intellitech

The Intelligent Use of Technology

(2)

N.mPc AND ITS UTILITY FOR SPACECRAFT APPLICATIONS

INT-83-47/1

(2)

N.mPc AND ITS UTILITY FOR SPACECRAFT APPLICATIONS

Computer Aided Engineering Tools for

Spacecraft Multiprocessor Systems

(Contract #OER82 - 05067)

AUTHORS: Dr. C. Laferriere (1)
Mr. A. Lam

APPROVED BY: Dr. S.A. Mahmoud

DATE: 17, January, 1983

INTELLITECH CANADA LIMITED
352 MacLaren Street,
Ottawa, Ontario
K2P 0M6

Government of Canada / Gouvernement du Canada

Department of Communications

DOC CONTRACTOR REPORT DOC-CR-SP -83-059

DEPARTMENT OF COMMUNICATIONS - OTTAWA - CANADA

SPACE PROGRAM

TITLE: N.mPc AND ITS UTILITY
FOR SPACECRAFT APPLICATIONS

AUTHOR(S): Dr. C. Laferriere
Mr. A. Lam

ISSUED BY CONTRACTOR AS REPORT NO: INT-83-47/1

PREPARED BY: Intellitech Canada Ltd.
352 MacLaren St.
Ottawa, Ontario
K2P 0M6

DEPARTMENT OF SUPPLY AND SERVICES CONTRACT NO: 36001-2-0560
SER NO. OER82-05067

DOC SCIENTIFIC AUTHORITY: R. A. Millar

CLASSIFICATION: Unclassified

DATE: January 17, 1983

TABLE OF CONTENTS

LIST OF FIGURES

1.0 INTRODUCTION

This report constitutes deliverable 4.2 of work done (under contract #OER 82-05067) for the Federal Department of Communications, Communications Research Centre, Ottawa, entitled "Computer Aided Engineering (CAE) Tools for Spacecraft Multiprocessor Systems".

The report describes:

1. The computer aided engineering tool N.mPc purchased under this contract.

2. The utility of the purchased CAE tool in the design of multiprocessor systems for spacecraft applications.

As such, the report describes the work done under tasks 3.2 and 3.3 of the contract. The installation of N.mPc was described in an earlier report, [LAFE82b]. Detailed program listings of the simulations implemented using N.mPc are contained in [LAFE83].

The report is divided into four main sections which are described below:

1. Section 1, is a general introduction which includes background material, a brief discussion of multiprocessor architectures for spacecraft applications, a description of N.mPc and an overview of the work accomplished in fulfillment of tasks 3.2 and 3.3.

2. Section 2 is a description of the technical work done with N.mPc. It describes the initial installation and testing of N.mPc followed by the simulations implemented. This work includes some preliminary simulations done to gain familiarity with the N.mPc tool followed by more advanced multiprocessor work. Also documented is the complete simulation of the Ferranti F100-L micro processor and its support chips.

3. Section 3 contains comments on the utility of N.mPc for spacecraft applications. Attention is paid to each of the components of N.mPc, and comments on the utility and the limitations of the current implementation of N.mPc are provided in the last two sub-sections.

4. Section 4 describes suggested directions for further work with N.mPc within an integrated computer aided engineering design environment involving complementary tools.

1.1 BACKGROUND AND REVIEW OF DESIGN METHODOLOGY

The work reported here is in fact part of a greater effort whose primary goal is to develop an integrated set of computer aided tools to assist and automate the design of multiprocessor systems and which is aimed primarily at spacecraft applications. A general design methodology and a set of computer aided tools to support this methodology are described in earlier reports [MAHM82a], [LAFE82a]. As illustrated in Figure 1.1, this methodology can be divided into two general stages:

a) The high level design stage, which is concerned with high level requirements, correctness of the design and functional decomposition.

b) The low-level design stage, which constitutes the work remaining after software/hardware boundaries have been defined. This low level work also includes implementation and testing.

The present contract involves the selection, purchase and application of a suitable computer aided design package to support the low level design stages and the definition of a set of suitable Ada* constructs which would be used in the high level design stages to facilitate the top down design and development of multiprocessor systems. The latter part of this contract work is the definition of the necessary mechanisms to interface high level design tools to the low level design environment. This report is concerned only with the former effort, namely the description and utility of the low level design tool N.mPc for simulating multiprocessor architectures.

---

* Ada is a trademark of the U.S. Department of Defence.

