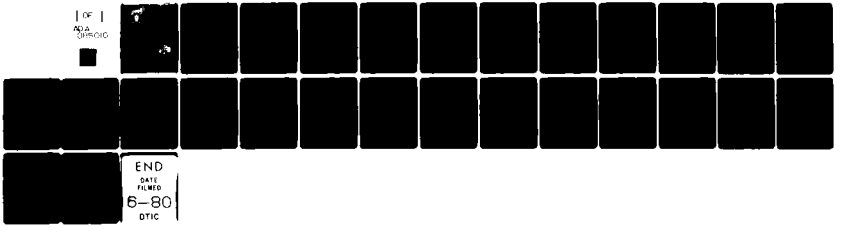
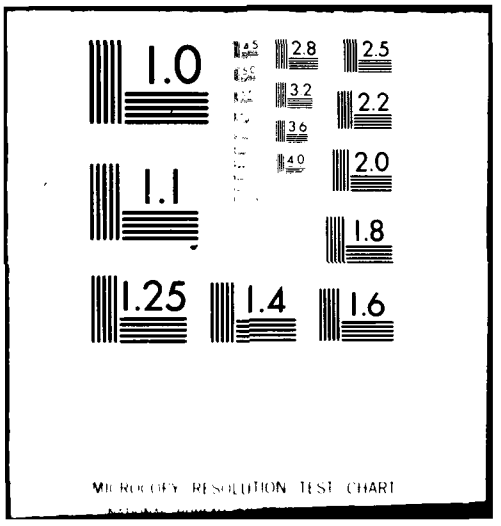
AD-A085 010 ROYAL SIGNALS AND RADAR ESTABLISHMENT MALVERN (ENGLAND) F/G 9/2
AN INTRODUCTION TO THE FLEX COMPUTER SYSTEM,(U)
OCT 79 J M FOSTER, C I MOIR, I CURRIE
UNCLASSIFIED RSRE-79016 DRIC-BR-73084 NL



END
DATE FILMED
6-80
DTIC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
MICROCOPY RESOLUTION TEST CHART

An Introduction to the FLEX Computer System

This report describes the background to the development of the FLEX system, which is a novel, high performance, multi-computer system being built at RSRE. FLEX has an instruction set which is oriented towards compiler generation of code, and which aids in security and parallel processing. The peripherals are controlled by a subsidiary computer. The inter-computer communication is provded by a packet switching network which is fast enough to allow sharing of peripherals and backing store. This report gives a high level description of the FLEX processor and memory, the peripheral control subsystem and COMFLEX, the communications subsystem. The report gives an overview of the processor architecture at the macro-code level. It describes the operating system and user program facilities provided on the machine. The report concludes by summarising the current state of the development of the FLEX computer system, and indicating the intended short term developments of the system.

Authors:

Dr. J M Foster, C I Moir, I F Currie,

J A McDermid, P W Edwards, J D Morison,

C H Pygott

Date:

October 1979

Contents

Introduction

The most salient features of the FLEX computer are described in section 1. An overview of the computer hardware is given in section 2 and more detailed descriptions of the processor and memory, the peripheral control subsystem, and the communications subsystem are in sections 3, 4 and 5. Section 6 gives a description of the processor architecture at the macro - code level, and section 7 describes the software, in particular the operating system kernel, provided on the machine.

1) The salient features of FLEX

1.1) Introduction

We can divide these features into three areas. First, FLEX has a high level language oriented instruction set, implemented in micro - code. This instruction set is of interest in security, in the use of parallel processes and as a target language for compilers. Especially important is the fact that procedures can be treated as values, in the conventional sense. Second, FLEX computers can be interconnected by means of a packet switching network sufficiently fast to match disc transfer speeds. The structure of these packets is used throughout the software. Third, the peripherals are controlled by a susidiary computer based on the Intel Multibus, to simplify the introduction of new peripherals. This computer is also used to diagnose the main computer for purposes of maintenance.

1.2) The instruction set

The allocation of storage in the FLEX memory (the memory of the main computer) is under the control of the micro - code. It can be allocated in pieces of any size from one word up to 1Mbyte. Every data byte has a tag bit. This tag bit does not enter into the arithmetic, but it can be tested and set by the micro - code. By using this tagged architecture, the micro - code can ensure that store references are different from other data, and so control which operations are legal. For example, an integer can not be added to a reference, nor can it be used to address the store. It is also possible to distinguish between various types of entity in the store. Procedures are identified as such and can only be obeyed, not read. Likewise arrays can not be obeyed. All reading and writing is checked to make sure that it is carried out within the bounds of the piece of store which is specified. Hence we can can be sure that it is impossible to stray outside the range of things which can properly be addressed. The same methods are used to check references to disc. All these checks are carried out, as relevant, on every instruction without exception. This provides a protection structure equivalent to that of capability machines, in a more elegant and efficient manner.

The store is garbage collected, that is blocks of store which can no longer be used are reclaimed at suitable moments and the wanted store is compacted. This can be carried out so quickly that on a 128k word machine it is imperceptible to the users. Hence store can be freely allocated and abandoned, and this makes the use of store by parallel processes easy to organise. The micro - code implementation of the change of environment on entering or leaving procedures or on changing processes, makes these operations cheap.

An RS Algol 68 [1],[2] compiler has been produced for this instruction set. For a given program, the code produced by

this compiler is half the size (in terms of words of memory occupied) of that produced by the Algol68R system on the ICL 1900 [2]. This instruction set arose out of research into high level language design being undertaken at RSRE [3].

1.3) The COMFLEX network

Both the main and subsidiary computers can be attached to nodes of the COMFLEX network, which provides very fast, packet switched, communications. Since COMFLEX can keep up with disc tranfer rates, it is not necesary for the main computers to use their local discs. They can use discs attached to other nodes in the network without degrading the rate of transfer. This network can also be used to transfer packets between the other peripherals and main computers. Hence any computer can use any peripherals, and this is especially interesting for the VDUs, since it means that a user can be atached to any of the machines on the network, without penalty.

2) An Overview of the FLEX computer hardware

2.1) Introduction

The FLEX computer hardware comprises three subsystems which can be interconnected to produce distributed computer systems to satisfy particular requirements. The three subsystems provide the three main functions required of a distributed computer system:

- processing of user programs
- peripheral control
- communications

It is possible to produce a single computer system without using the communications subsystem, as shown in figure 2.1. A simple triple computer system which could be constructed from these subsystems is shown in figure 2.2. This configuration is designed to meet the requirements for the RSRE computing service.

An overview of the three subsystems is given here and they are described in more detail in sections 3, 4 and 5. The processor architecture at the macro - code level is described in section 6.

2.2) User Program Processing

User programs are held in the FLEX memory and are executed by the FLEX processor [4]. The processor design and memory organisation are sympathetic to the requirements of High Level Languages (HLL), thus the fast and efficient execution of user programs written in HLL is possible.

