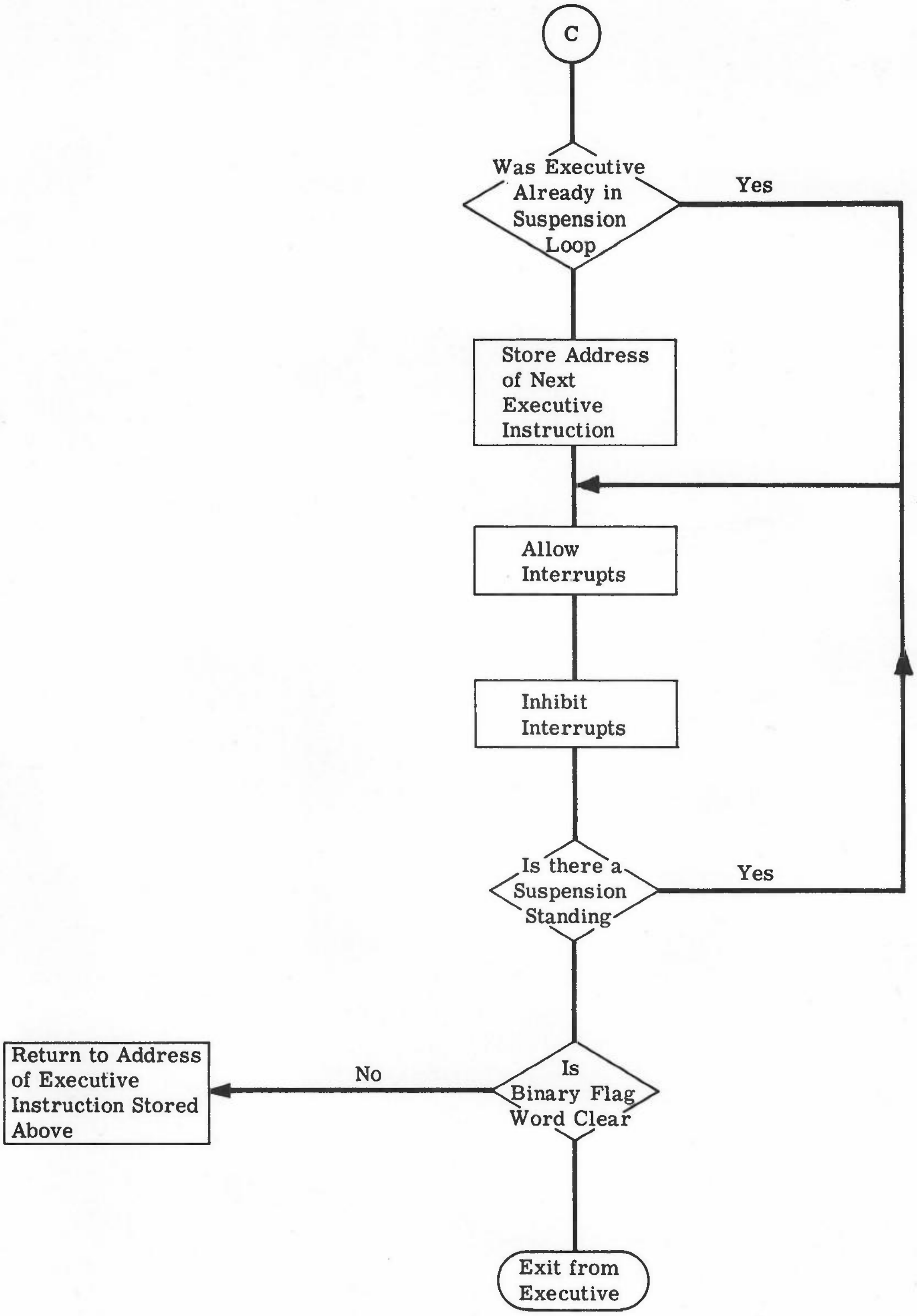
Flowchart 1


A
Test for 1st Letter: 2nd Letter: Next Item:
1st Letter
Next Item
B
2nd Letter
Is Character 'Accept'
Yes
Is Character 'Accept'
Store Character
Combine with 1st Letter
Set Switch to 2nd Letter
Compare with List of Messages to find Message Specification
Exit from Executive
Does Message Exist
Yes
Type Out 'Error M1'
Store Specification Address
Type Out 'Error M0'
Set Switch to 1st Letter
Set Swtich to Next Item
Exit from Executive
Exit from Executive

Flowchart 2


B
Is Character a Number
Yes
Is Character *
Yes
Is Character #
Yes
Further Routines for Analyzing Messages with Parameters
Is Character 'Accept'
Yes
Set Switch to Next Item
Exit from Executive
Is it Legal to End Message at this Point
No
Type Out 'Error M0'
Exit from Executive
Start Typing 'OK' and Reset State of Message Switch
Branch According to Type of Message
Su
Go
Set Operator Suspension Indicator
Remove all Suspensions
C

Flowchart 3


C
Was Executive Already in Suspension Loop
Yes
Store Address of Next Executive Instruction
Allow Interrupts
Inhibit Interrupts
Is there a Suspension Standing
Yes
Is Binary Flag Word Clear
No
Return to Address of Executive Instruction Stored Above
Exit from Executive

Flowchart 4

Chapter 8 Paging

THE PRINCIPLES OF PAGING

Unpaged systems

On most 1900 Series central processors a program being run must either be held in its entirety in core store or else be divided into discrete parts that can be overlayed. In either case, the programmer must keep the addresses that he references within the limits of the core store locations available. Protection between object program areas is achieved by storing the datum and limit of each area and making illegal any attempt by a program to reference an address outside its area. This scheme has the virtue of simplicity, but lacks the flexibility desirable in very large computer systems.

Paged systems

In a paged system the whole store, core store plus backing store, is theoretically available to the programmer. The addressable area is limited only by the number of bits available to hold an address, which in 22AM is 22, giving an addressable area of over four million words.

The physical store is divided into areas, each of the same size, known as *pages* Each page may hold a *block* of program, which may be either a whole program or any part of one program. The division of a program into blocks and the allocation of program blocks to pages is under the control of the operating system, which compiles a directory of the page address of each block. The directory also contains *permission* bits, associated with each page, which indicate whether a page can be written to, read from, or both, or whether the contents are to be executed as program.

The flexibility derived from a paging system lies in the ability to distribute blocks of program or data over the total storage area in a manner most likely to ensure efficient use of facilities. The programmer therefore has the entire internal and backing storage facilities at his disposal. There is no longer any need to be concerned with the size of core store available; instead the programmer think in terms of 4M words. A block of program must be resident in core store while it is being executed, but is normally held on backing store and is brought in to core store when required. The program blocks need not therefore be discrete parts of a program since any reference to a block not currently in core store will cause the operating system to retrieve the block from backing store.

Moreover, program areas need not be mutually exclusive. By means of the permission bits, any block of program or data may be made only conditionally exclusive, so that a single program may be operating on a number of sets of data at the same time.

PAGING OPTIONS

Paging is an optional feature, available with the 1904A and the 1906A, which may be specified initially or added in the field. When it is fitted, it can be switched off by the ICL engineer to enable the central processor to be run in datum-limit mode.

The size of a page is 1024 (1K) words. The number of pages in the core store will therefore be the same as the K size of the store, but some of the pages will be used by Executive and the operating system.

Programming for paging

Note: The word *segment,* as used in the rest of this chapter, has a specific significance in paging that is defined below.

Programs that are to make use of paging facilities must be written in 22AM. (This restriction applies only to paging mode, not to the central processor itself.)

For programming purposes, the total store, core store and backing store, is considered as being divided into segments, blocks, and words. A block consists of up to 1024 words of program or data, and a *segment* is up to 64 blocks.

Each block is stored on a separate page. A segment has no hardware representation, but merely provides a convenient means of grouping blocks for addressing purposes.

Addressing

Addressing is achieved by means of current page registers (see below) and a number of tables that are normally held in core store within the operating system.

TABLES

For each program there is a segment table, and for each segment used by the program there is a page table. The starting addresses of the segment tables of all programs stored in the machine are held in a program table. The program currently running has the starting address of its segment table copied into a special register to give a faster store access time.

The tables hold a number of addresses each of which is stored in one word. The two most significant bits of each word are used as control bits whose significance is described for each table below.

One use of the control bits is to indicate the replacement facility. When this setting is detected, the contents of the rest of the word will be interpreted as holding not the information required, but the address of the word where the required information is to be found. This facility is provided so that where sections of program or data are shared, tables associated with one program can reference tables associated with other programs. Tables within the set for any one program can also reference one another by means of this facility.

All the tables are set up and maintained by Executive and the operating system but are consulted during a store access entirely by hardware. Interrupts and software intervention will occur only if the information required is not in core or if a violation is discovered.