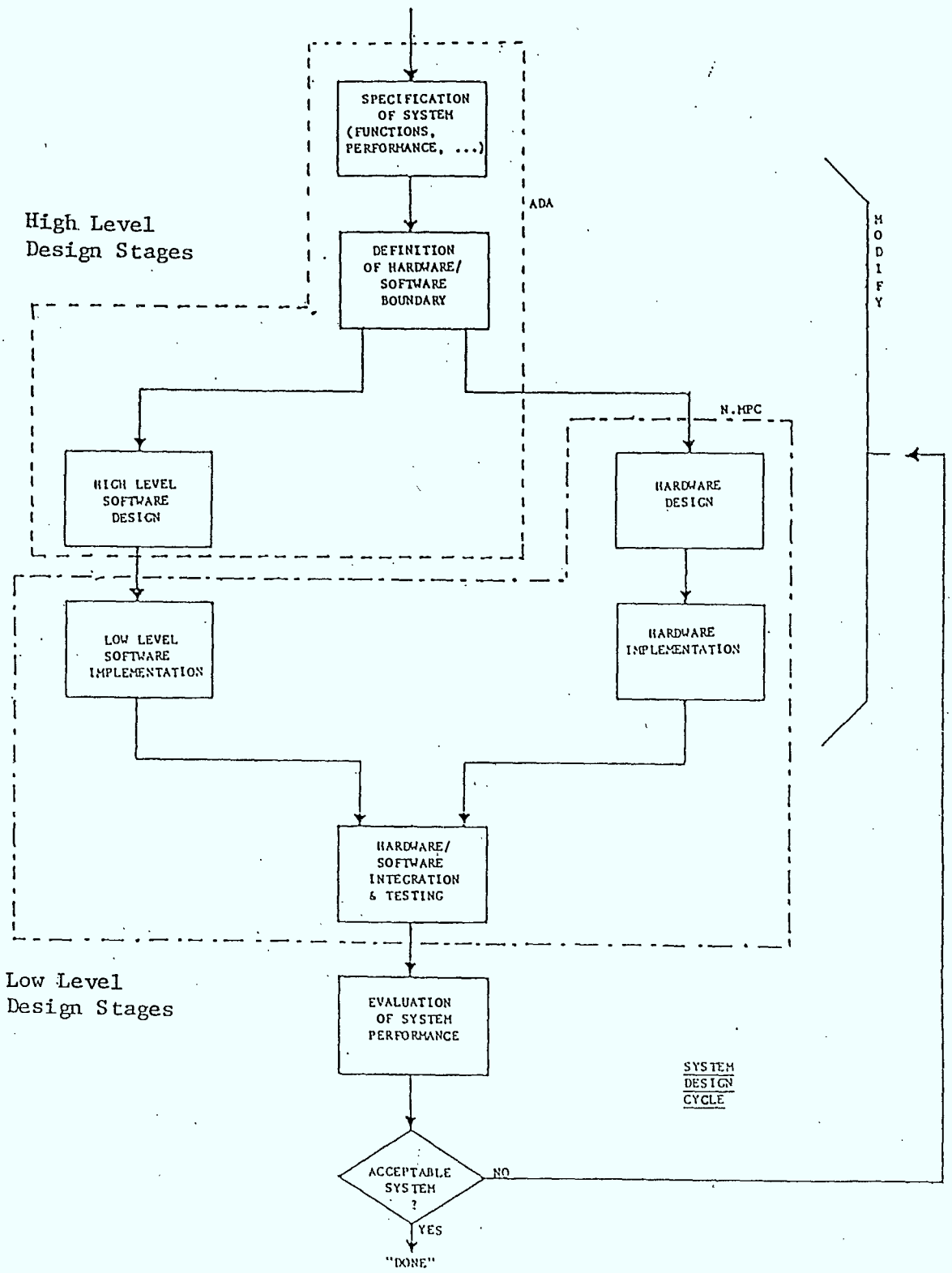

FIGURE 1.1 General Design Methodology

1.2 MULTIPROCESSOR ARCHITECTURES FOR SPACECRAFT APPLICATIONS

A survey of various multiprocessor configurations currently used in spacecraft applications is to be found in [OUIM82] and it may be worthwhile to review the rationale behind multiprocessor systems for spacecraft applications. For a few years already, the trend has been towards cheaper, more flexible and more powerful processors. As well, spacecraft designers are now more inclined to put advanced application programs on spacecraft "on-board" computers. Having computers and programs where they are needed relieves earth stations from the burden of having to support expensive systems. Furthermore, responsiveness of programs such as control monitors from on-board equipment is improved. When processing capabilities are to be put aboard a spacecraft, several difficulties have to be solved, two of which are of great importance:

1. The first point deals with the processing power required, combined with the final size and weight of the on board computer. This suggests the use of microprocessors which happen to be cheap and lightweight but are unfortunately somewhat slow(*). This relative lack of speed can be obviated by using multiple processors in a suitable arrangement.

2. The second difficulty is concerned with reliability. When a spacecraft is in orbit, reliable operation of all its parts is of paramount importance. Several techniques can be used to achieve this goal, such as the use of redundant components (e.g. standby processors), the use of a serial bus to minimize the number of lines, etc.

The European designed L-SAT exemplifies some of the above techniques. L-SAT supports two serial busses, and uses a multiple processor arrangement to increase reliability. Modelling such a

(*) Although faster and more powerful processors are always continuously being developed.

system had tentatively been set as a goal. Several factors such as lack of time, host processor limitations etc., have prevented a complete simulation of L-SAT. Efforts were directed towards multiprocessor simulation with processors being interconnected by a "Multibus"# structure. In this fashion, issues such as processor behaviour and especially processor co-ordination were investigated. In particular, two methods of processor co-ordination were simulated, namely busy waiting by polling a global structure and interprocessor interrupts.

The vehicle to carry out those tests was the N.mPc system developed at Case Western Reserve University, Cleveland, Ohio. N.mPc and the reasons for choosing it have already been described in earlier reports [LAFE82b] (deliverable 4.1) [MAHM82] (August Progress Report). The multiprocessor system simulations were built using the N.mPc system and run using N.mPc runtime environment.

1.3 DESCRIPTION OF N.mPc

N.mPc ([PARK79a], [PARK79b], [ROSE79], and [ORDY79b]) is an interactive environment for the design and evaluation of multiprocessor systems. It is composed of the following software modules; (shown in figure 1.2 and 1.3):

1. Metamicro assembler, which is a programmable assembler to be used in program development for target processors.
2. Linking Loader, which complements the programmable assembler and whose main purpose is to resolve addressing conflicts so as to produce a loadable object file.

(#) Multibus is a trademark of Intel Corporation.