The memory data words are 32 bits wide. The memory is byte addressed and it is protected by parity bits on each 8 bit data byte. In the current implementation the maximum memory size is 1Mwords. The memory has some sophisticated addressing modes which are conducive to efficient store usage and fast data access. Each memory byte also carries a tag field which is used to facilitate the memory management. The memory contains a high speed parallel interface to the communications subsystem.

The processor is microprogrammed, that is, the instruction set recognised by the processor is determined by a program (more strictly a mico - program), not directly by the hardware. Peripherals are controlled separately (by the peripheral subsystem) so the processor is largely free of the time constraints of peripheral handling, which allows considerable scope in designing the macro - instruction set. A micro - programmed implementation of a macro - instruction set suitable for use with high level languages has been developed.

The processor data word is 24 bits wide. The addressing modes of the memory allow the processor and memory to work together efficiently, despite the apparent incompatibility in word length. Since the memory is byte addressed the main significance of the word width is that it specifies the maximum number of data bits (32) which can be transferred from or to the memory in any store operation. The arithmetic operations in the processor use 24 bits for integers and 48 bits for floating point numbers. Hardware assisted floating point multiply and divide are provided.

2.3) Peripheral Control

The Peripheral Control Subsystem has four primary functions. It operates as a disc controller and can handle up to 4 Calcomp Trident (or equivalent) discs of 25, 50 or 80 Mbytes capacity. It controls low speed peripherals including VDUs and a lineprinter, and it provides some industry - standard general purpose interfaces (e.g. CCITT V24). It controls COMFLEX, the communications subsystem. It provides micro - program loading facilities and diagnostics for the FLEX processor. The peripheral control subsystem is also responsible for the initialistion of all three subsystems.

The peripheral control subsystem comprises four processors and two special interfaces, to COMFLEX and the discs respectively, communicating via a bus. The subsystem is itself modular and can be configured to perform various subsets of its primary functions or to incorporate interfaces to other deivces. If a system without discs were configured an alternative medium for storing the initialisation information, such as a floppy disc, might have to be provided (dependent on the system configuration).

2.4) Communications - COMFLEX

The Communications Subsystem is the COMFLEX packet switch. In a particular system the COMFLEXes are interconnected to provide fast and reliable data transmission between the peripheral control and processing subsystems. COMFLEX enables any subsystem to communicate with any other subsystem. COMFLEX can transmit data at disc speed, and its operation is transparent to application programs running on the FLEX processors, making viable the separation of the main processors and the peripheral control subsystems. The speed and transparency of COMFLEX allow the main processors to share peripherals and backing store. The communications medium is resilient to communications line and subsystem failures.

COMFLEX controls up to eight incoming lines and up to eight outgoing lines at a maximum data rate of 2.5 Mbytes per line. It contains logic for the routing of packets, for flow control and for error detection and recovery.