Program table

The program table contains an entry for each active program stored in the machine. The sequence of entries in the table is the sequence in which the programs were loaded. Bits 2 to 23 of each entry contain the address, m, of the first word of the chapter table associated with the program concerned. The control bits, 0 and 1, are interpreted as follows:

00 Replace the address m by the contents of address m

01 Segment table is 16 words long

10 Segment table is 32 words long

11 Segment table is 64 words long

Segment table

The segment table contains an entry for each chapter used by the program. The table length is then rounded up to 16, 32, or 64 words, the contents of any unused words being zero. Entries are in segment number sequence, so that the first entry is for Segment 0 and the nth entry for segment n-1. Each entry consists of an 18-bit address, m, in Bits 2 to 19, pointing to the first word of the page table associated with the segment, or a 22-bit replacement address in Bits 2 to 23. Bits 0 and 1 are interpreted as follows:

00 Replace the address m by the contents of address m

01 Page table is 16 words long

10 Page table is 32 words long

11 Page table is 64 words long

In the last three cases, Bit 23 is set equal to one if the page table concerned has been removed to backing store.

Page table

There is a page table for each segment used by the program. Entries in the page table are in block number sequence similarly to the entries in the segment table, and the size of the table is similarly rounded up to 16, 32, or 64 words. Each entry has a 12-bit page address, *m*, in Bits 2 to 13, or a 22-bit replacement address in Bits 2 to 23. Bits 0 and 1 are interpreted as follows:

00 Replace the address *m* by the contents of address *m*

01 Block not in core

10 Block in core and available

11 Block in core but not available.

Unless the replacement code is set i.e. Bits 0 and 1 are set to 00, any or all of the Bits 14 to 16 may be set equal to one with the following significance:

Bit 14 Block may be obeyed

Bit 15 Block may be read

Bit 16 Page may be written to.

These bits are known as the permission bits.

CURRENT PAGE REGISTERS

The current page registers, of which there are 16 on the 1906A, are used to store the page addresses of the blocks most frequently used. The contents of each C.P.R. is illustrated below.

Bits 0,1	2 - 13	14 - 16	17 - 20	21-23	24	25	26 - 37
Availability	Page address	Permission		Use		Lock-in	Segment and block number

Bits 0 to 16 correspond to those in the page table, and are copied when the C.P.R. is loaded.

The *use* bits are set equal to one to indicate any of the following occurrences:

Bit 21 Block has been obeyed

Bit 22 Block has been read

Bit 23 Page has been written to.

These bits are interpreted to decide whether, at any given time, the contents of the C.P.R. should be allowed to stand or whether they should be overwritten.

Loading current page registers

C.P.R.'s are loaded cyclically, so that at any time they generally contain the addresses of the last 16 different blocks referred to. The next C.P.R. to be loaded is determined by a four-bit counter which is stepped on cyclically; it is stepped past any C.P.R. whose *Lock-in* bit is set. This bit is set if:

1 The last operand store access was from that page;

2 The last instruction call was from that page;

3 The page contains the source or destination of a multi-operand (e.g. MOVE) order which is in progress;

4 It contains Program Block 0.

Action when store access required

All the C.P.R.'s are examined simultaneously to see if the required segment and block address are to be found in Bits 26 to 37. If so, the corresponding page address from Bits 2 to 13 is used to access the core address required. The appropriate use bit is then set.

If the required segment and block address is not found in the C.P.R., the segment table address for the program is retrieved from the special register in which it is held. The segment and page tables are then searched until the required address is found. The table entries are then copied into a C.P.R.

All these operations are carried out by hardware, accessing core store as required to read entries in the tables. At each stage checks are performed to see that the entry lies within the table. If any of these fail, or if the page table entry shows that the block is not available, or if the appropriate permission bit is not set, there is an interrupt and Executive is entered.

Further tables are kept to show the position of blocks on backing store. These tables are consulted by the operating system, not by hardware. There is also a store use table with a half-word entry for every page; when a C.P.R. is unloaded, its use bits are *or*-ed into the corresponding entry for the information of the operating system.

If the block required is found in a C.P.R., the whole process takes no longer than the datum-limit checks. Otherwise, three extra core cycles are required, to read the segment and page tables and to write away the use bits from the C.P.R. which is unloaded. Each replacement requires an additional store access.

If the block required is not in core, an interrupt occurs and the operating system is entered to fetch it from backing store. It will be necessary to clear a page in core in order to accept it, and the operating system's *page turning* routine will be entered to decide which page should be given up. Its choice will be based on records of store usage which it compiles with the help of the store use table. The contents of the page selected are written away to backing store, unless the use bits show that it has not been written to, in which case the existing copy on backing store is still valid and the page can simply be overwritten.

Programming restrictions on paged 1904A

The FORTRAN compilers for the paged 1906A optionally use the extended precision floating-point feature. The same compilers may be used for the 1904A, but the extended precision floating-point feature is not available. Extended precision floating-point may be performed by system-provided subroutines.

Chapter 9

Chapter 9 of this manual has been deleted.

Chapter 10 The 1901 to 1907 central processors

GENERAL

The 1901 to 1907 computers were the first in the 1900 Series to be announced. The machines range in size from the small but powerful 1901 to the large 1907.

The 1901, 1902, 1903, 1904 and 1906 are successively more powerful central processors. The 1905 is fundamentally similar to the 1904 but incorporates an autonomous floating-point unit. The 1907 bears the same relationship to the 1906.

Individual processors

Characteristics of individual processors are described below. The headings Store sizes, Store cycle time, Number of cabinets, Operator communication, Peripheral interfaces, Fixed-point operations and Floating-point operations apply to all processors. Further characteristics are described in alphabetical order under the heading Other facilities. The facilities given under this heading are the following:

Console Logging Punch

Mill Timer

Multiprogramming and Subprogramming

Peripheral Control Connector

Real Time Clock

Standard Interface Switching Unit

Store Access Control

If any of these headings does not appear under the description of an individual processor then the facility is not available with that processor.

THE 1901

The 1901 may have any one of three type numbers, 1901/1, 1901/2, 1901/3, according to the size of core store used.

Store sizes

4K, 8K or 16K words of core store may be fitted.

Store cycle time

In all cases the cycle time is 6 microseconds.

Number of cabinets

A single cabinet, $57 \times 25\frac{1}{2} \times 49$ inches, is required in all cases.

Operator communication

The 1901 is supplied as standard with a control panel of handswitches and lights for operator/Executive communication. A console typewriter can be used instead of the control panel provided that at least 8K of core store is fitted. However, the control panel must still be used to load Executive. On a disc system and certain magnetic tape systems a console typewriter is mandatory.

Peripheral interfaces

3 standard interface channels are provided with the basic processor; a maximum of 3 more can be fitted individually.

Fixed-point operations

Performed by hardware supplied as standard apart from multiplication, division, double-length shifts and binary to decimal/decimal to binary conversion, which are performed by extracode. Hardware to perform these operations, apart from conversion, is available as an optional feature.

Floating-point operations

These are optionally available in either extracode or hardware form.

Other facilities

STANDARD INTERFACE SWITCHING UNIT	Optionally available

THE 1902

The 1902 has only one type number, 1902/1.

Store sizes

4K, 8K, 16K or 32K words of core store may be fitted.

Store cycle time

In all cases the cycle time is 6 microseconds.

Number of cabinets

1 cabinet, 69 × $25\frac{1}{2}$ × 49 inches, is required, unless a 32K store is fitted, in which case a further cabinet, 22 × $25\frac{1}{2}$ × 49 inches, is necessary.

Operator communication

A console typewriter is supplied as standard.

Peripheral interfaces

8 standard interface channels are fitted as standard; this is the maximum number allowed.

Fixed-point operations

Performed by hardware supplied as standard apart from multiplication, division, double-length shifts and binary to decimal/decimal to binary conversion, which are performed by extracode. Hardware to perform these operations, apart from conversion, is available as an optional feature.

Floating-point operations

These are optionally available in either extracode or hardware form.

Other facilities

MULTIPROGRAMMING AND SUBPROGRAMMING

Limited facilities are available with a minimum 16K core store and EX2M Executive. The limited version of multiprogramming, known as dualprogramming, allows two programs to be run concurrently. For a full description, see page 60. The subprogramming facility allows a program to be divided into a maximum of three members.

STANDARD INTERFACE SWITCHING UNIT	Optionally available

THE 1903

The 1903 has only one type number, 1903/1.

Store sizes

8K, 16K or 32K words of core store may be fitted.

Store cycle time

1.8 microseconds for 8K and 16K stores and 2 microseconds for 32K stores.

Number of cabinets

1 cabinet, 69 × 25½ × 49 inches, is required, unless a 32K store is fitted in which case a further cabinet, 22 × 25½ × 49 inches, is required.

Operator communication

A console typewriter is supplied as standard.

Peripheral interfaces

8 standard interface channels are fitted as standard; this is the maximum number allowed.

Fixed-point operations

Performed by hardware supplied as standard apart from multiplication, division, double-length shifts and binary to decimal/decimal to binary conversion, which are performed by extracode. Hardware to perform these operations, apart from conversion, is available as an optional feature.

Floating-point operations

These are optionally available in either extracode or hardware form.