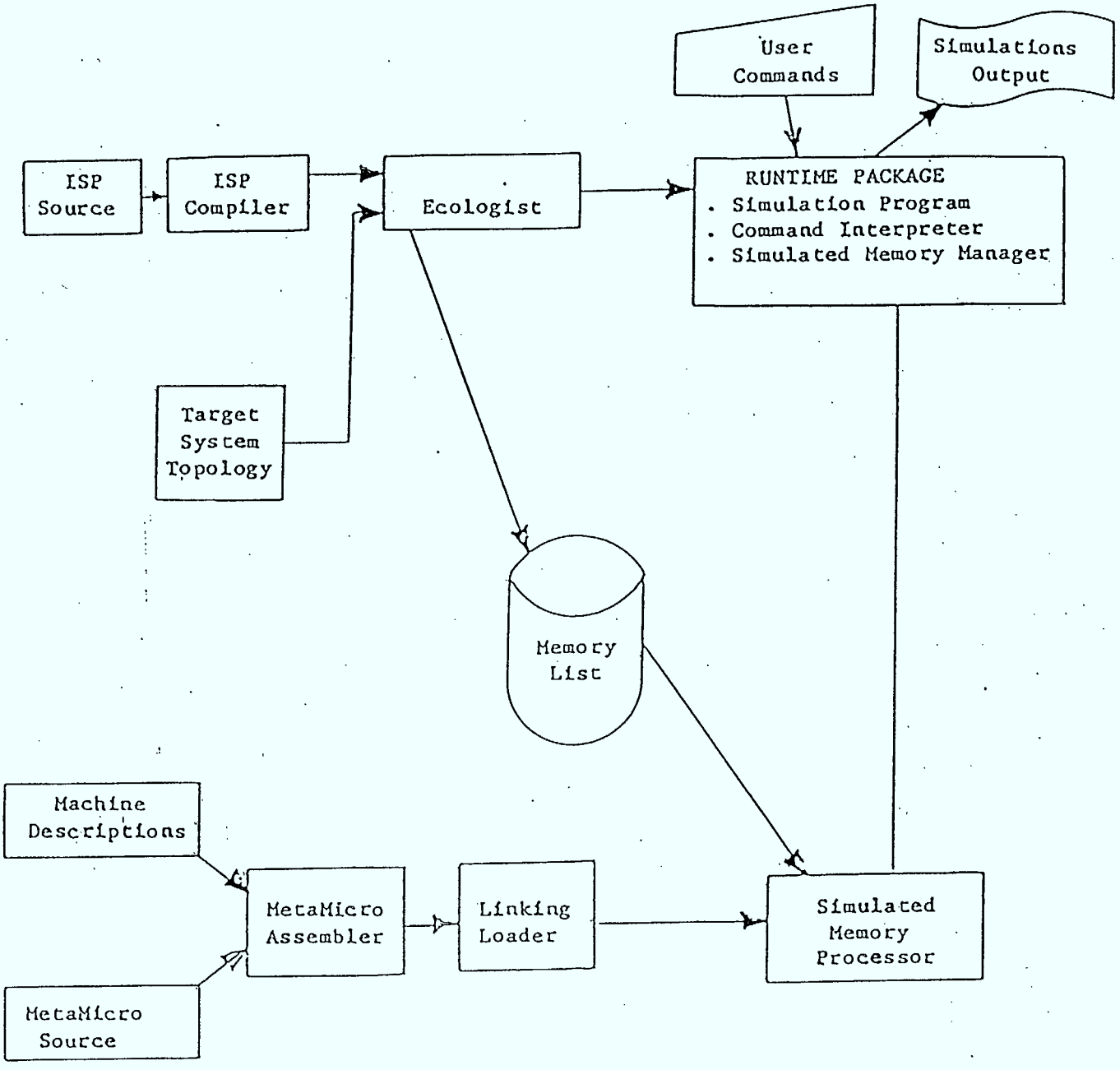

FIGURE 1.2 Main Components of the N.mPc System

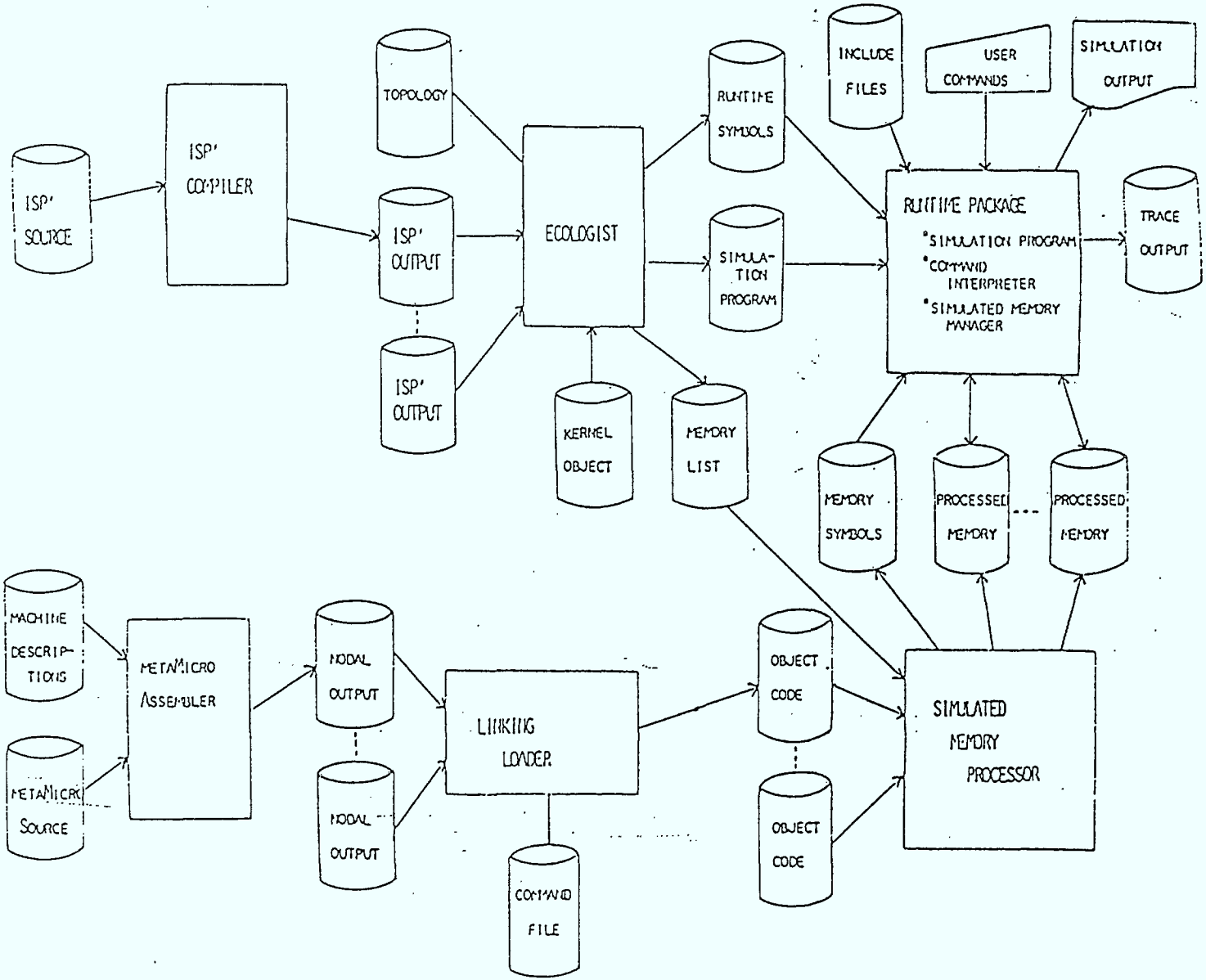

FIGURE 1.3 Detailed Block Diagram of N.mPc

3. ISP´ Compiler which is used to compile an ISP´ description of the hardware to be simulated. The out-put of the computer is executable PDP-11 code.