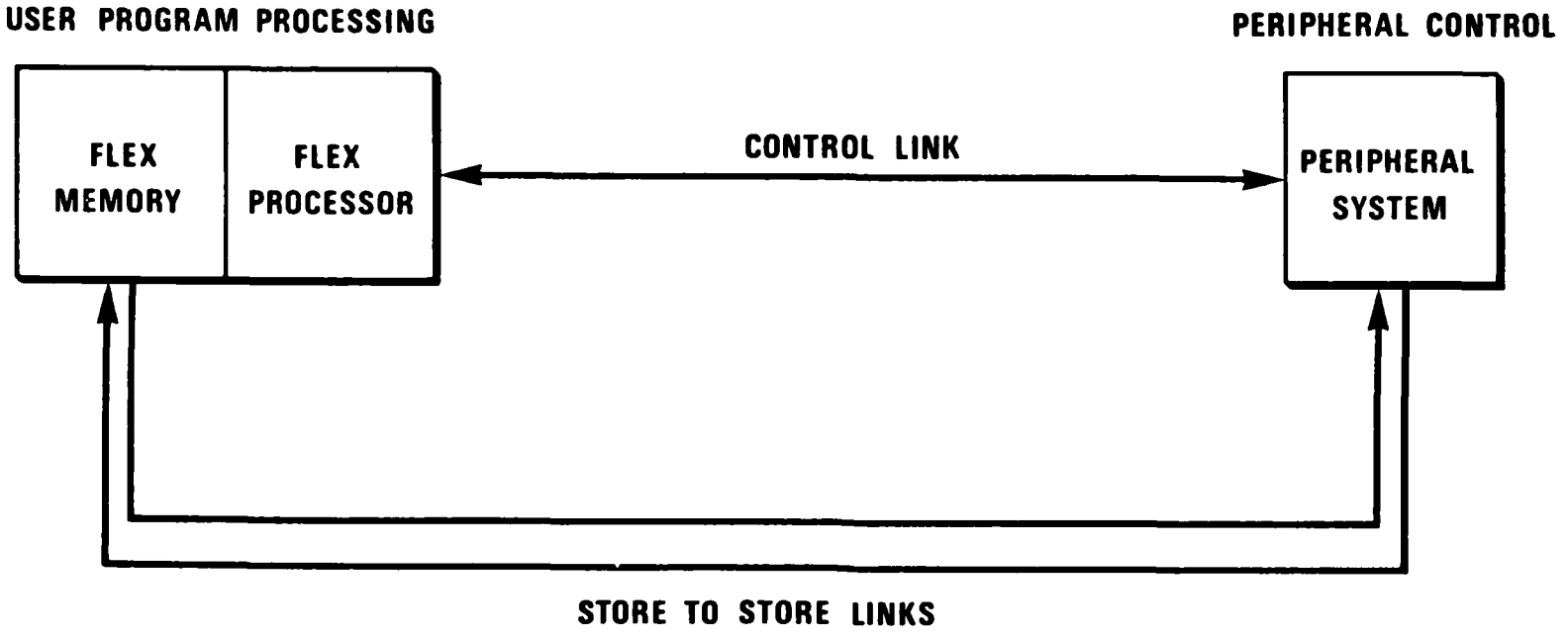


FIGURE 2·1 SINGLE COMPUTER FLEX CONFIGURATION

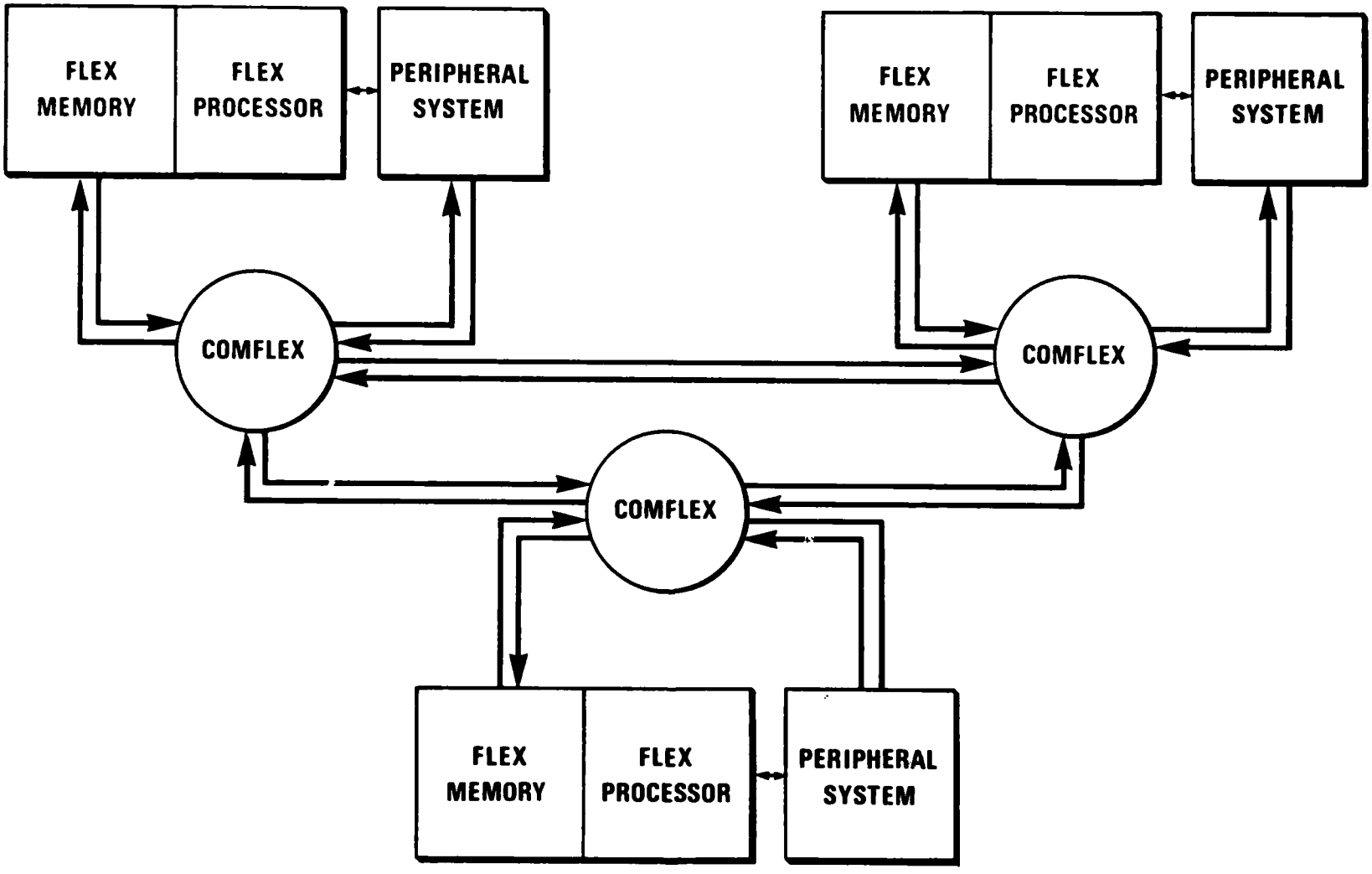


FIGURE 2·2 FLEX CONFIGURATION FOR RSRE

3) The FLEX Processor and Memory

3.1) The FLEX Processor

The main processor of the FLEX computer system is a soft-microprogammable processor based on bipolar bit-slice elements. The processor is constructed as a set of 8 bit slices and the processor word width can be 16, 24 or 32 bits. The current version of the processor is 24 bits wide. To make a processor of different width would require some changes to the micro - code.

The processor configuration is shown in figure 3.1. There are two main highways which connect the processor to memory and other subsystems. The 60 bit highway transfers data and tags to and from the memory, and transfers addresses to the memory. Packets can be transferred to and from the COMFLEX data channel, under microprogram control, via the byte multiplex highway. The COMFLEX data channel contains full packet buffering for incoming packets. Data can also be transferred to and from the controlling processor in the peripheral control subsystem by means of the byte multiplex highway and the SBC data channel. A further interface to the controlling processor is provided which is used for microcode loading, diagnostic monitoring of processor registers and highways, error reporting, and processor control functions (such as run, stop and "single stepping" the microprogram). The diagnostics and microprogram control facility permits comprehensive hardware fault location.

The microprogram control word is designed to allow many operations to be performed in parallel and has no overlapping use of control fields within the word, giving maximum flexibility. The writeable control store size is 4096 words of 80 bits per word. The microinstruction cycle times vary between 200 nanoseconds for internal processor operations to 600 nanoseconds for operations using the main highway.

A high speed fixed point multiplier allows single or double precision multiply, with a typical speed of 3 microseconds to form a signed 96 bit product. Hardware assisted divide is also provided. Microprogrammed floating point operations are provided giving a typical execution time for 48 bit floating point multiply of 8 microseconds.

The processor is constructed on double Europa circuit cards with the current 24 bit version using 26 cards of 15 different types.

3.2) The FLEX Memory

The FLEX memory is a high density, solid state dynamic memory which allows byte or word access via the 60 bit highway. The design is modular, having a number of semi-independent byte memories controlled by a microprogrammed interface processor. The interface processor has a cycle time

of about 100nS. Figure 3.2 shows the memory organisation.

The control algorithms allow 1, 2, 3 or 4 bytes to be read or written independent of any word address boundaries. This type of memory is often described as "spiralised" and it allows efficient use of storage, as items less than one word long do not use a complete word of storage. The memory interface processor performs access sequence checks during each memory cycle to ensure that illegal accesses (i.e reading more bytes than had been requested) are prohibited. This traps some microprogram and hardware faults. The design of the memory interface allows a high degree of overlap between the FLEX processor and the memory, giving high run time efficiency. One or two microcycles can be executed in parallel with memory access.

The current configuration uses four 10 bit wide memories, with a total data width of 32 bits (four memories are used for convenience of addressing). Each 10 bit memory word contains an eight bit data byte, a parity bit and a bit known as the "tag bit". These tag bits are used to indicate those locations containing references. The tag bit can only be read or changed by the microprogram.

Memory access codes are provided which allow direct memory to memory transfers using separate registers for the read and write addresses; auto increment and auto decrement of addresses; indirect addressing (via a third address register); and a number of special purpose read and write operations for manipulation of the tag bits.

The read and write cycle times are approximately 550 nanoseconds (including data transfers) and the unit size of the memory is a single crate of 28 double Europa cards currently giving 384 kilowords x 40 bits per crate. A maximum of 4 crates can be driven by the FLEX processor.