Other facilities

MULTIPROGRAMMING AND SUBPROGRAMMING

Limited facilities are available with a minimum 16K core store and EX2M Executive. The limited version of multiprogramming, known as dualprogramming, allows two programs to be run concurrently. For a full description, see page 60. The subprogramming facility allows a program to be divided into a maximum of three members.

STANDARD INTERFACE SWITCHING UNIT	Optionally available

THE 1904

The 1904 may have any of the type numbers 1904/1 to 1904/4 according to the store size and number of cabinets used.

Store sizes

16K or 32K words of core store may be fitted.

Store cycle time

2 microseconds in both cases.

Number of cabinets

Between 2 and 4 cabinets are required. The depth and height is in all cases the same, 34 × 60½ inches; the length is 97½, 145½ or 193½ inches depending on whether there are 2, 3, or 4 cabinets.

Operator communication

A console typewriter is supplied as standard.

Peripheral interfaces

The number of standard interface channels supplied depends upon the peripheral configuration. In ideal circumstances an absolute maximum of 23 standard interface channels may be fitted, of which 5 may be connected via P.C.C.s.

Fixed-point operations

Performed by hardware supplied as standard.

Floating-point operations

Floating-point extracodes are optionally available.

Other facilities

CONSOLE LOGGING PUNCH	Optionally available
MILL TIMER	Optionally available
MULTIPROGRAMMING AND SUBPROGRAMMING	Supplied as standard (up to 4 programs/members)
PERIPHERAL CONTROL CONNECTOR	Optionally available
REAL TIME CLOCK	Supplied as standard
STANDARD INTERFACE SWITCHING UNIT	Optionally available

THE 1905

The 1905 may have any of the type numbers 1905/1 to 1905/3 according to the store size and number of cabinets used.

Store sizes

16K or 32K words of core store may be fitted.

Store cycle time

2 microseconds in both cases.

Number of cabinets

Between 3 and 5 cabinets are required. The depth and height is in all cases the same, 34 × 60½ inches; the length is 145½, 193½ or 241½ inches depending on whether there are 3, 4 or 5 cabinets.

Operation communication

A console typewriter is supplied as standard.

Peripheral interfaces

The number of standard interface channels supplied depends upon the peripheral configuration. In ideal circumstances an absolute maximum of 23 standard interface channels may be fitted, of which 5 may be connected via P.C.C.s.

Fixed-point operations

Performed by hardware supplied as standard.

Floating-point operations

Performed by hardware supplied as standard.

Other facilities

CONSOLE LOGGING PUNCH	Optionally available
MILL TIMER	Optionally available
MULTIPROGRAMMING AND SUBPROGRAMMING	Supplied as standard (up to 4 programs/members)
PERIPHERAL CONTROL CONNECTOR	Optionally available
REAL TIME CLOCK	Supplied as standard
STANDARD INTERFACE SWITCHING UNIT	Optionally available

THE 1906

The 1906 may have either of the type numbers 1906/2 or 1906/3 according to the number of cabinets used.

A feature of the 1906 is the provision of 8 high speed registers which are used as accumulators, thereby increasing the speed of processing.

Store size

Between 32K and 256K words of core store may be fitted in modules of 32K.

Store cycle time

1.1 and 2.1 microseconds. The distance of core store cabinets from the central processing unit will influence cycle time.

Number of cabinets

4 or 6 power and logic cabinets are required, the dimensions being $193\frac{1}{2} \times 34 \times 60\frac{1}{2}$ inches in the former case and $289\frac{1}{2} \times 34 \times 60\frac{1}{2}$ inches in the latter case. Additionally, a core store cabinet, $49 \times 34 \times 60\frac{1}{2}$ inches, is required for every 32K of store.

Operator communication

A console typewriter is supplied as standard.

Peripheral interfaces

The number of standard interface channels provided depends on the peripheral configuration. A maximum of sixteen may be fitted, of which some may be connected via P.C.C.s. Additionally, one or two S.A.C.s, each capable of handling 2, 4 or 6 fast peripheral controls, may be fitted as an optional feature.

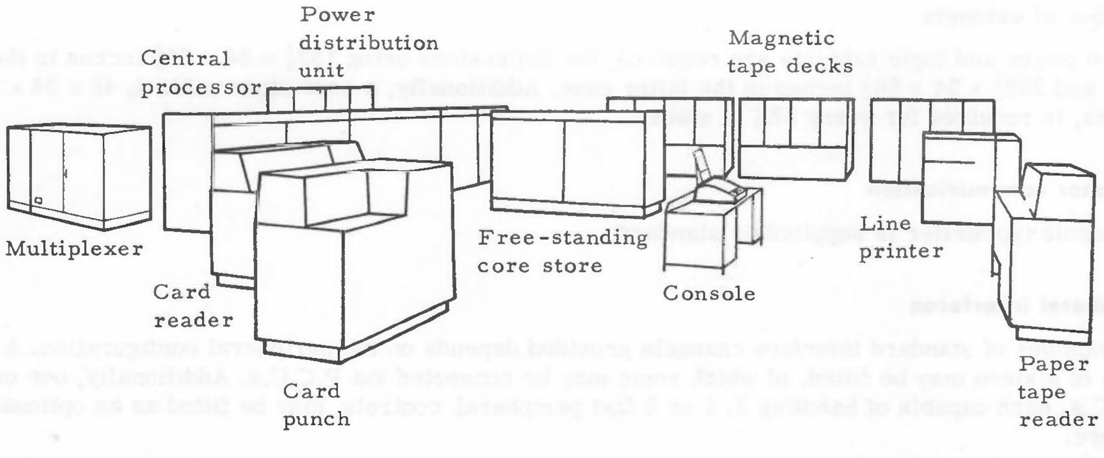
Fixed-point operations

Performed by hardware supplied as standard.

Floating-point operations

Floating-point extracodes are optionally available.

(Courtesy Ministry of Social Security)



1906 central processor system

Other facilities

CONSOLE LOGGING PUNCH	Supplied as standard
MILL TIMER	Supplied as standard
MULTIPROGRAMMING AND SUBPROGRAMMING	Supplied as standard (up to 16 programs/4 members)
PERIPHERAL CONTROL CONNECTOR	Optionally available
REAL-TIME CLOCK	Supplied as standard
STANDARD INTERFACE SWITCHING UNIT	Optionally available
STORE ACCESS CONTROL	Optionally available

THE 1907

The 1907 may have either of the type numbers 1907/1 or 1907/2 according to the number of cabinets used.

A feature of the 1907 is the provision of 8 high speed registers which are used as accumulators, thereby increasing the speed of processing.

Store size

Between 32K and 256K words of core store may be fitted in modules of 32K.

Store cycle time

1.1 and 2.1 microseconds. The distance of core store cabinets from the central processing unit will influence cycle time.

Number of cabinets

4 or 6 power and logic cabinets are required, the dimensions being $193\frac{1}{2} \times 34 \times 60\frac{1}{2}$ inches in the former case and $289\frac{1}{2} \times 34 \times 60\frac{1}{2}$ inches in the latter case. Additionally, a core store cabinet, $49 \times 34 \times 60\frac{1}{2}$ inches, is required for every 32K of store.

Operator communication

A console typewriter is supplied as standard.

Peripheral interfaces

The number of standard interface channels provided depends on the peripheral configuration. A maximum of sixteen may be fitted, of which some may be connected via P.C.C.s. Additionally, one or two S.A.C.s, each capable of handling 2, 4 or 6 fast peripheral controls, may be fitted as an optional feature.

Fixed-point operations

Performed by hardware supplied as standard.

Floating-point operations

Performed by hardware supplied as standard.

Other facilities

CONSOLE LOGGING PUNCH	Supplied as standard
MILL TIMER	Supplied as standard
MULTIPROGRAMMING AND SUBPROGRAMMING	Supplied as standard (up to 16 programs/4 members)
PERIPHERAL CONTROL CONNECTOR	Optionally available

REAL TIME CLOCK	Supplied as standard
STANDARD INTERFACE SWITCHING UNIT	Optionally available
STORE ACCESS CONTROL	Optionally available

Chapter 11 The E and F central processors

GENERAL

The E and F central processors were introduced into the 1900 Series to provide machines intermediate between the 1904 and 1907. There are thus the 1904E, 1904F, 1905E and 1905F. Each of these central processors can be paired with another of its kind when it becomes known as 1906E, 1906F, 1907E and 1907F respectively. These latter processors provide similar computing power to the 1906 and 1907 and incorporate the added safety factor in the dual processor concept. All of these processors are, of course, compatible with previously announced processors of the 1900 Series.

The principal difference between E and F processors with the same series number is the core store cycle time, the F processor being the faster. The 1905F and 1907F also have faster floating-point hardware than the 1905E or 1907E. The relationship between the series numbers is the same as for the 1904 to 1907 processors; that is to say, a 1905E or F is basically the same as a 1904E or F, but incorporates a floating-point unit, and similarly the 1907E or F differs fundamentally from the 1906E or F only in this respect.

Individual processors

The characteristics of individual central processors are described below under the headings Store size, Store cycle time, Number of cabinets, Operator communication, Peripheral interfaces, Fixed-point operations, Floating-point operations and Other facilities. The items that may appear under the heading Other facilities are, in alphabetical sequence.