4. Ecologist which is used to build the simulation according to a topology file describing the various hardware modules (compiled ISP´ files) to be used in the simulations. The topology file also describes the memory modules and the interconnections of all modules in the simulation.

5. Simulated Memory Processor which prepares memory files (i.e. containing executable programs) for use by the runtime package.

6. Runtime package which consists of the simulation itself, the memory manager and the command interpreter. This last component (i.e. the command interpreter) is the interface with the user and the means by which the user can control the simulation.

Looking back at Figure 1.1, it is possible to link certain stages of the design methodology with the components of N.mPc. The software implementation stage corresponds, albeit partially, to the combination of the metamicro assembler and the linking loader. Hardware design and implementation are done by the ISP´ language description facility and its associated compiler. Hardware/Software integration, testing and evaluation tasks are performed by the ecologist, simulted memory manager and run-time environment.

In fact, at the integration and testing stage, N.mPc can be thought of as a software logic state analyzer and a software In Circuit Emulator. Based on those capabilities, N.mPc proved to be a suitable system to support the simulation activities outlined in the next section.

1.4 OVERVIEW OF THE SIMULATION WORK

Simulation work began with an attempt to gain familiarity with N.mPc. Consequently, very simple simulations were undertaken; such simulations typically involved a single processor (e.g. RCA 1802, Intel 8085) and a simple program.

Once a reasonable proficiency was achieved, multiprocessor simulations were designed around the Intel 8085 processor. At first, a simple interconnection structure was used, but eventually a better engineered simulation was produced in which each processor had its own private memory and processors were interconnected by a multibus structure.

The next goal was to develop a new processor description and to program the metamicro assembler/linking loader combination to accept the assembly language of this processor. The target processor was the Ferranti F100-L 16 bit bipolar microprocessor. Besides the F100-L, the associated support chip set was also defined using N.mPc.

Finally, the F100-L, was used as processor in a multibus based multiprocessor simulation. Processor co-ordination achieved through interprocessor interrupts.

2.0 DESCRIPTION OF TECHNICAL WORK

2.1 INSTALLATION OF N.mPc

N.mPc is distributed on a magnetic tape, recorded in the UNIX* "tar" format. The system is distributed in source form and installation consists of compiling and assembling various modules. The directory organization of Intellitech's N.mPc system is exactly that suggested in the N.mPc installation manual [ORDY80] for Release 2. This directory organization is shown in Figure 2.1.

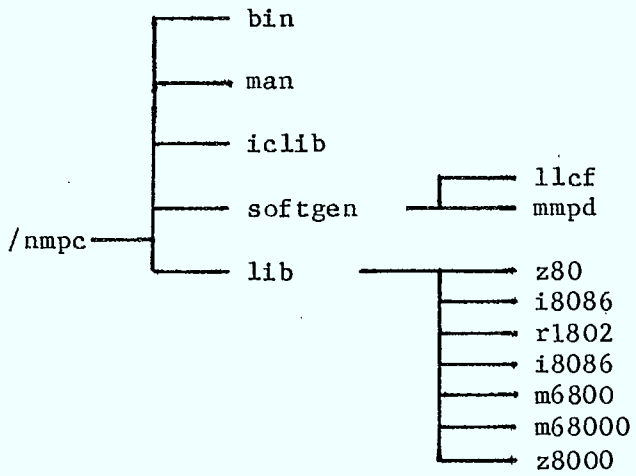

FIGURE 2.1: N.mPc Directory Organization

Each sub directory of nmpc contains a particular type of file and furthermore, the system expects this particular file classification. The contents of each subdirectory are as follows:

1. The "bin" directory contains executable N.mPc programs in the same fashion as root/bin contains executable UNIX programs.

(*) UNIX is a trademark of Western Electric Co.

2. The man directory contains an on-line manual for the N.mPc System, which is very similar to the UNIX on-line manual (also called "man").

3. As its name indicates, the "iclib" directory is a library of ISP´ hardware descriptions for various microprocessors. This directory can be automatically searched by the "ec" program if so desired.

4. The software counterparts to the ISP´ hardware descriptions are to be found in "softgen". In that directory, two subdirectories are present: "llcf" and "mmpd". Metamicro assembler description files for a given processor assembly language reside in "mmpd", whereas linking loader command files are in "llcf".

5. The "lib" directory is comprised of several subdirectories, one for each target processor type. Subdirectory "i8085", for example, contains the ISP´ description of the 8085 hardware, the meta micro descriptor of the 8085 assembly language and a linking loader command file for the 8085. The "lib" directory does not present new processors or features, but re-organizes information already contained in "iclib", and "softgen". The overhead thus introduced is small since UNIX uses file links.

N.mPc comes supplied with a library of processor descriptions which covers the most popular 8 and 16-bit devices. Processors such as the Intel 8080, 8085, 8086 , the Motorola 6800, 68000, the Zilog Z80, Z8000, are supported in terms of both hardware and software. Unfortunately, peripheral chips such as bus controllers, programmable interrupt controllers, DMA controllers, etc., are notably absent from the N.mPc libraries. It is therefore left to the users to develop those modules.

Another implementation detail is the speed of execution and the maximum size of a simulation. It becomes obvious fairly quickly that simulating a fast processor on a slow machine is not the most desirable situation, although the usefulness of the simulation is not diminished. Of greater importance is whether the host PDP-11 supports separate Instruction and Data spaces or not

(Intellitech's 11/23 does not). Not having separate I/D spaces reduces the size of the simulation but also reduces the maximum size of dynamically created tables used by N.mPc system programs. The ISP' compiler, for example, had to be processed by a special program called "23fix" which performs a partial mapping of the Data space. The compiler thus becomes able to process larger processor descriptions. It is worthwhile mentioning that any ISP' description more complicated than that of an 8085 cannot be compiled using the standard compiler. The "23fix" proved very valuable.

2.2 PRELIMINARY SIMULATION

The N.mPc library includes hardware and software descriptions for many popular microprocessors such as the Intel 8085/86, Motorola 6800/68000, RCA 1802, Zilog 80, etc. Also included in the library are a number of simulation examples. Two of the examples were built and run and their immediate benefit was twofold: Firstly, they proved that the simulation package had been properly installed, and secondly, they allowed familiarity and experience to be gained with the simulation package.