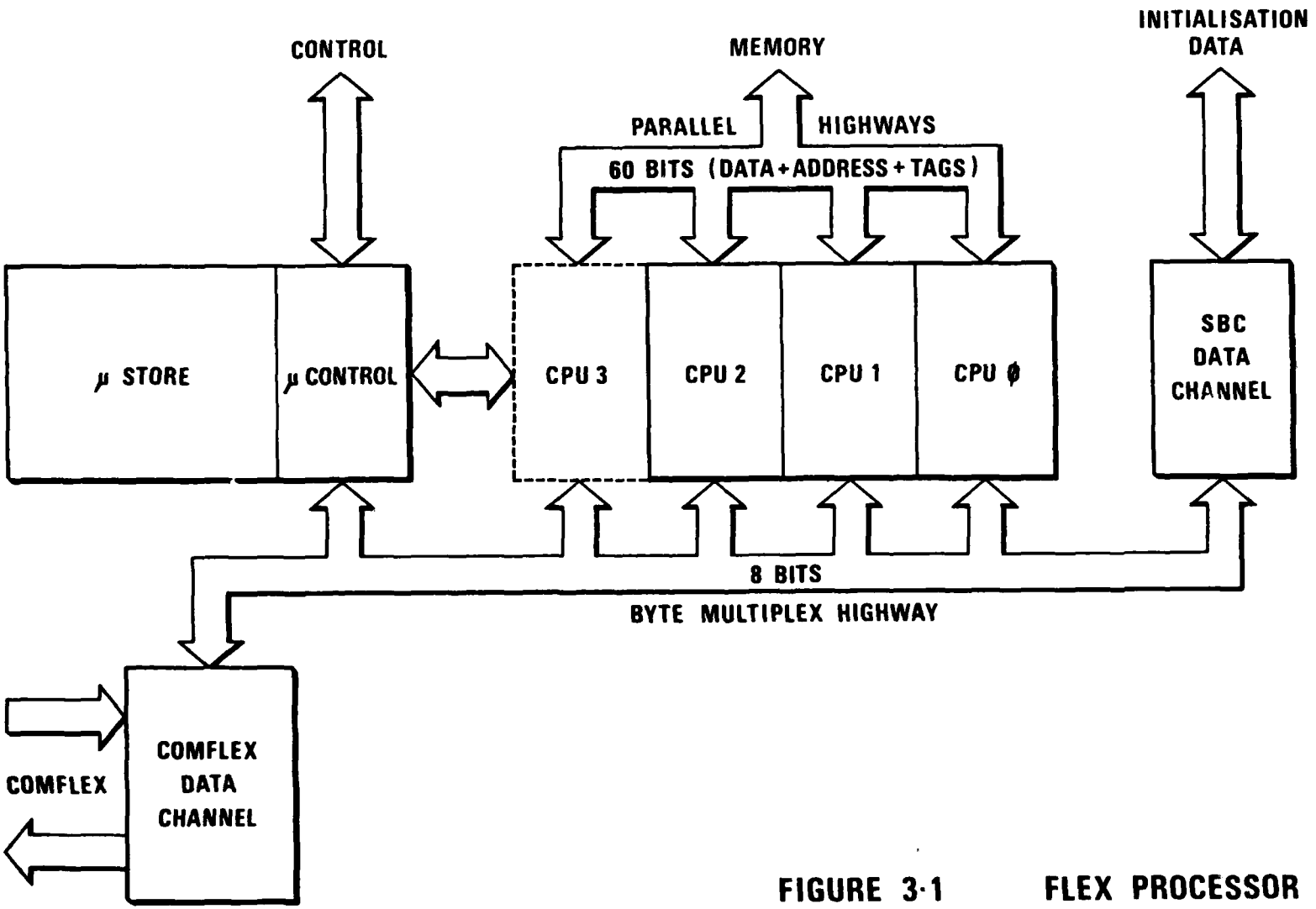


FIGURE 3·1 FLEX PROCESSOR

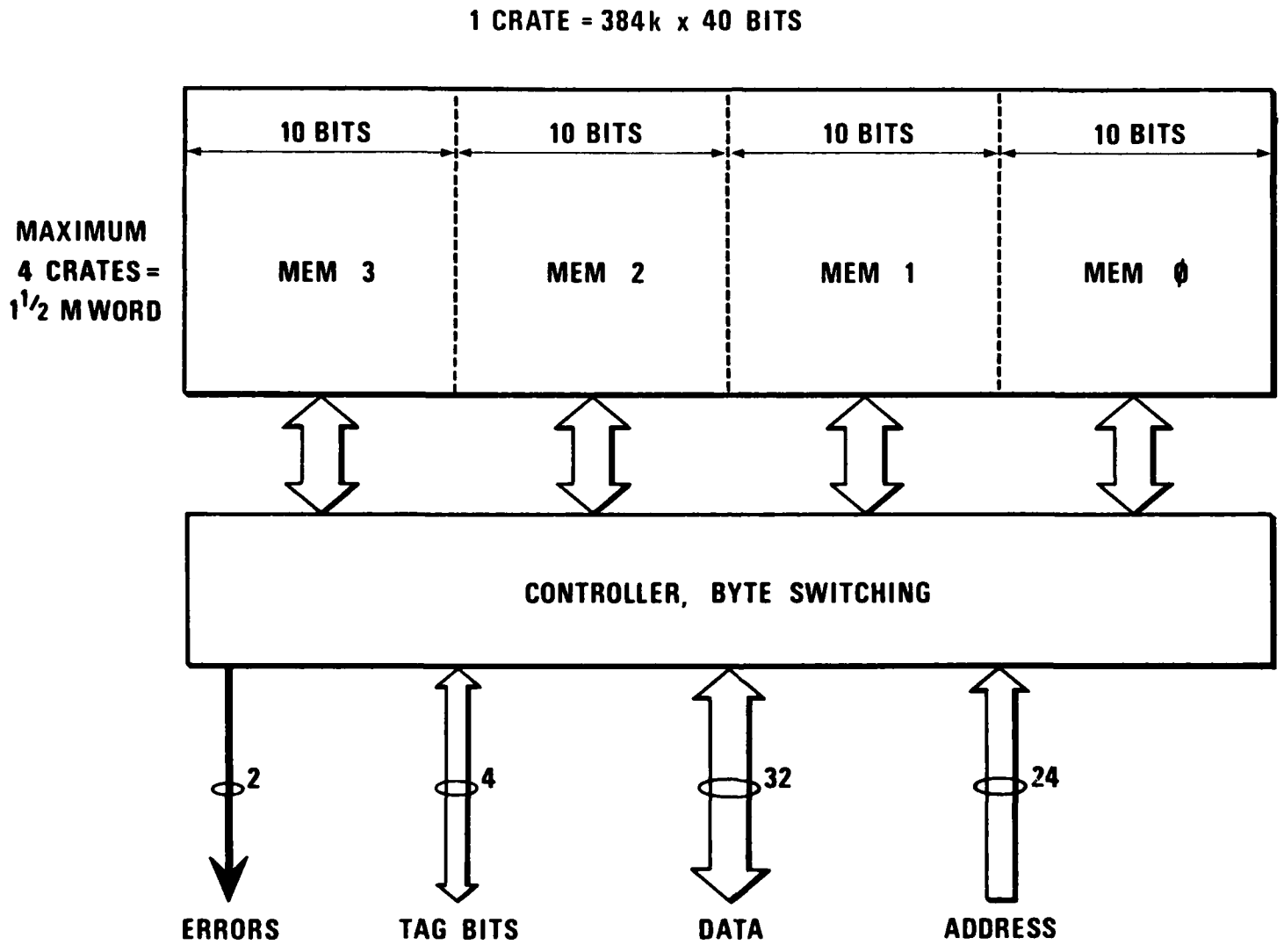


FIGURE 3·2 FLEX MEMORY

4) The Peripheral Subsystem

4.1) Function

The Peripheral Subsystem is designed to control a range of peripherals, to interface them to COMFLEX, and to perform system initialisation. The subsystem is required to control five VDUs, one or more disc drives interfaced to a CALCOMP 1150 formatter/controller, a paper tape reader and a line printer. The subsystem also has one bidirectional BSI and three CCITT V24 communication interfaces.

4.2) Hardware

The Peripheral Subsystem is based on the INTEL 8080 SBC (Single Board Computer) Multibus system, but includes some purpose built circuit cards, as shown in figure 4.1. Three of the peripheral channels: the disc interface unit, COMFLEX input and COMFLEX output, have high speed DMA (direct memory access) to the Packet Store. All other peripherals are under direct processor control. The four processors perform the following functions:-

Processor 1 provides control (addresses, status etc.) for DMA access between disc and Packet Store, and also provides control for the FLEX processor.

Processor 2 provides control for DMA access to and from COMFLEX and the COMFLEX control functions.

Processor 3 controls the four user VDUS.

Processor 4 controls all other slow peripherals: the tape reader, the line printer, the command VDU, the BSI communications channel and the V24 interfaces (which will include an interface to the RSRE site network.