Console Logging Punch

Hardware Accumulators

Mill Timer

Multiprogramming and Subprogramming

Peripheral Autonomous Control

Real Time Clock

Standard Interface Switching Unit

If any of the items does not appear in the description of a particular processor then it is not available with that processor.

Dual processors

The principles of dual processor operation can be found on page 3.

The store cycle time and store sizes available are the same as for the individual processors except that a minimum 64K of core store must be fitted. The fixed- and floating-point operation facilities are available in each of the processors (e.g. a 1907E or F has two floating-point units) and two console typewriters are provided for operator communication. Double the number of peripheral interfaces are available and double the number of cabinets are required. The features listed under Other facilities are available with each of the processors in the dual processor configuration. However, hardware accumulators, optionally available for 1904E and 1905E processors, are supplied as standard for 1906E and 1907E machines.

The appropriate number of store multi-access controls will be provided as standard with each configuration. The GEORGE 3 operating system must be used with dual processor configurations.

THE 1904E

Store size

32K, 48K, 64K, 80K, 96K, 128K, 160K, 192K or 256K words of core store may be fitted.

Store cycle time

1.8 microseconds; this will be slightly increased if a store extension unit (see below) is used.

Number of cabinets

1 or 2 power cabinets, 1 or 2 logic cabinets, and 1 core store cabinet for every 64K words of store are required. The dimensions of each cabinet are $49 \times 34 \times 60\frac{1}{2}$ inches. If more than one core store cabinet is required the remainder of the store is connected by means of a store extension unit, and a remote store unit is required for each extra cabinet. These devices facilitate the use of large core stores and are supplied as standard for each configuration that requires them. They require no extra space.

Operator communication

A console typewriter is supplied as standard.

Peripheral interfaces

6 standard interface channels are supplied as standard and up to 2 further groups of 6 are optionally available. Between 2 and 12 further channels may be provided as options via a P.A.C.

Fixed-point operations

Performed by hardware supplied as standard.

Floating-point operations

Floating-point extracodes are optionally available.

Other facilities

CONSOLE LOGGING PUNCH	Optionally available
HARDWARE ACCUMULATORS	Optionally available
MILL TIMER	Supplied as standard
MULTIPROGRAMMING AND SUBPROGRAMMING	Supplied as standard (up to 8 programs/4 members; up to 16 programs optionally available)
PERIPHERAL AUTONOMOUS CONTROL	Supplied as standard (if peripheral channels are ordered)
REAL TIME CLOCK	Supplied as standard
STANDARD INTERFACE SWITCHING UNIT	Optionally available

THE 1904F

Store size

Between 32K and 256K words of core store may be fitted in modules of 32K, but no 224K store is available.

Store cycle time

750 nanoseconds; this will be slightly increased if a store extension unit (see below) is used.

Number of cabinets

1 or 2 power cabinets, 1 or 2 logic cabinets and 1 core store cabinet for every 64K words of store are required. The dimensions of each cabinet are 49 × 34 × 60½ inches. If more than one core store cabinet is required the remainder of the store is connected by means of a store extension unit, and a remote store unit is required for each extra cabinet. These devices facilitate the use of large core stores and are supplied as standard for each configuration that requires them. They require no extra space.

Operator communication

A console typewriter is supplied as standard.

Peripheral interfaces

6 standard interface channels are supplied as standard and up to 2 further groups of 6 are optionally available. Between 2 and 12 further channels may be provided as options via a P.A.C.

Fixed-point operations

Performed by hardware supplied as standard.

Floating-point operations

Floating-point extracodes are optionally available.

Other facilities

CONSOLE LOGGING PUNCH	Optionally available
HARDWARE ACCUMULATORS	Supplied as standard
MILL TIMER	Supplied as standard
MULTIPROGRAMMING AND SUBPROGRAMMING	Supplied as standard (up to 8 programs/4 members; up to 16 programs optionally available)
PERIPHERAL AUTONOMOUS CONTROL	Supplied as standard (if peripheral channels are ordered)
REAL TIME CLOCK	Supplied as standard
STANDARD INTERFACE SWITCHING UNIT	Optionally available

THE 1905E

Store size

32K, 48K, 64K, 80K, 96K, 128K, 160K, 192K or 256K words of core store may be fitted.

Store cycle time

1.8 microseconds; this will be slightly increased if a store extension unit (see below) is used.

Number of cabinets

1 or 2 power cabinets, 1 or 2 logic cabinets, and 1 core store cabinet for every 64K words of store are required. The dimensions of each cabinet are 49 × 34 × 60½ inches. If more than one core store

cabinet is required all the store is connected by means of a store extension unit, and a remote store unit is required for each extra cabinet. These devices facilitate the use of large core stores and are supplied as standard for each configuration that requires them. They take up no extra space.

Operator communication

A console typewriter is supplied as standard.

Peripheral interfaces

6 standard interface channels are supplied as standard and up to 2 further groups of 6 are optionally available. Between 2 and 12 further channels may be provided as options via a P.A.C.

Fixed-point operations

Performed by hardware supplied as standard.

Floating-point operations

Performed by hardware supplied as standard.

Other facilities

CONSOLE LOGGING PUNCH	Optionally available
HARDWARE ACCUMULATORS	Optionally available
MILL TIMER	Supplied as standard
MULTIPROGRAMMING AND SUBPROGRAMMING	Supplied as standard (up to 8 programs/4 members; up to 16 programs optionally available)
PERIPHERAL AUTONOMOUS CONTROL	Supplied as standard (if peripheral channels are ordered)
REAL TIME CLOCK	Supplied as standard
STANDARD INTERFACE SWITCHING UNIT	Optionally available

THE 1905F

Store size

Between 32K and 256K words of core store may be fitted in modules of 32K, but no 224K store is available.

Store cycle time

750 nanoseconds; this will be slightly increased if a store extension unit (see below) is used.

Number of cabinets

1 or 2 power cabinets, 1 or 2 logic cabinets, and 1 core store cabinet for every 64 K words of store. The dimensions of each cabinet are 49 × 34 × $60\frac{1}{2}$ inches. If more than one core store cabinet is required the remainder of the store is connected by means of a store extension unit, and a remote unit is required for each extra cabinet. These devices facilitate the use of large core stores and are supplied as standard for each configuration that requires them. They take up no extra space.

Operator communication

A console typewriter is supplied as standard.

Peripheral interfaces

6 standard interface channels are supplied as standard and up to 2 further groups of 6 are optionally available. Between 2 and 12 further channels may be provided as options via a P.A.C.

Fixed-point operations

Performed by hardware supplied as standard.

Floating-point operations

Performed by hardware supplied as standard.

Other facilities

CONSOLE LOGGING PUNCH	Optionally available
HARDWARE ACCUMULATORS	Supplied as standard
MILL TIMER	Supplied as standard
MULTIPROGRAMMING AND SUBPROGRAMMING	Supplied as standard (up to 8 programs/4 members; up to 16 programs optionally available)
PERIPHERAL AUTONOMOUS CONTROL	Supplied as standard (if peripheral channels are ordered)
REAL TIME CLOCK	Supplied as standard
STANDARD INTERFACE SWITCHING UNIT	Optionally available

THE 1906E

See pages 113 and 114.