The first example involves an RCA/1802 processor with internal memory which runs a program that sorts a list of numbers. It is a simple simulation as the processor does not have any port to interface to other hardware. The topology that describes the hardware connections is very simple and consists of only three lines. Files related to these two examples can be found in Appendix I of [LAFE83]. The ISP' hardware description of the RCA 1802 as well as that of the Intel 8085 are also included.

The second example simulates eight Intel 8085 microprocessors connected to a distributed pipe system. This example is more complex than the previous one as it involves three types of hardware devices: Intel 8085, Memory and First In First Out (FIFO) Memory. The Simulation diagram is shown in Figure 2.2. The Source and Sink are only the special cases of FIFO memory: they are output only and input only FIFO's.

An identical program is loaded into all memory modules and executed by the 8085's. The program simply reads from the incoming FIFO memory and writes its contents into the outgoing FIFO memory. Eventually the data generated by the Source will arrive at the Sink. This is a trivial example of a pipe system. In a real system, the data read from the incoming FIFO would be processed before being written to the outgoing FIFO.

2.3 MULTIPROCESSOR SIMULATION I

Having successfully completed the simulation examples, a target was set to simulate a multiprocessor system related to a spacecraft application. An example of the multiprocessor system is shown in figure 2.3, in which the single board computers are indentical and each of them is capable of replacing the other in case of a failure. A common bus is needed to establish a communication link among the single board computers and other electronic devices.

The Intel 8085's were used in the simulation since the description of this processor was available from the N.mPc library. An Intel designed bus known as "Multibus" was also chosen for the common bus. It is a parallel bus with a well defined