The three DMA channels and the four processors can instigate Multibus transfers and their access to the Multibus is controlled by a priority mechanism. The processors communicate by means of the RAM on the RAM/IO card, and through parallel I/O channels.

4.3) Implementation

The implementation comprises eighteen SBC cards in one 19 inch rack mounting chassis. The formatter/controller is a separate unit which will be housed in one of the disc drives. Seven of the cards are standard Intel cards, but the remainder have been designed specifically for this application. Care has been taken with the design of the custom built cards to ensure that they can be used in any combination, so that the subsystem is truly modular, and can be configured to control any subset of its full complement of peripherals or any other peripherals interfaced to Intel Multibus, such as an X-25 interface.

Note: "Multibus" is a trademark of the Intel corporation.

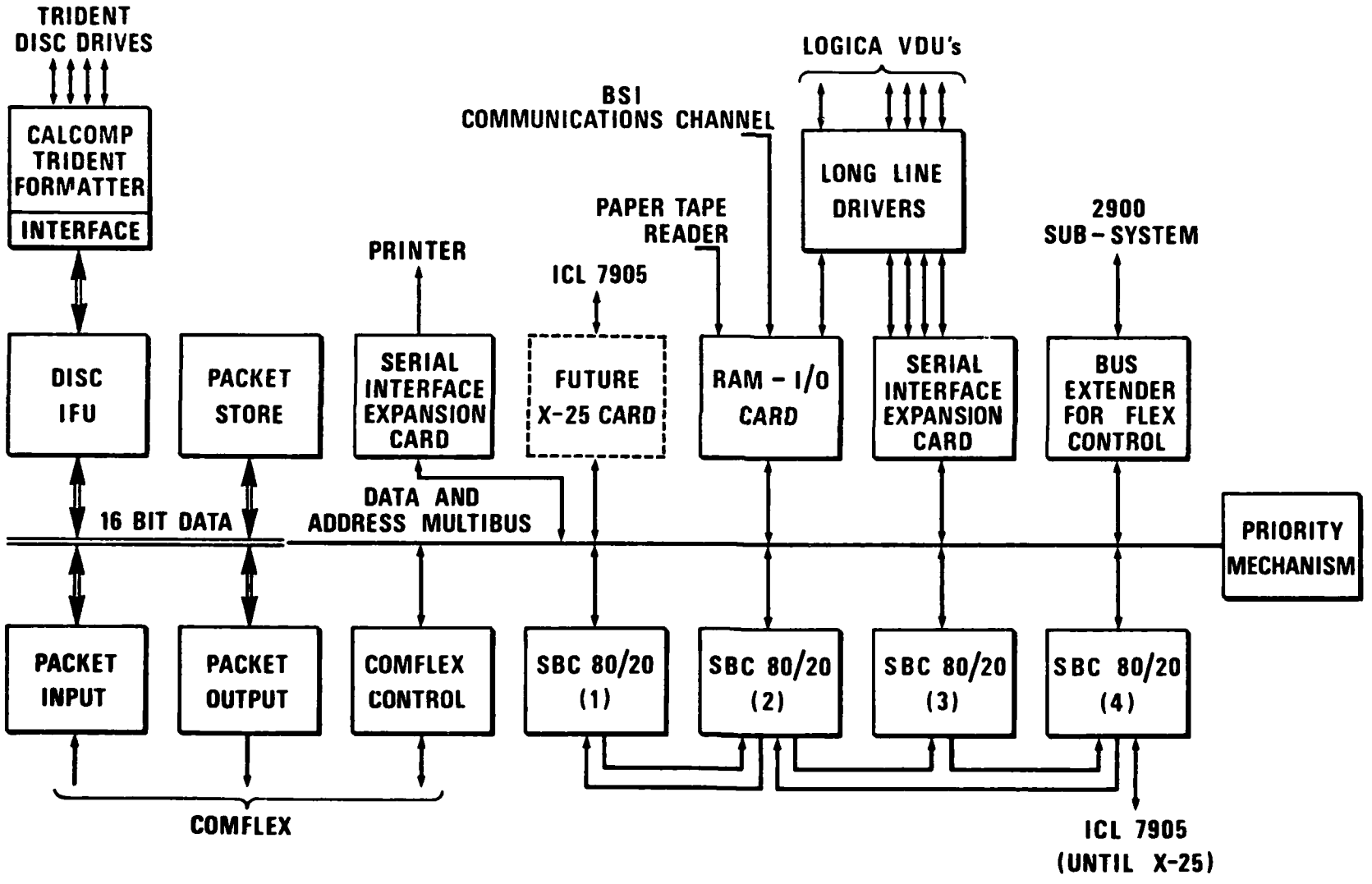


FIGURE 4·1 FLEX PERIPHERAL SUB-SYSTEM

5) The COMFLEX Communications Subsystem

5.1) Requirement

The communications subsystem is required to provide fast and reliable communications between the other FLEX subsystems. It should be capable of transmitting data at disc speed and should be resilient to transmission line and subsystem failures. This requirement is met by using a localised packet switching network.