THE 1906F

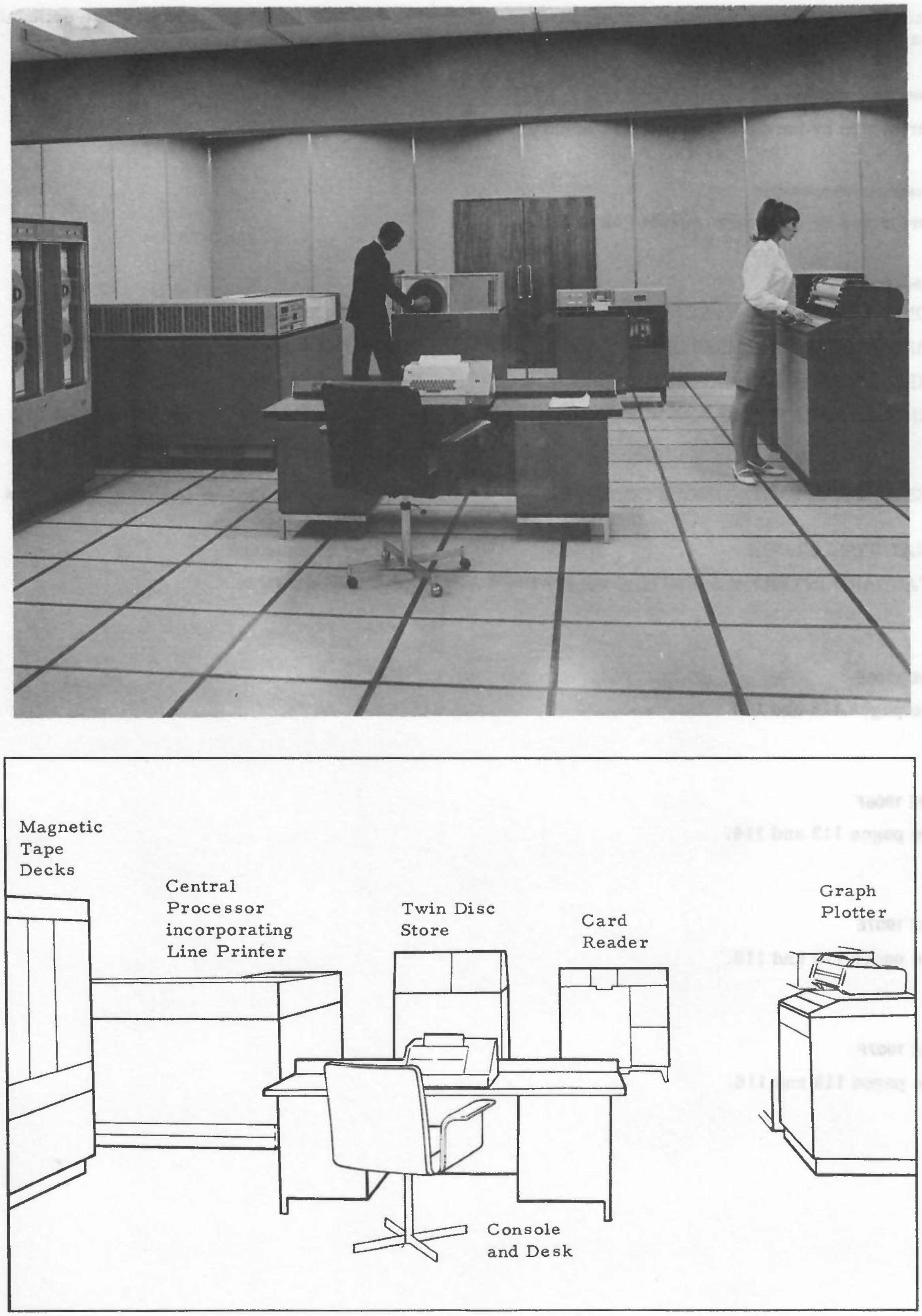
See pages 113 and 114.

THE 1907E

See pages 113 and 115.

THE 1907F

See pages 113 and 116.



1901A central processor system

Chapter 12 The A central processors

GENERAL

The A central processors are the latest to be added to the 1900 Series. These processors incorporate recent developments in technology, such as integrated microcircuits, and thus provide faster function times and more computing power without a corresponding increase in the size or cost of the machines. However, compatibility with other central processors in the 1900 Series has been maintained.

Individual processors

Characteristics of individual processors are described below. The headings Store size, Store cycle time, Number of cabinets, Operator communication, Peripheral interfaces, Fixed-point operations apply to all processors. Further characteristics are described under the heading Other facilities. The facilities given under this heading are in alphabetical sequence, the following:

Hardware Accumulators

Mill Timer

Multiprogramming and Subprogramming

Paging

Peripheral Autonomous Control

Peripheral Processing Unit

Real Time Clock

Standard Interface Switching Unit

Store Access Manager

If any of the items does not appear in the description of a particular processor then it is not available with that processor.

THE 1901A

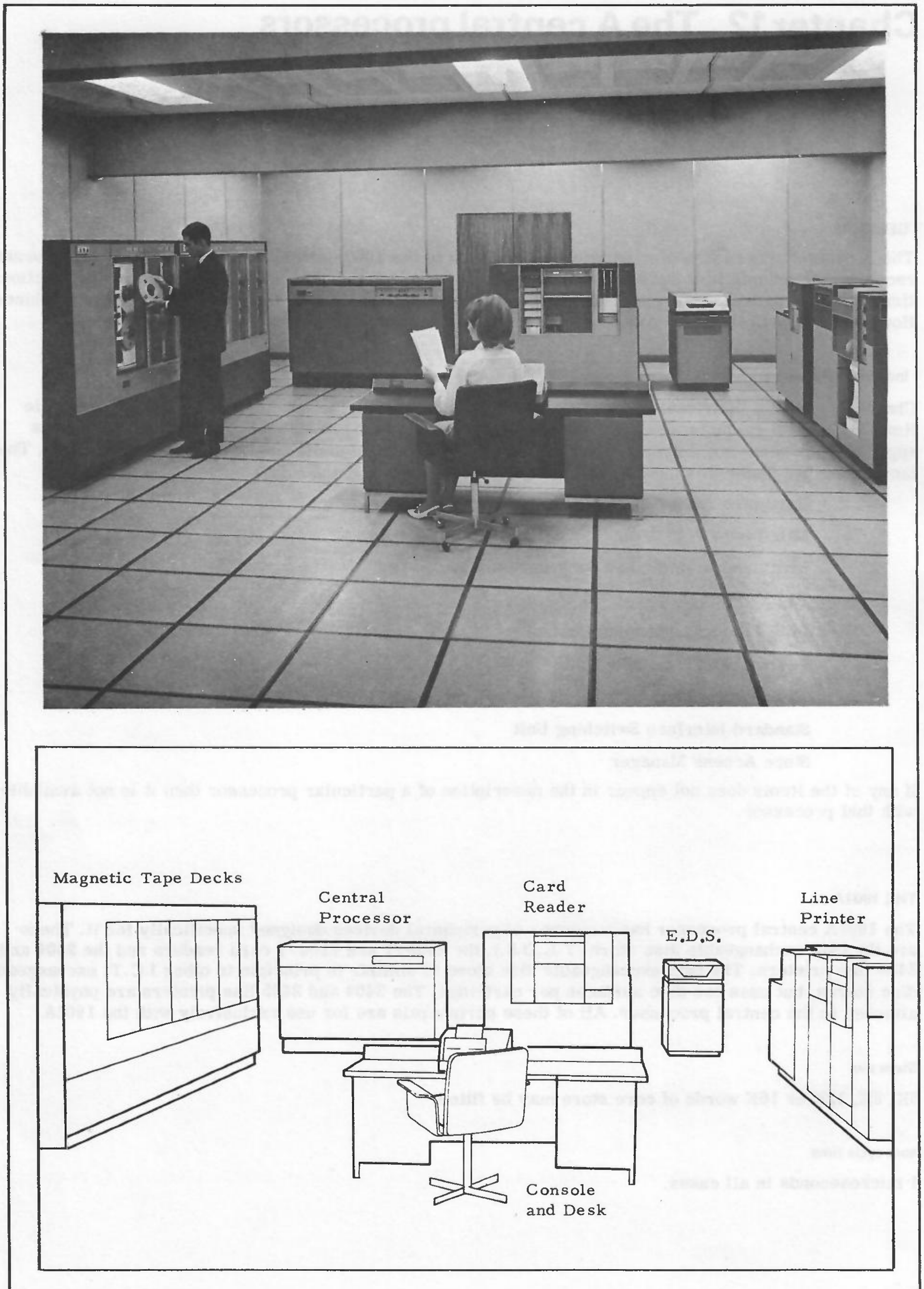
The 1901A central processor has a number of peripheral devices designed specifically for it. These are the twin exchangeable disc store (T.E.D.S.), the 2105/1 and 2106/1 card readers and the 2404 and 2405 line printers. The twin exchangeable disc store is similar in principle to other I.C.T. exchangeable disc stores, but uses two disc surfaces per cartridge. The 2404 and 2405 line printers are physically attached to the central processor. All of these peripherals are for use exclusively with the 1901A.

Store size

6K, 8K, 12K or 16K words of core store may be fitted.

Store cycle time

4 microseconds in all cases.



1902A central processor system

Number of cabinets

A single cabinet is required. If a 2404 or 2405 line printer is incorporated, the cabinet is L-shaped, the long sides of the L being 62 inches and $67\frac{1}{2}$ inches long. The height is 45 inches. If no 2404 or 2405 line printer is incorporated, the cabinet is approximately 62 × $25\frac{1}{2}$ × 45 inches, one $25\frac{1}{2}$ inch side being extended slightly where the line printer would otherwise be attached.

Operator communication

A control panel of handswitches and lights is supplied as standard. A console typewriter can be used for operator/Executive communication; however, the control panel must still be used to load Executive. A console typewriter option is available and is mandatory with magnetic tape and disc systems.

Peripheral interfaces

3 interface channels are provided for the attachment of peripherals specifically designed for the 1901A. That is, one channel for a card reader, one for a line printer, and one for up to 4 twin exchangeable disc stores. A single standard interface channel is provided with the basic processor, and up to 3 more standard interface channels may be added singly as optional features.

Fixed-point operations

Performed by hardware supplied as standard apart from multiplication, division, double-length shifts and binary to decimal/decimal to binary conversion. Hardware to perform these operations is available as an optional feature.

Floating-point operations

Optionally available in extracode or hardware form.

Other facilities

STANDARD INTERFACE SWITCHING UNIT	Optionally available

The 1902A

The 1902A is basically similar to the 1903A. The store capacity can be extended and the processor up graded to a 1903A in the field. There are two versions of the 1902A. The 1902A (2020) can be further enhanced by an Advanced Systems Feature (ASF). The 1902A (2015) is supplied with an Integrated Disc Control (IDC) as standard.