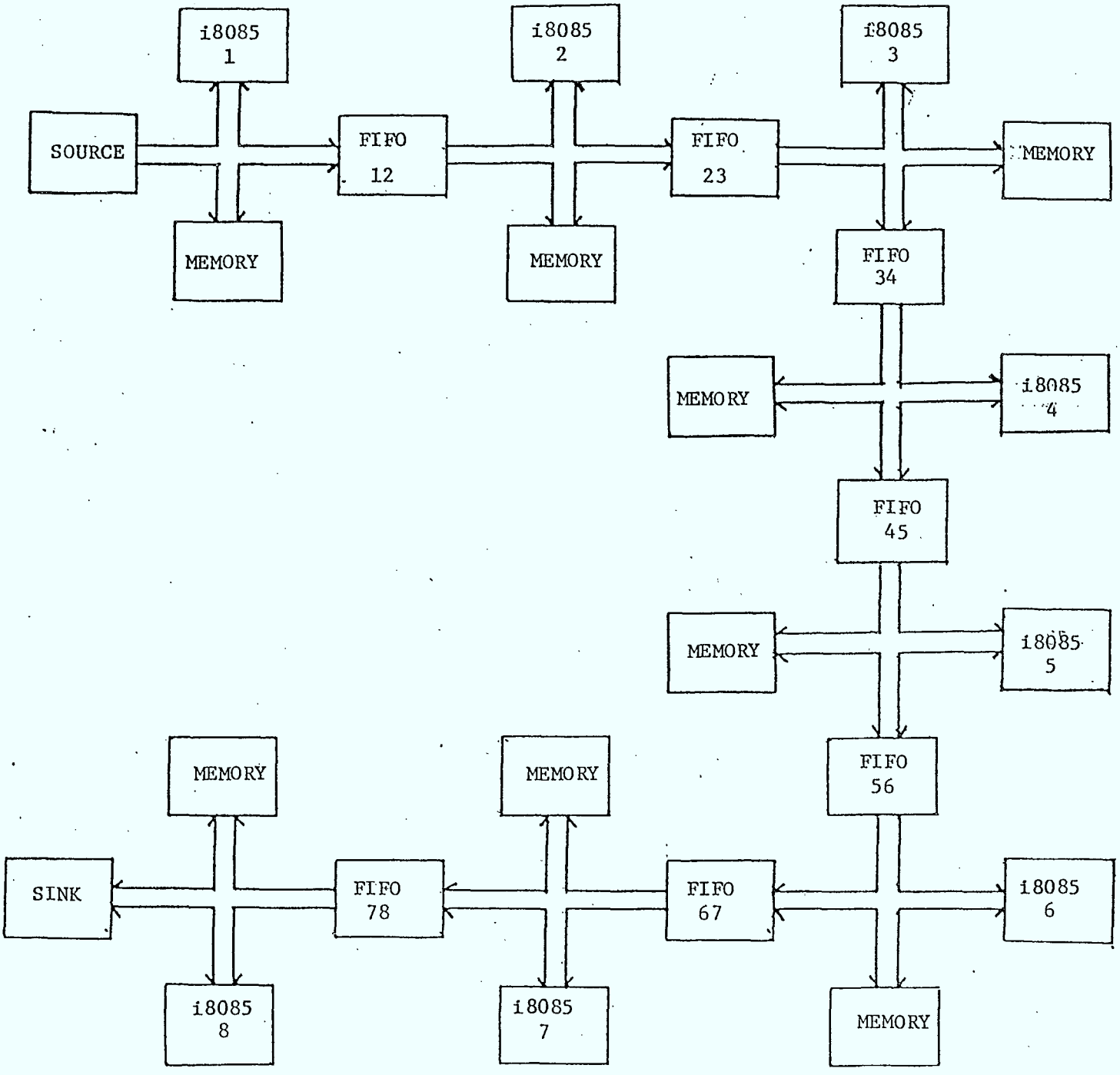

Figure 2.2 SIMULATION BLOCK DIAGRAM FOR DISTRIBUTED PIPE SYSTEM

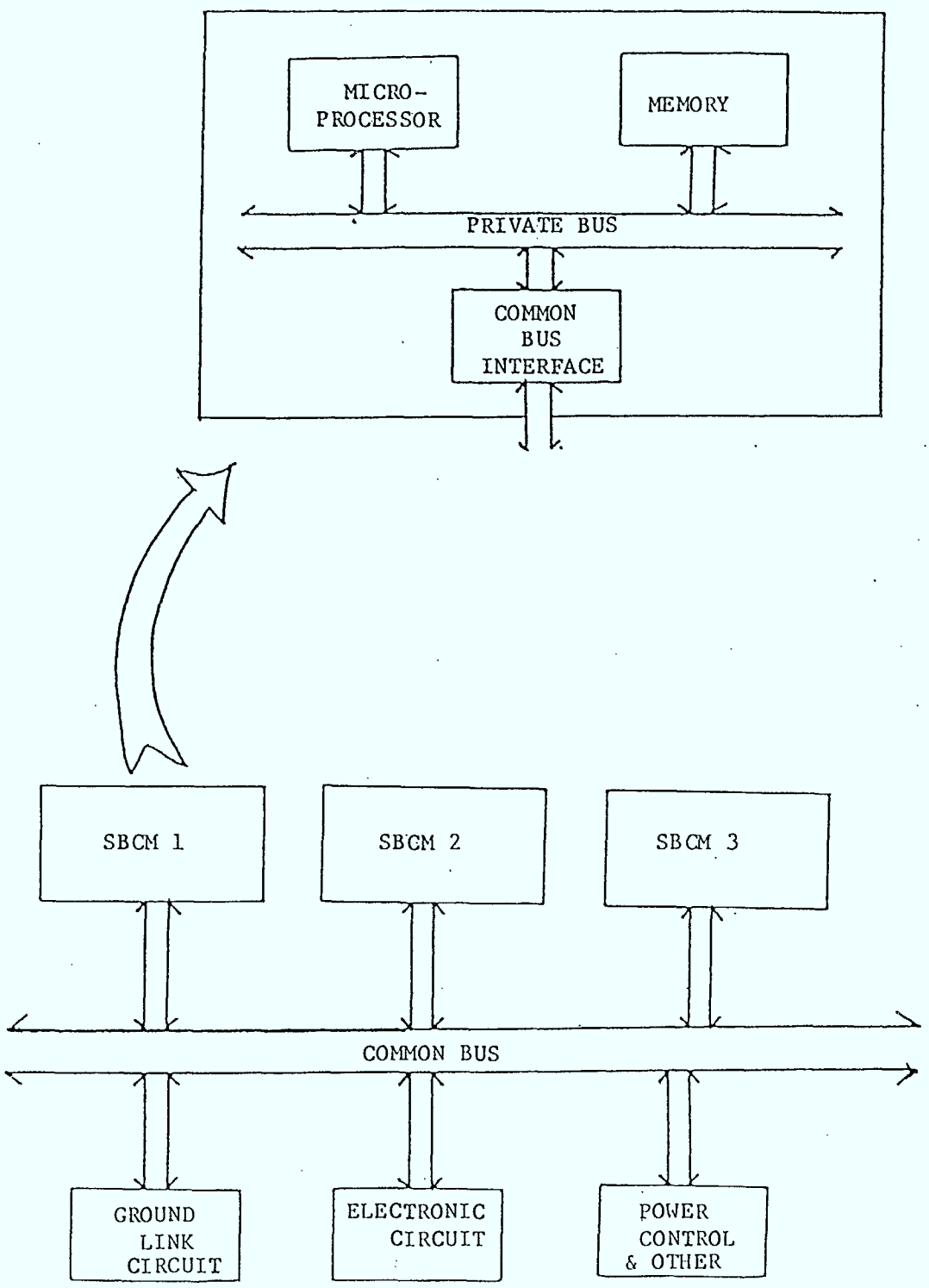

Figure 2.3 EXAMPLE OF SPACECRAFT MULTIPROCESSOR SYSTEM

architecutre and is documented in the literature [BART80].

In the simulation, five Intel 8085 based computer modules and a global memory module are connected to the Multibus system, as shown in Figure 2.4. A simulated terminal using the ISP' "raw memory" structure is connected to the private bus of the first simulated computer module. This module executes a program that reads a pair of numbers from the terminal, converts the numbers from ASCII to binary values and writes the values to the global memory. The other computer modules read the two numbers, process them and store the sum, difference, product and quotient into the global memory. Meanwhile, the first processor is waiting for the completion of those operations so that it can display the results on the terminal once they are available. Listings related to the Simulation I can be found in Appendix II of [LAFE83]. The following sections describe the simulation in details.

2.3.1 INTERPROCESSOR CO-ORDINATION

Two versions of the simulation have been developed using different co-ordination methods: Polling and Interrupt. In the Polling version, the four processors handling arithmetic operations poll the status word and read the numbers from the global memory after the status word is set by the first processor. The first then polls the "result ready" status word of each of the four processors and reads the results from the global memory into its own private memory.

In the interrupt version, the processors wait for an interrupt instead of continuously polling. In the present architecture, the advantages of this method are not so important since the processors