5.2) Function

COMFLEX is a packet switching node which is capable of handling up to eight input lines and eight output lines at line speeds up to 2.5Mbytes/s. The COMFLEX packet size is variable from 15 to 270 bytes. There is a fixed packet header of 14 bytes, a data field which is variable from 0 to 255 bytes and a trailer byte. COMFLEX uses a parity bit on each byte to check the accuracy of the data transmission and will retransmit packets which have been transmitted incorrectly. The trailer byte contains a longitudinal sum check which is used for end to end error checking.

Data transmission and flow control are performed entirely by hardware. The routing function is performed partly by hardware and partly by software running in the controlling processor. COMFLEX contains logic to detect transmission line failures and to report these to the controlling processor. The routing operation is also checked and faults are reported to the controlling processor. The design philosophy of COMFLEX and its control algorithms are described in more detail in [5].

5.3) Hardware Implementation

The structure of COMFLEX is shown in figure 5.1. The input ports contain full packet buffering and the output ports are simple slave devices containing single byte buffering. The controller and the ports communicate via a data and control bus. The bus is multiplexed by a 20MHz, 8 phase clock, one phase being allocated to each input port.

The input ports control the data flow through COMFLEX. They arrange for packet retransmission in the event of data being transmitted incorrectly, monitor the state of the transmission lines and check the routing operation. The controller implements the routing algorithm by means of a look-up table. The table is loaded from the controlling processor (in the peripheral system). The controller also implements the flow control algorithm.

The controller, input ports and output ports are all implemented in hard-wired logic. The input port is based on a microprogrammed sequencer. The hardware implementation is

described in more detail in [6].

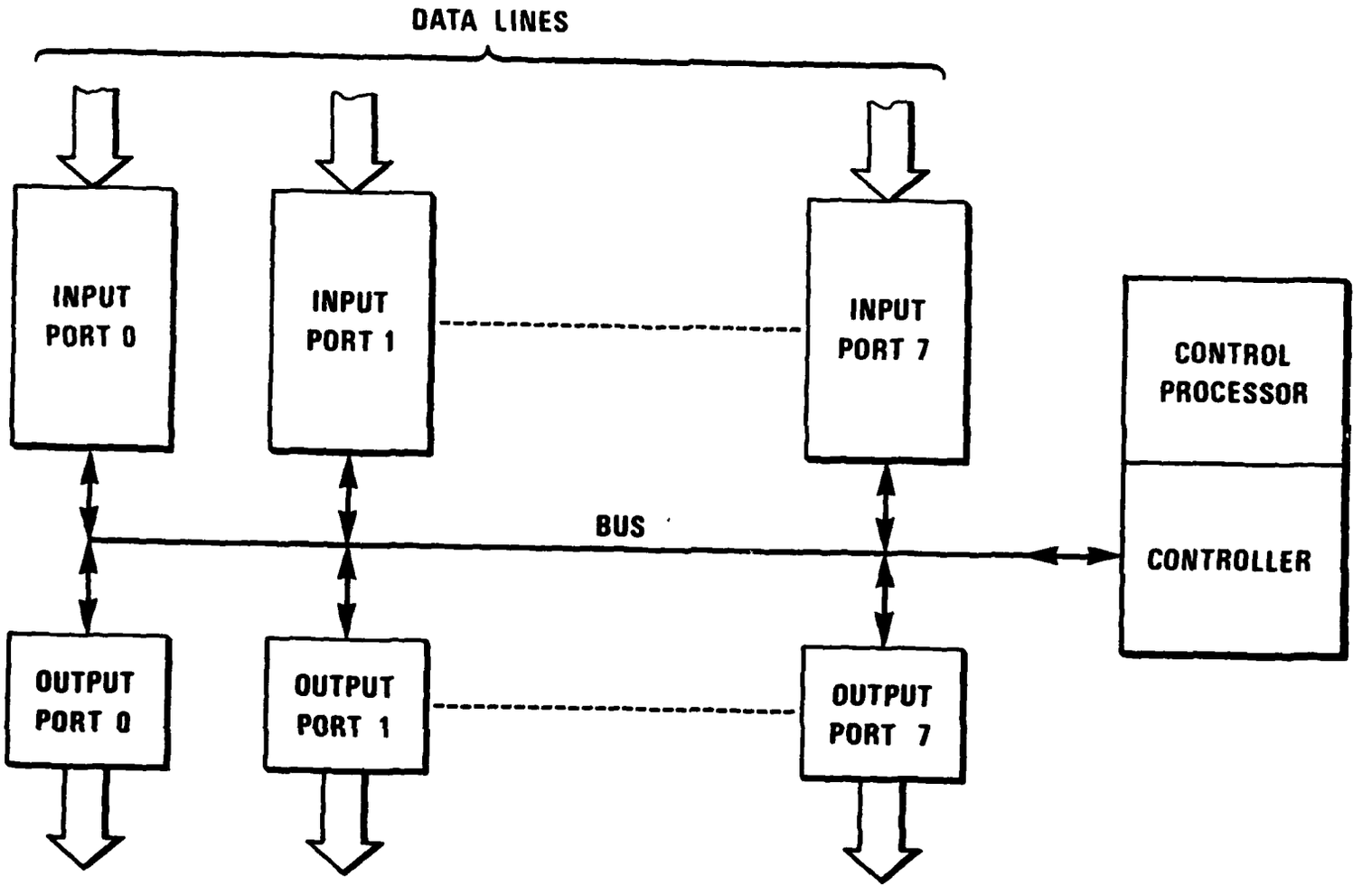


FIGURE 5·1 COMFLEX STRUCTURE

6) Processor Architecture

6.1) Introduction

The architecture of the FLEX processor is intended to match the essential requirements of most high level languages [7]. The provision of such an architecture simplifies the compiler writers task, enables user programs to be executed more efficiently, and also provides the (protection) facilities necessary to ensure that a secure environment for running user programs can be easily and elegantly produced.