Store size

8K, 16K, 32K or 48K words of core store may be fitted to the 1902A (2020).

12K, 16K, 24K and 32K words of core store may be fitted to the 1902A (2015).

Store cycle time

6 microseconds for 1902A (2015), 3 microseconds for 1902A (2020).

Number of cabinets

One cabinet, 73 x $25\frac{1}{2}$ x 49 inches, is required up to 32K core stores: an additional cabinet is required for core stores of 48K.

A console typewriter is supplied as standard.

Peripheral interface

For the 1902A (2020), 4 standard interface channels are provided as standard and up to a further 4 may be fitted singly as options.

For the 1902A (2015), 4 standard interface channels are provided and up to a further 3 may be fitted singly as options.

Where Twin E.D.S. is specified, this will be connected via the IDC. The Integrated Twin E.D.S. Control will provide an interface in place of one of the 4 standard interfaces provided as standard.

Fixed point operations

For the 1902A (2020), performed by hardware supplied as standard apart from multiplication, division and binary to decimal/decimal to binary conversion. Hardware to perform these operations is optionally available with single programming Executives and mandatory with multiprogramming Executives.

For the 1902A (2015), performed by hardware supplied as standard.

Floating point operations

Optionally available in extracode or hardware form.

Other facilities

REAL TIME CLOCK	Optionally available
STANDARD INTERFACE SWITCHING UNIT	Optionally available
STORE ACCESS MANAGER	Supplied as standard
MILL TIMER	Optionally available

THE 1903A

Store size

16K, 32K, 48K, 64K, 96K or 128K words of core store may be fitted. If more than 32K of store is fitted the GEORGE 3 operating system may be used in which case a mill timer and real time clock are mandatory.

Store cycle time

1.5 microseconds

Number of cabinets

1 to 5, dependent on the size of the store. The basic cabinet dimensions are 73 × $27\frac{1}{2}$ × $49\frac{1}{2}$ inches, and each additional cabinet is 31 × $27\frac{1}{2}$ × $49\frac{1}{2}$. One additional cabinet is required for 48K or 64K core stores, two for 96K and three for 128K. The maximum processor size is, therefore, 197 × $27\frac{1}{2}$ × $49\frac{1}{2}$ inches.

Where P.A.C. is fitted to current processors, P.A.C. will be sited in an extension cabinet.

Operator communication

A console typewriter is supplied as standard.

Peripheral interface

4 standard interface channels are supplied as standard and up to 8 more may be provided singly as an optional feature. A P.A.C. incorporating 6 fast standard interface channels is optionally available, thus giving a maximum of 18 interfaces.

Fixed-point operations

Performed by hardware supplied as standard.

Floating-point operations

Optionally available in extracode or hardware form.

Other facilities

MILL TIMER	Optionally available (see store size)
MULTIPROGRAMMING AND SUBPROGRAMMING	Available with E3TM, E3DM (up to 4 programs/3 members) and E3DG Executives.
PERIPHERAL AUTONOMOUS CONTROL	Optionally available
REAL TIME CLOCK	Optionally available (see store size)
STANDARD INTERFACE SWITCHING UNIT	Optionally available
STORE ACCESS MANAGER	Supplied as standard

THE 1904A

Store size

32K, 64K, 96K, 128K, 192K, or 256K words of core store may be fitted.

Store cycle time

750 nanoseconds in all cases.

Number of cabinets

2 cabinets plus an extra cabinet for each 64K of store. The 2 basic cabinets together are 101½ × 24 × 54 inches and each core cabinet is 56 × 24 × 54 inches.

Operator communication

A console typewriter is supplied as standard.

Peripheral interfaces

6 standard interface channels are supplied as standard and up to 2 further groups of 6 interfaces are optionally available. A P.A.C. incorporating 4 fast standard interface channels is supplied as standard. Additionally a high speed standard interface channel, and between 1 and 8 further fast channels are optionally available.

Fixed-point operations

Performed by hardware supplied as standard.

Floating-point operations

Optionally available in extracode or hardware form.

Other facilities

HARDWARE ACCUMULATORS	Supplied as standard
MILL TIMER	Supplied as standard

MULTIPROGRAMMING AND SUBPROGRAMMING	Supplied as standard (up to 8 programs /4 members; up to 16 programs optionally available)
PAGING	Optionally available
PERIPHERAL AUTONOMOUS CONTROL	Supplied as standard (with 4 peripheral channels)
REAL TIME CLOCK	Supplied as standard
STANDARD INTERFACE SWITCHING UNIT	Optionally available

THE 1906A

The 1906A differs radically from less powerful processors in the 1900 Series, although compatibility is maintained, in that it has an interleaved store (see page 9) and allows instruction overlap. Moreover, all peripheral transfers are carried out autonomously, by means of a peripheral processing unit. All these facilities are supplied as standard.

Either the GEORGE 3 or GEORGE 4 operating system must be used with this processor.

Store size

64K, 128K, 192K, 256K, 384K or 512K words of core store may be fitted. The 64K and 128K stores are two-way interleaved; the 192K store is four-way interleaved up to 128K and thereafter two-way interleaved; the 384K store is four-way interleaved up to 256K and thereafter two-way interleaved; the 256K and 512K stores are four-way interleaved.

Stores larger than 512K may be fitted by special arrangement with ICL.

Store cycle time

750 nanoseconds in all cases.

Number of cabinets

10 cabinets plus one extra for every 128K or less of core store. The 10 cabinets are split into two blocks each measuring 27 feet $4\frac{1}{2}$ inches × 44 inches × 76 inches. Each core cabinet is 74 × 44 × 76 inches; a cooling bay 54 × 44 × 76 inches must be used with 1 or 2 store cabinets and a further cooling bay with 3 or 4 store cabinets.

Operator communication

A console typewriter is supplied as standard.

Peripheral interfaces

The basic processor is provided with 10 slow standard interface channels and 4 fast channels. The slow channels can be incremented to 16, 24 or 30 channels. The fast channels can be incremented to 8, 12 or 14; further, any or all of the blocks of high speed channels may be used for fast peripherals provided that 14 fast channels are already fitted. High speed channels can be supplied in 2 blocks of 2 and 1 single channel, so that there may be between 2 and 5 high speed channels.

Fixed-point operations

Performed by hardware supplied as standard.

Floating-point operations

Floating-point extracodes are optionally available. Hardware to perform extended precision floating-point operations, see page 126, is also available.

Other facilities

HARDWARE ACCUMULATORS	Supplied as standard
INSTRUCTION OPERAND COUNTER	Supplied as standard
MULTIPROGRAMMING AND SUBPROGRAMMING	Not applicable as a GEORGE operating system must be used
PAGING	Optionally available
PERIPHERAL PROCESSING UNIT	Supplied as standard
REAL TIME CLOCK	Optionally available
STANDARD INTERFACE SWITCHING UNIT	Optionally available.

Chapter 13 Other central processor features

GENERAL

This chapter provides either a description of the features mentioned in the three previous chapters or, where a feature has already been described in this manual, a reference to the relevant pages. The features are listed in alphabetical sequence.

COMMERCIAL COMPUTING FEATURE

See Fixed-point Operations below.

CONSOLE LOGGING PUNCH

The console logging punch consists of a small punch housed in the right-hand cupboard of the console table and attached to the cupboard door. The punch is connected to the console typewriter and will punch into paper tape every data or control character sent to the typewriter from the central processor or from the keyboard. Punching is in eight tracks, the last of which provides even parity.

The punch is invisible to the operator while the cupboard is shut, but the operator will be informed of the only condition he needs to rectify, paper low, by a flashing light on the console typewriter.

The punch can be fitted in the field. In most cases this will involve merely the removal of the right-hand cupboard door of the console table and its substitution by another door with the punch attached. Some early 1904/5 central processors, however, may have odd parity typewriters and insufficient electronics to drive the punch. In these cases a new typewriter and supplementary electronics will have to be installed.

CONSOLE TYPEWRITER SWITCH

Large processors in the 1900 Series are supplied with two console typewriters, the second to act as a spare in case of typewriter failure. The console typewriter switch facility, available on these processors, provides a means of switching control to the spare console typewriter when the on-line typewriter fails, and also provides for easier replacement of a faulty typewriter.

The hardware for this facility will fit into the right- or left-hand cabinet under the console desk, and will include the necessary cabling and sockets for connecting both typewriters. A rotary switch, available to the operator, will transfer control from one typewriter to the other.

If a console message is being output by the processor when the switch is operated, one or several characters may be lost. However, the switch is intended for emergency use only and, in such circumstances; it is likely that characters will have been lost or messages garbled already. In any event, operation of the switch will not affect in any way the operation of Executive, operating system or user program, provided no attempt is being made to use either typewriter as an input device when the switch is used.

DUALPROGRAMMING

See page 60.

EXTENDED MATHEMATICAL UNIT

See Fixed- and Floating-point Operations below.