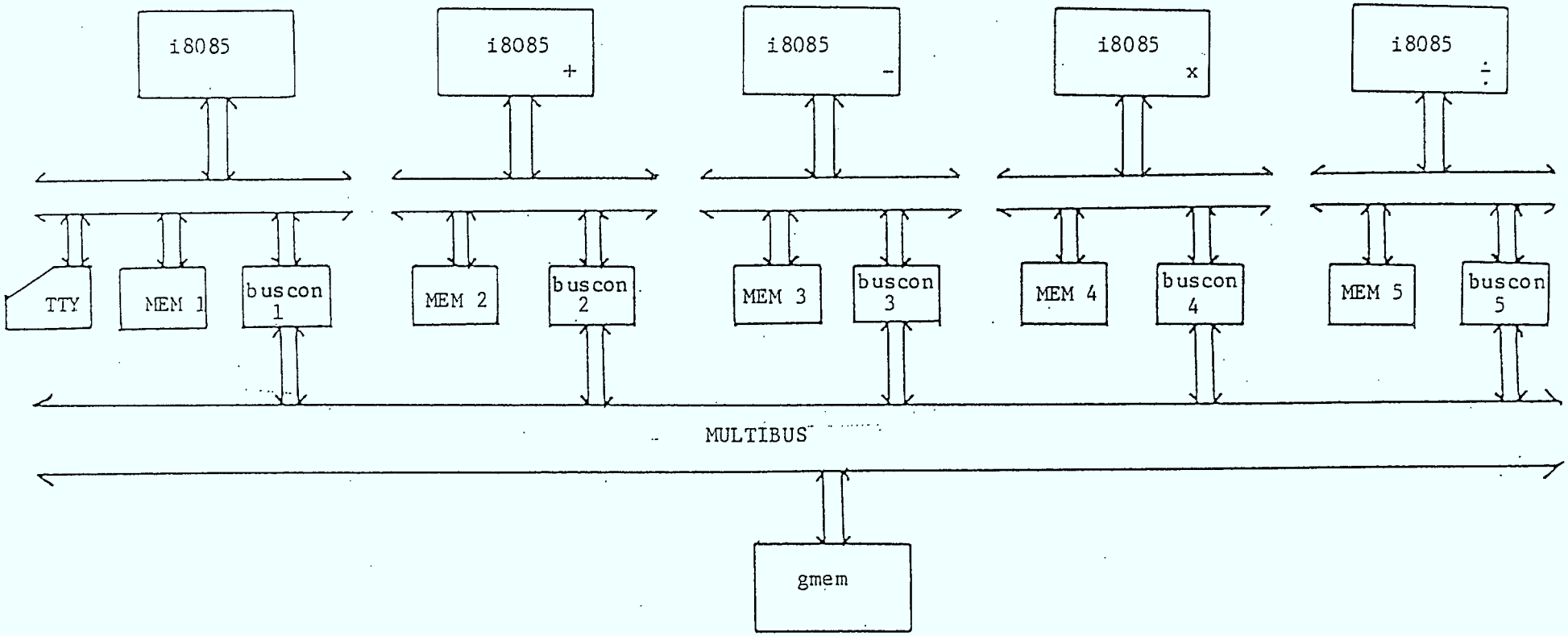

Figure 2.4 BLOCK DIAGRAM OF MULTIPROCESSOR SIMULATION I

are lightly loaded and the common bus is only used for synchronization. It remains, however, that co-ordination through interprocessor interrupts reduces the traffic on the common bus and also allows the waiting processors to do other tasks if so required. In most systems, the overall performance would be improved. Incidentally, from a simulation's standpoint, co-ordinating processors by interrupts has the extra advantage of reducing the size of the "ready to run" queue, thus affording faster execution to the other processors.

2.3.2 PRIVATE BUS DEFINITION

The 8085 processors use their private bus to fetch instructions and local data from their private memories. The private bus also serves as a communication link between the processor, its own I/O devices and the bus coupler. The private bus signal definition are as follows:

ah0 - ah7	:	higher order address
ad0 - ad7	:	multiplexed lower address and data
st0 - st1	:	status signals, 01 = write, 10 = read
iolm	:	addressing memory (high) or I/O (low)
ale	:	address latch enable. To indicate that address in the bus is stable
rd	:	set on completion of a read command
wr	:	set as data is stable on the multiplexed bus lines
wait	:	to halt the processor

2.3.3 MULTIBUS OPERATIONS

In a Multibus system, there are two types of modules that are connected to the bus: Master and Slave. A computer module is an example of a bus master and a memory module that of a bus slave. The bus master makes a bus request, waits until it is granted, addresses a bus slave and commands the bus slave to provide a

service. The slave replies with an acknowledgement when the task is done and the bus master then releases the bus. A multibus system may have more than one bus master connected to it and a bus arbitration scheme is required. This arbitration is carried out by the bus coupler.

Two priority resolution techniques are used in multibus systems [BART80] : Serial and Parallel Priority techniques. In the Serial Priority technique, the priority resolution is accomplished with a daisy chain scheme. Each bus coupler has a priority input and a priority output pin. In the daisy chain scheme, the input of a bus coupler is connected to the priority output pin of the previous bus coupler. When a bus request is initiated by the processor, the bus coupler waits until a bus grant signal is received from the priority input pin. Meanwhile the processor has to wait until an acknowledgement from the bus coupler is received. If no bus request is pending and the bus coupler receives a bus grant signal, it simply sends the signal through the priority output pin, passing the bus grant to the next bus coupler.

In the parallel priority, the priority output and input pins are tied to a Priority Encoder/Decoder Circuit. The circuit sends the bus grant signal starting with the bus coupler of highest priority, waits for the bus grant signal to be returned and sends it to the next highest priority. This process is repeated.

The serial priority scheme is simple although not the most efficient. The parallel scheme requires extra hardware circuit and is more complex. For this simulation, the serial scheme was used.

2.3.4 MULTIBUS COUPLER

The bus coupler interfaces the processor to the multibus. It can recognize a bus request made by the processor and sends the data and commands to the common bus once they becomes available.

The ISP' hardware description of the bus coupler can be found in Appendix II of [LAFE83]. As far as the processor is concerned, the bus coupler is a memory module that responds to addresses in a pre-defined range. Read and Write operations are possible and the bus coupler will delay the processor when a request is still pending.

The simulated bus coupler is unrealistic since it needs more than 60 pins to interface to a multibus. In reality, several integrated circuits are required. For example, a bus coupler implementing a daisy chain scheme would require a Bus Arbiter (Intel 8289), a Bus Controller (Intel 8288), an Address Latch (Intel 8283/8282), and a Data Tranceiver 8286/8287 [BART80].

2.3.5 MULTIBUS SIGNAL DEFINITION

The Multibus is responsible to carry data and commands originating from a bus master and destined to a bus slave. The simulated bus consists of the following signals:

Address 0 - 15	:	address
Databus 0 - 8	:	data
Read	:	high true read command
Write	:	high true write command
Ale	:	address latch enable
Xack	:	transfer acknowledge. Signal emitted by a bus slave to indicate completion of a service.
int 0 - 7	:	interrupt lines

During a Read operation, the read signal is set to true and the address is loaded onto the Address bus. At the same time, a strobe is sent down the Ale line. The bus slave recognizes the address and the command. It then loads the memory contents of the addressed location onto the Databus. An acknowledgement is then sent through the Xack.

The Write operation is similar in principle. The write interrupt line operation simply sets any of the eight interrupt lines and there is no acknowledgement.

Using those three mechanisms, communication between bus components is achieved. Furthermore, interrupts can be used between bus masters to provide better co-ordination of data (or control) transfer.