One primary aim of an operating system is to guard against unauthorised use of system resources such as memory, file store and peripherals. The operating system also aims to protect the users from each other or to allow controlled co-operation as appropriate. The protection structures of conventional machines (e.g. ICL 1900, IBM 360) are unsuitable for this, and consequently the operating system has to provide a "firewall" behind which the programs which provide the desired protection reside. As a further consequence of the unsuitable hardware the operating systems tend to become very large (e.g. GEORGE 3, OS 360) and to contain programs which are not (logically) part of the Operating System at all (e.g. the text editor in GEORGE). We note however that most of the requisite protection is implicit in good programming languages, viz:

- procedures provide functions with well - defined interfaces whose contents are inaccessible;
- identifier names of objects are non - computable;
- one cannot jump to an arbitrary place in the program (or out of it).

The FLEX instruction set enforces this protection, hence the "firewall" for main store allocation has been reduced to the properties of the microprogram. Controlled and secure access to backing store and peripherals is achieved through a procedural interface to a small kernel written in RS Algol 68.

The remainder of this section gives an overview of the FLEX processor architecture at the macro - code level.

6.2) Storage Management

Store is allocated in blocks by the micro - code, and every piece of program or data is held in some block. Blocks start on word boundaries and contain an overhead word which specifies the block type and its size in words. The maximum block size is 1Mword. There are several block types, and these fall into two groups - those associated with procedures and those containing data. The modes of access which are allowable on any block are defined by its type.

The data blocks can be organised in words, bytes or bits. They can be indexed, and both writing and reading is permitted.

For each procedure there is a closure block and a workspace block. The procedure is accessed through a closure block on which the only valid operation is procedure call. The closure contains pointers to the code, constants and non - locals for the procedure. The workspace contains the locals and the stack for the procedure and may be read or written, and (conceptually) a new workspace is created at each procedure call. Clearly this arrangement allows code to be used re-entrantly. Less obviously, this store architecture allows procedures to be written which deliver procedures as results, without the restrictions necessary on conventional stack machines to overcome the problems associated with accessing non - locals. This attribute aids in the provision of a secure procedural interface to the kernel.

Store can be addressed by pointers and references. Pointers are created by the micro - code and are identified by means of the tag bit in the memory word (which only the micro - code can access). Pointers are thus unforgeable and are immutable once created, hence providing the protection described in 6.1 above. A reference comprises a pointer plus an offset and provides indexed addressing into a block. The store protection mechanisms check that the pointer is to a block header, that the offset is within the block and that the type of access is valid for that block type.

The tagged store architecture has enabled the provision of a garbage collector in micro - code for which the execution time increases only linearly with store size. On conventional machines garbage collection time increases more than linearly with store size.

Disc pointers (and references) are identified by the tag bit like their main store counterparts.Disc pointers are marked to distinguish them from main store pointers. They may be stored on disc and are created by the filing system (in the peripheral subsystem) rather than by FLEX micro - code.

6.3) Registers

There are effectively only four registers in the machine which are relevant at the macro - code level. These are:

U, the universal register. U is capable of holding one object of N words, bytes or booleans, where N is an arbitrary integer (max. size 1Mword).
WS is the current workspace pointer.
ST is the top of stack in the current workspace.
PC is the program counter.

Most operations in the machine operate on U or on U and the object on top of the stack. U is adjusted by the micro - code to the size appropriate to object which it is to contain.

6.4) Control

Clearly FLEX has to provide instructions to allow change

in the flow of control. Simple conditional jumps are provided which depend on the state of a boolean in U. In addition to conventional jumps FLEX has two instructions which implement a simple case (multi - way jump) and an associative case switch respectively.

Procedures are the most important type of object in FLEX and their execution warrants more detailed discussion. A procedure is called by placing the closure on top of stack, the parameters (if any) in U and executing the call instruction. The current values of PC and ST are stored in the current workspace, a new workspace is created and a link to the old workspace is put in the new one. The values of ST and PC are set to zero and the new workspace is made current.

On procedure exit, control is returned to the calling workspace in the obvious manner. The current workspace becomes idle, but is still associated with the code block that has just been exited. If the garbage collector is run before the procedure is next called then the space will be recovered, otherwise the workspace will be re - used. This reduces the rate at which garbage will be generated and thus reduces the frequency of garbage collection, and increases the effective speed of execution of the call instruction.

The mechanism for handling program execution failures stems from the procedural structure. When a procedure is called a place in the calling procedure may be nominated as the return point in the event of a failure in the called procedure. When a failure does occur control will pass to the nearest procedure in the call chain for which there is such a failure return point. This mechanism enables a hierarchic failure handling (often called exception handling) structure to be implemented, if desired.

6.5) Instructions

The instructions are of 1, 2 or 3 bytes in length, and thus good use may be made of the memory's byte addressing capability to acnieve compact code. There are essentially four types of instruction:

Instructions which load or store U or the top of stack.
Control instructions, e.g. CALL.
Instructions which operate on U and the top of stack, e.g. PLUS.
Special instructions.

The load and store instructions conceptually move N words, bytes or bools, where N is an arbitrary integer. Data is only moved if it is strictly necessary, and often the effect of the load is achieved simply by altering the location of U. These instructions take one parameter which is the offset in the source or destination block. The source of data for the load instructions is the locals, the non - locals, the constants or the block specified by a pointer in U. The destination for the

store instruction excludes the constants.

The control instructions have been described in section 6.4.

The arithmetic, logical and certain data manipulation instructions work on U and the top of stack. The obvious arithmetic operations of PLUS, MINUS, TIMES and DIVIDE are provided on reals and integers. Logical operations such as AND and OR on bools are provided. There are a number of comparison operators, such as EQUALS and LESS THAN, which leave a boolean in U. There are certain instructions which are not found in conventional machines which help provide compact code and speedy program execution. An example of such an instruction is INDEX which forms an index into an N dimensional array. The instruction takes an N - tuple of integers (the indices) on the top of stack, and a 3N - tuple in U in the form

l ,u ,s ..... l ,u ,s

where l is the lower bound, u the upper bound and s the stride of the dimension of the array given by the index. The instruction leaves the offset into the array in U or produces a failure if one of the indices is out of bounds.

There are a number of special instructions provided for a variety of purposes. To facilitate parallel processing two instructions: SECURE SEMA and RELEASE SEMA are provided. These enable separate processes to make controlled use of shared resources, and have similar semantics to Dijkstras P and V operations (these instructions have not been micro - coded and are currently interpreted by the kernel). Some instructions are provided to perform the mode changes required by the code generator in a compiler. Examples are MAKE CODE BLOCK which turns a data block into a code block, and CLOSE which similarly produces a closure block.

7) Software

7.1) Operating System

It is not intended that a complicated operating system shall be produced as part of the FLEX development, but a safe operating environment will be provided in which users can run their programs. The design of this operating environment is such that the user can indulge in activities, such as the production of his own file structuring and naming scheme, which are normally the preserve of the systems programmer. He can do this without any particular priviledge or status, and without being aware that he is doing what is normally "systems programming".