FIXED-POINT OPERATIONS

Fixed-point multiplication and division, shift and conversion functions may be carried out in one of three ways on 1900 Series:

1 By extracode functions on the smaller basic processors.

2 By hardware circuits built in to the larger processors.

3 By additional hardware circuits as an optional extra on the smaller processors.

On the 1901/2/3, the optional extra is called the extended mathematical unit and the fixed-point version provides hardware circuits for fixed-point multiplication and division, and double length shifts. The commercial power of the processor is improved by 20%. On the 1901A it is called the commercial computing feature and provides hardware circuits for fixed-point multiply and divide, double length shifts and binary to decimal/decimal to binary conversion.

On 1902A/3A it is called the commercial computing feature and provides hardware circuits for fixed-point multiply, divide and binary to decimal/decimal to binary conversion.

The commercial computing feature when used on 1901A, 1902A or 1903A improves commercial performance by 80%.

When a floating-point feature is fitted to any of the processors not possessing hardware fixed point multiply/divide functions as standard, the fixed-point functions are automatically incorporated.

FLOATING-POINT OPERATIONS

Floating-point operations are normally carried out by extracodes unless:

1 The processor is a scientific processor, e.g. 1905, 1905E, 1905F, 1907, 1907E, 1907F.

2 The processor has a special floating-point feature fitted.

In the case of 1 above, these processors are simply their basic equivalents, i.e. 1904, 1904E/F, 1906, 1906E/F incorporating a floating-point unit.

(a) The 1905 floating-point unit is used on the 1905, 1905E and 1907E. It consists of an extra 11 rows of circuit boards which perform floating-point arithmetic autonomously while the processor may be carrying out fixed-point operations.

(b) The 1907 floating-point unit is autonomous and is similar to the 1905 unit and is used on the 1907, 1905F and 1907F. It has more overlap between floating-point instructions and is about twice as fast in operation as the 1905 unit.

In the case of 2 above, the floating-point unit consists of additional hardware circuits which carry out floating-point operations at a much higher speed than do extracodes.

The floating-point units fitted to various processors differ both in the speed and mode of operation and are as follows:

(a) An E.M.U. or extended mathematical unit may be fitted to the 1901/2/3 and is a separate unit incorporating hardware circuits for fixed-point multiply and divide, shift, normalize and floating-point arithmetic.

(b) The 1902A/3A scientific computing feature consists of additional hardware circuits providing fixed-point multiply and divide and binary to decimal/decimal to binary conversion, and floating-point arithmetic functions.

(c) The 1904A floating-point unit consists of additional hardware circuitry and is similar in performance to the 1907 unit.

(d) The 1906A extended precision floating-point unit includes circuitry to perform both single length and extended precision operations and is about three times as fast as the 1907 unit.

Extended precision floating-point facilities

This section describes the operation of the 076 and 130 to 137 instructions in an extended precision floating-point environment. At present, extended precision floating-point facilities are available for the 1906A only.

For the format of an extended precision floating-point number in store, see page 22.

NOTATION

The following notation is used in this section in addition to the notation defined on page 28.

Symbol	Meaning
a	The contents of the 48-bit standard precision floating-point accumulator.
a^*	The contents of the accumulator extension for extended precision floating-point. The accumulator extension is 37 bits in length and corresponds to Bits 49 to 71 and Bits 73 to 86 of an extended precision floating-point number in store. The extended precision floating-point unit ignores Bits 48, 72 and 87 to 95 on input and sets them equal to zero on output.
a:	The combination of a and a^*.
arg:	The contents of the signed extended precision floating-point argument. Its 75 bits correspond to Bits 0 to 23, Bits 25 to 38, Bits 40 to 71 and Bits 73 to 86 of an extended precision floating-point number in store.
e	The exponent of the value held in A after rounding and/or normalising if applicable. If *FOV* is set then e is always considered to be positive.
FOV	A binary condition introduced for the purpose of explanation. It is set equal to 1 if and only if floating-point overflow occurs during the current instruction. Otherwise it equals zero.
FOVR	The contents of the floating-point overflow register.
FUN	A binary condition introduced for the purpose of explanation. It is set equal to 1 if and only if floating-point underflow occurs during the current instruction. Otherwise it equals zero. The variable limit register controls the point at which underflow occurs.
FUNR	The contents of the floating-point underflow register.
$n24$	The contents of Bit 24 of the operand addressed by the current instruction.
n::	The content of locations *DN* to *DN*+3.
VL	The value of the variable limit.
VLR	The contents of the variable limit register.

ACCURACY

In general, to attain maximum accuracy of arithmetic floating-point operations, the initial operands must be normalised or be floating-point zero.

1 ROUNDED RESULTANT OPERAND The machine result of an arithmetic operation producing a rounded resultant operand is the normalised machine floating-point number nearest the true result of the operation. Where there are two such numbers, the larger is taken.

2 UNROUNDED RESULTANT OPERAND The machine result of an arithmetic operation producing an unrounded resultant operand is the greatest normalised machine floating-point number not greater than the true result of the operation.

The end sequence of operations for instructions requiring rounding always follows the pattern: normalise, round, renormalise. For addition, subtraction, division and multiplication, the length of the argument used in the first normalise sequence equals the nominal argument length plus two bits.

When un-normalised operands are used, the accuracy will be that implied above. In the case where an un-normalised divisor is used in division and

$$|\text{dividend argument}| > |\text{divisor argument}|$$

or

$$\text{dividend argument} = \text{divisor argument}$$

FOVR is set and the arithmetic result is indeterminate.

OPERATIONS SUMMARY

The table on page 126.2 defines the 076 and 130 to 137 instructions for extended precision floating-point. For overflow and underflow see the table on page 126.3.

Function	Function extension (that is, X field of instruction)							
	$X = 0$	$X = 1$	$X = 2$	$X = 3$	$X = 4$	$X = 5$	$X = 6$	$X = 7$
076	Branch if $arg{:} = 0$	Branch if arg: $\neq 0$	Branch if $arg{:} \geqslant 0$	Branch if $arg{:} < 0$	Branch if $FOVR$ clear.	Branch if $FOVR$ set.	Branch if $FUNR$ clear.	Branch if $FUNR$ set.
130	$a' = float(n{:})$ $N(a{:}), AU$ $a^{*\prime}=0$	VLR' = Bits 40 to 47 of $n{:}$ UB	Undefined	Undefined	Undefined	Undefined	Undefined	Undefined
131	$n{:}' = fix(a)$	Undefined	Undefined	Undefined	Undefined	Undefined	Undefined	Undefined
132 (★denotes +) 133 (★denotes –) 134 (★denotes ×) 135 (★denotes ÷)	$a'=a \bigstar n{:}$ $N(a{:}), R(a{:}),$ $N(a{:}), AU$ $a^{*\prime}= 0$	$a{:}'=a{:} \bigstar n{:}$ $N(a{:}), R{:}(a{:}),$ $N(a{:}), AU, UB$	$a{:}'=a{:} \bigstar n{::}$ $N(a{:}), R{:}(a{:}),$ $N(a{:}), AU, UB$	$a{:}'=a{:} \bigstar n{::}$ $N(a{:}), AU, UB$	$a'=n{:} \bigstar a$ $N(a{:}), R(a{:}),$ $N(a{:}), AU$ $a^{*\prime} = 0$ (As $X = 0$ but with operands separated by ★interchanged)	$a'=n{:} \bigstar a{:}$ $N(a{:}), R{:}(a{:}),$ $N(a{:}), AU, UB$ (As $X = 1$ but with operands separated by ★interchanged)	$a{:}'=n{::} \bigstar a{:}$ $N(a{:}), R{:}(a{:}),$ $N(a{:}), AU, UB$ (As $X = 2$ but with operands separated by ★interchanged)	$a{:}'=n{::} \bigstar a{:}$ $N(a{:}), AU, UB$ (As $X = 3$ but with operands separated by ★interchanged)
136	$a' = n{:}$ AU $a^{*\prime} = 0$	$a{:}' = 0$	$a{:}' = n{::}$ AU, UB	$a^{*\prime} = 0$	$a' = a{:}$ $N(a{:}), R(a{:}),$ $N(a{:}), AU$ $a^{*\prime} = 0$	Undefined	Undefined	Undefined
137	$n{:}' = a$	$n{:}' = a$ $a' = 0$	$n{:}' = a^*$	$n{:}' = a^*$ $a^{*\prime} = 0$	$N(a{:})$ $a{:}' = -a{:}$ $N(a{:}), AU, UB$ $n{:}' = a'$	$a{:}' = 0$ $n{:}' = 0$	Undefined	Undefined

The symbols used in the table indicate the actions that occur when an instruction is obeyed. A key is given below. The order of these symbols corresponds to the order in which an action occurs. For example, the effect of performing a 137/4 instruction is as follows, in the specified order:

1 The contents of *a:* are normalised.

2 *a:'* = –*a:*

3 The contents of *a:* are normalised.

4 The action on underflow occurs.

5 The use bit is set.

6 *n:'* = *a'*

Key to operation mnemonics

Mnemonic	Meaning
AU	Action takes place on underflow, see *Variable limit register* section below.
fix(*a*)	Convert the floating-point number in *A* to mid point form.
float(*n:*)	Convert the mid-point number in locations *DN* and *DN* + 1 to floating-point form.
N(*a:*)	The contents of *a:* are normalised.
R(*a:*)	The contents of *a:* are rounded as a normal precision quantity, that is, 38 bit argument.
UB	The use bit in the extended precision floating-point unit is set. Once the program member has used any instruction that sets the use bit, its Words 14 and 15 are used by the operating environment as a dump for *a** and *VLR*, and can no longer be used directly by the program member.

OVERFLOW AND UNDERFLOW CONDITIONS SUMMARY

The following table defines the state of *V, FOVR, n*24 and *FUNR* after an 076 or 130 to 137 instruction has been obeyed. A hyphen denotes that an entry is not applicable. The sign ∪ denotes an 'inclusive logical or' operation. For example, the state of *V* after an 076 instruction is given by *V'* = *V* ∪ *FOVR*. Thus, if *V* = 1 and *FOVR* = 0 before the instruction has been obeyed, *V'* = 1 ∪ 0 = 1.

Instruction			*V'* =	*FOVR'* =	*n*24' =	*FUNR'* =
Function	Valid *X*	Mnemonic				
076	0 to 3	BFP	*V* ∪ *FOVR*	*FOVR*	–	*FUNR*
	4 to 7	BFP	*V*	*FOVR*	–	*FUNR*
130	0	FLOAT	*V*	*n*24	*n*24	*FUN*
	1		*V*	*FOVR*	*n*24	*FUNR*
131	0	FIX	*V* ∪ *FOVR* ∪ *FOV*	*FOVR*	0	*FUN*
132	0 to 7	FAD	*V*	*FOVR*∪*n*24∪*FOV*	*n*24	*FUN*
133		FSB				
134		FMPY				
135		FDVD				
136	0	LFP	*V*	*n*24	*n*24	*FUN*
	1	LFPZ	*V*	0		0
	2		*V*	*n*24	*n*24	*FUN*
	3		*V*	*FOVR*	–	*FUN*
	4		*V*	*FOVR* ∪ *FOV*	–	*FUN*
137	0	SFP	*F* ∪ *FOVR*	*FOVR*	*FOVR*	*FUN*
	1	SFPZ	*V* ∪ *FOVR*	0	*FOVR*	0
	2		*V*	*FOVR*	0	*FUN*
	3		*V*	*FOVR*	0	*FUN*
	4		*V* ∪*FOVR*∪*FOV*	*FOVR* ∪ *FOV*	*FOVR* ∪ *FOV*	*FUN*
	5		*V*	0	0	0

VARIABLE LIMIT REGISTER

The variable limit register is an eight bit register in the extended precision floating-point unit which can be used to control the limiting value of the exponent for which underflow occurs. It also controls the action subsequently taken if underflow occurs.

The *VLR* register is set by means of 130/1 instruction and its content corresponds to Bits 40 to 47 of *n*: for the instruction; Bit 45 is reserved. The remainder of *n*: is ignored. The effective value of the variable limit, denoted by *VL*, has a range of 0 to 248 in steps of 8; this value corresponds to Bits 40 to 44 of *n*:, each bit having a value of 2^{47-b} where *b* is the bit number. The limiting value for which the underflow register is set also depends on the state of Bits 46 and 47 of *n*:, see the table below.

The initial state of the *VLR* register at completion of a program load is equivalent to that produced by obeying a 130/1 instruction with *n*: = 0.

The setting of the *VLR* register can affect the actions of instructions 130, *X* = 0 and *X* = 1; 132 to 135; 136, *X*= 0, *X* = 1 and *X* = 2; 137, *X* = 1 and *X* = 5, only.

The following table defines the limiting values for which underflow occurs.

n: of last 130/1 instruction		Set *FUN* if	Set *a*:' = 0 (except *FOVR*)
Bit 46	Bit 47	*e* + 256 <	if *e* + 256 <
0 or 1	0	0	0
0	1	*VL*	0
1	1	*VL*	*VL*

FLOATING-POINT UNIT

See Floating-point Operations above.

INSTRUCTION OPERAND COUNTER

The instruction operand counter replaces the mill timer in the 1906A processors. Therefore instruction operands, and not clock pulses, are counted.

INTERLEAVING

See page 9.

MILL TIMER (PROGRAM TIMER)

This device provides clock pulse counting facilities. It consists of a hardware register that is incremented by one for every central processor clock pulse (see page 15) while the central processor is in object program mode.

Clock pulses occurring during hesitations or while the central processor is in Executive mode are not included.

At the beginning of every interrupt, Executive increments a word in the Executive area by the contents of the hardware register, which is then reset to zero. Executive maintains counters for each program in the machine.

The mill time printed on the console typewriter is the number of millions of clock pulses (not millions of microseconds) counted for the particular program.

MULTIPLY/DIVIDE UNIT

See Fixed-point Operations above.

MULTIPROGRAMMING

See page 59.

OPERATOR COMMUNICATION

See page 2.

PAGING

See Chapter 8.

PERIPHERAL AUTONOMOUS CONTROL

See page 52.

PERIPHERAL CONTROL CONNECTOR

See page 51.

REAL TIME CLOCK

This device causes the computer time to be typed out on the console log at intervals of approximately one minute. The time should theoretically always be printed at intervals of exactly one minute on the minute, but two kinds of delay may occur. If certain operations are being performed when the minute expires the interrupt will not be attended to immediately. Furthermore, when the message is generated it joins a queue of messages waiting to be output; the queue will delay the appearance of the time message. The first kind of delay is catered for in the message, the time printed being the computer time to the nearest second at the moment that the message was generated. Thus a console log may show

```
    12/07/00
```

other messages

12/08/03

The real time clock also provides the time in response to a'165 N(M) = 2 instruction. The correct time must be loaded into Executive before the real time clock can be used.

REMOTE STORE UNIT

See page 13.

SCIENTIFIC COMPUTING FEATURE

See Floating-point Operations (page 126).

SLOW HESITATION CONTROL

See page 47.

STANDARD INTERFACE SWITCHING UNIT

See page 53.

STORE ACCESS CONTROL

See page 52.

STORE ACCESS MANAGER

See page 53.

STORE EXTENSION UNIT

See page 13

STORE MULTI-ACCESS CONTROL

See page 5.

SUBPROGRAMMING

See page 61.

Index

15AM 26
22AM 26
1901 105
1901A 119
1902 106
1902A 121
1903 107
1903A 122
1904 107
1904A 123
1904E 114
1904F 114
1905 108
1905E 115
1905F 116
1906 109
1906A 124
1906E 117
1906F 117
1907 111
1907E 117
1907F 117

Access time of store 5
Accumulators 11
Address modes 26
Allocation of store 9
Arithmetic unit 15
Availability see individual processor description

Binary programs 81
Branch modes 26

Carry 23
C.C.F. see commercial computing feature
Character counter modifier 20
Characters, storage of 19
Clock pulse counter see mill timer
Clock pulses 15
Clock, real time 127
Commercial computing feature 125
Console logging punch 125
Console typewriter 2
Console typewriter switch 125
Control panel, operator's 2
Control unit 15
Core store 5
Counter modifier words, character 20
word 20
C.P.R. see current page register
Current page registers 101
Cycle time of store 5

Data, representation of 19
DBM 26
Dual processor configurations 2,113
Dualprogramming 60

EBM 26
E.M.U. see extended mathematical unit
Extended mathematical unit 125,126
Extended precision floating-point 126

Fixed-point numbers 20
Floating-point numbers 21
Floating-point instructions 26
Floating-point unit 126

Hesitations 47

Interleaving of store 9
Interprocessor buffer 4

Members of a program see subprogramming
Mill, the 16
Mill timer 127
Multiply/divide unit see extended mathematical unit (fixed-point)
Multiprogramming 4, 59

Numbers, storage of 20,21

Operator's control panel 2
Overflow 23

P.A.C. see peripheral autonomous control
Paging 99
Parity, bit 1
checking 9
P.C.C. see peripheral control connector
Peripheral autonomous control 52
Peripheral control connector 51
Peripheral processing unit 53
P.P.U. see peripheral processing unit
Program timer see mill timer

Real time clock 127
Registers, A 16
B 16
current page 101
datum and limit 16
F 16
M 16
N 16
P 16
X 16

Regular interrupt feature see real time clock
Remote store unit 13
Request block 81
Request slip see request block

S.A.C. see store access control
S.A.M. see store access manager
Scientific computing feature 128
S.I.S.U. see standard interface switching unit
S.M.A.C. see store multi-access control
Standard interface 45
Standard interface switching unit 53
Store access, control 52
- manager 53
- mechanism 5
- time 5

Store cycle time 5
Store extension unit 13
Store interleaving 9
Store, layout of object program in 10
- entry parameters 11
- lower data 12
- program 12
- reserved 11,12
- upper data 12

Store multi-access control 5
Store sizes see individual processor description
Store speed 5
Subprogramming 61

Word, character 19
- counter modifier 19
- definition of 1
- pure binary 19