The safe operating environment is provided by the protection implicit in the FLEX instruction set, and by the procedural interface presented by the Kernel.

7.2) Kernel Facilities

Rather than being a program, as such, the Kernel is a set of basic procedures designed to allow the FLEX machine to be run in multi - processing, multi - access mode. These procedures implement such functions as interrupt, peripheral and process handling. Certain of these functions are made directly accessible to the user. It is not appropriate to describe the Kernel functions in detail here, but some of the more salient ones (to the user) are described.

The user interface to a VDU is effectively via two procedures: READVDU and WRITEVDU. WRITEVDU takes two parameters, an array of characters which is the message to be sent and a screen address which specifies where the message is to be sent. READVDU takes two parameters, an array of characters which is a prompt to be sent to the VDU, and a screen address which specifies where the prompt is to be sent. The read interface to the VDU includes some editing functions. Pressing the send key after inputting some text, or pressing one of the editing keys will cause an action by the reading procedure. If the normal "send" key is pressed the procedure delivers an array of characters which is the message typed on the VDU after the prompt. If an editing key is pressed then the appropriate action, such as inserting or deleting a line, is carried out. Any instance of these two procedures is specific to one VDU.

Access to the file store is achieved by means of two procedures WRITEDISC and READDISC. In order to help preserve the data integrity, the file store is organised so that data can not be overwritten. Thus when writing to the file store the procedure WRITEDISC is called with the data to be written as a parameter, and it delivers a backing store pointer which specifies where the data has been stored. The data will be stored as one logical block and the pointer may subsequently be used to retrieve the block but not to overwrite it. To read

the data from the file store the procedure READDISC is called with the appropriate disc pointer as a parameter and it returns the block previously written, starting at that address.

The no over - writing property of the file store implies that the data on the file store forms a tree structure and it can be accessed from a set of root pointers. Clearly, these root pointers must themselves be over - writeable; this in itself could lead to inconsistencies particularly where several FLEX machines are accessing the same backing store. This problem is overcome by making the modification of a root pointer an indivisible operation in the peripheral system, requiring the previous root pointer as key. Thus the onus for maintaining the consistency of the data pointed to by the root pointer is with the procedure which tried to alter it - outside the "firewall". The user thus has responsibility for his own data integrity, but is provided with the mechanism for ensuring this integrity.

Because of the no over - writing property, unless space were reclaimed, the file store would eventually become full. However, space which is no longer pointed to can be reclaimed, and a disc garbage collector is provided for this purpose. The no over - writing property prevents the use of randomly addressable files so applications, such as databases, which conventionally use this type of file have to be implemented using a different type of file structure.

In order that a user may carry out parallel processing within his own programs, a facility for creating processes has to be provided. This is done by means of a procedure PROCESSMAKER, which takes a procedure parameter, turns that parameter into a process and delivers a procedure to run this new process.

7.3) User Procedures

A "program" may be regarded as a special case of a procedure, thus the concept "program" is unnecessary. FLEX recognises procedures but does not recognise "programs" and what are conventionally known as "user programs" are referred to as "user procedures". An RS Algol68 compiler exists and can be used for the production of user procedures. A user procedure has access to the Kernel procedures as non - locals and this non - local block is known as the environment interface. The Kernel procedures can be bound to the user program at the time that it is initiated, and the READVDU and WRITEVDU procedures in this environment interface are specific to the terminal from which the program was initiated.

Diagnostics will be provided by means of a program which allows a programmer to follow the call chain of his program, examining the data as he goes. The information will be presented so that the correlation with the program source text is apparent.

7.4) Facilities on RSRE Service Machine

Although the aim of the FLEX development is not to produce a complicated operating system, it will be necessary to provide a simple, standard, operating system for the casual users of the machine. The form that this will take has not yet been defined but will include some form of file structuring and naming facility, and will probably include a higher level interface to the input - output system.

Facilities such as a text editor will be available, but they will be implemented as independent procedures rather than as part of the operating system (although the majority of the "editing" work is performed by the READVDU procedure in the kernel).

8) The State of the FLEX Development

Laboratory models of all the three hardware subsystems have been designed and commissioned. The FLEX micro - code, the Kernel and the peripheral control software in the peripheral subsystem have been implemented and are operational. The RS Algol68 compilation system for FLEX is complete and is resident on FLEX, so it is now possible to run RS Algol68 programs on the laboratory model FLEX machine. A simple diagnostic program and a text editor have been produced thus the FLEX laboratory model is now a viable, stand alone, single computer system.

To produce a multi - computer system further development is required to bring the hardware to a state where it is reliably reproducible, and so a second version of FLEX, known as the prototype, is being developed. Logica Data Systems Ltd. will manufacture copies of the prototype under licence. A prototype of COMFLEX has been produced and three copies of this prototype have been manufactured by Logica Data Systems Ltd. A peripheral subsystem prototype has been designed by Logica Ltd. and is now being commissioned. The FLEX processor prototype has been designed at RSRE and commissioning is due to be completed in early 1980. It should be noted that this document describes the design of the prototype, not the laboratory model.

As the designs of the prototype and the laboratory model have minor differences some of the low level software (e.g. the micro - code) will have to be re - written, but the functions implemented by the software will be the same for the two versions.

It is expected that the prototype FLEX computer will be operational in mid 1980, and that a network of three computers will be available as a research machine about a year later.

9) Acknowledgements

The FLEX computer has been developed as a co - operative effort between C1 and C2 divisions at RSRE, with assistance from Logica. The team currently includes Dr. J M Foster, I F Currie, P W Edwards and J D Morison from C2 division, C I Moir, J A McDermid, Dr. C H Pygott and P J Bradford from C1 division and Dr. J Harrison, D Fishwick and R Cummings of Logica. Various other people have given assistance in the past. Special thanks are due to Dr. A J Fox for his support and encouragement throughout the project.

This document draws on internal RSRE papers produced by various members of the project team, and has been compiled and edited by J A McDermid.

10) References

[1] "Introduction to the RS portable compiler", RRE Technical Note 802, S G Bond, P M Woodward, August 1977.

[2] Algol 68R Users Guide, P M Woodward, S G Bond, HMSO, 1974.

[3] "Abstract data and functors", J M Foster, P D Foster, Sigplan notices, pp161 - 167, 1977.

[4] "The FLEX Computer: hardware aspects", C I Moir, RSRE Technical Note (to appear).

[5] "COMFLEX" - A high speed packet switch for inter - computer communication", J A McDermid, Proceedings of Eurocomp 78, pp187 - 204, May 1978.

[6] "COMFLEX Operators Manual", Dr. J Harrison, Logica Ltd. Report No. 2559/003A. Oct. 1979.

[7] "The FLEX computer: software aspects", J M Foster, RSRE Technical Note (to appear).