2.4 DEFINITION OF NEW HARDWARE/SOFTWARE STRUCTURES

2.4.1 INTRODUCTION

This sub-section describes the N.mPc implementation of a new processing element, the Ferranti F100-L. This work was undertaken for several reasons: Firstly, defining processors and instruction sets is one of the great advantages of N.mPc. It allows a designer to use a combination of existing microprocessors, special purpose microprocessors and other special devices implemented as programmable logic arrays. Secondly, by implementing new structures, it becomes possible to test some special features of N.mPc which would otherwise not have been used. For example, the resulting size of certain simulations required careful treatment. Thirdly, implementing a new processor yields a complete development

system, ready to be put to use. Such a development system includes an assembler/linker, and a total environment for testing programs.

The Ferranti F100-L was chosen as the processor to be implemented for two reasons:

1. It is a 16 bit processor built using bipolar technology. Because of this fabrication process, the F100-L is radiation hardened and thus suitable for a spacecraft environment.

2. The F100-L is also the processor chosen for the European Large Satellite (L-Sat) program.

A full N.mPc implementation of the F100-L involves command files for the metamicro assembler and the linking loader, an ISP description of the F100-L itself as well as of the necessary support chips (F113 memory interface, interrupt controller, etc.).

The following two sections will describe each of those items in detail. The hardware aspects of the implementation will be treated first, with descriptions of the processor and of a system based upon it. In the second part, the software aspect, more precisely issues of syntax, addressing modes, address resolution, etc. will be dealt with.

2.4.2 HARDWARE ASPECTS OF THE F100-L

2.4.2.1 PROCESSOR DESCRIPTION

As mentioned previously, the F100-L is a 16 bit processor [FERR81a] with a multiplexed data and address bus, referred to as the system "highway". The total address space of the F100-L is 64 kilo bytes, addressed as 32k 16 bit words. Incidentally, the F100-L does not have byte addressing capability. A block diagram of the internal organization of the F100-L is shown in figure 2.5. From that block diagram, it can be observed that the F100-L does not

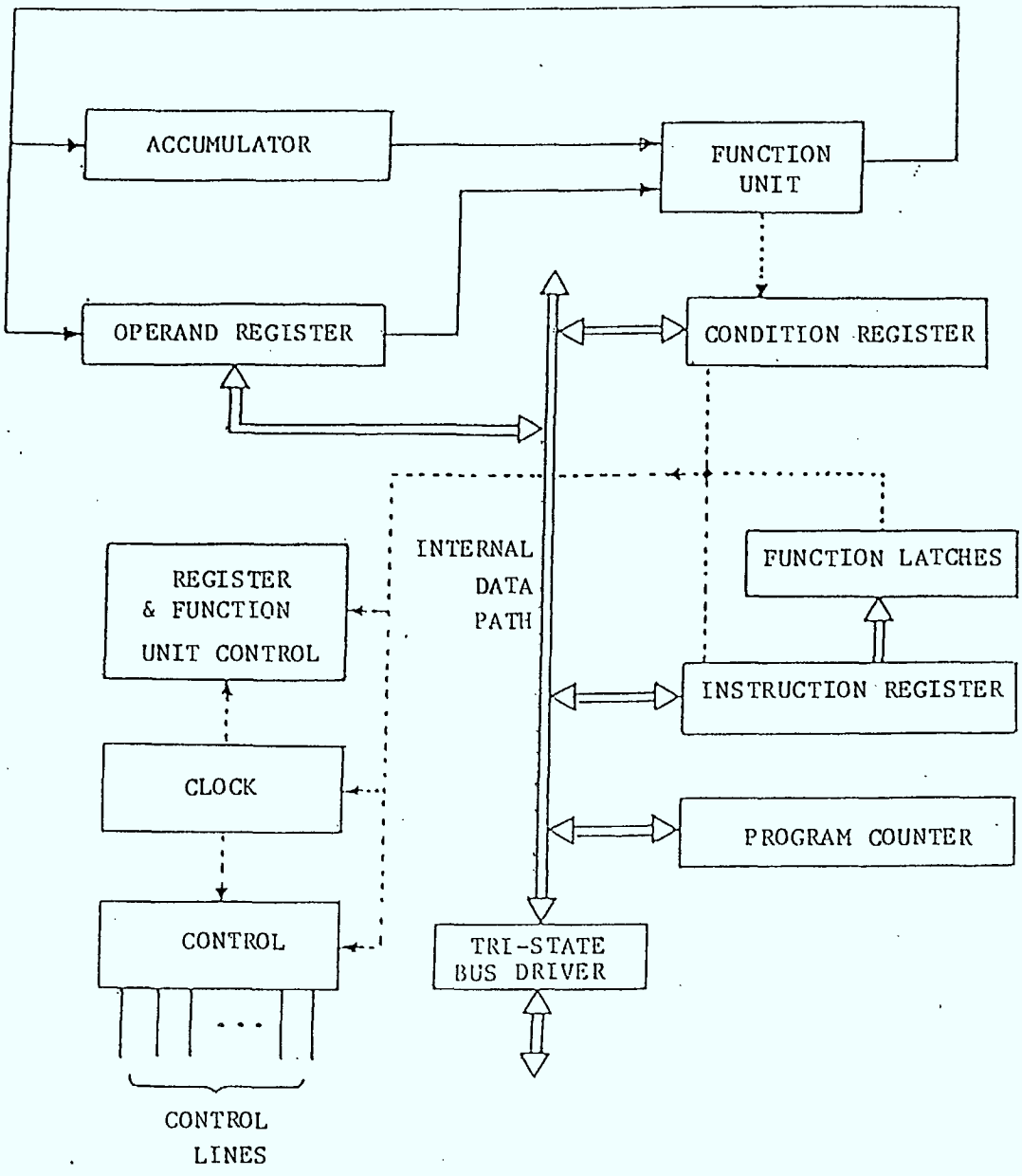

FIGURE 2.5 Block Diagram of F100-L

possess a bank of general purpose on-chip registers. Instead, the F100-L has a 16 bit accumulator and an operand register to be used in data manipulation and testing. A seven bit condition code register can also be used in a similar fashion although its primary function is to record the status of arithmetic and logical operations.

On the control side, the F100-L possesses a program counter used in the traditional way and an instruction register which holds the instruction to be decoded. There is no on-chip stack pointer, but instead the stack pointer is stored in location 0 and is used in subroutine calls and interrupt processing. Figure 2.5 also shows other control structures which are necessary to proper hardware operations and whose functions will be emulated by the ISP description of the processor.

The F100-L also possesses the necessary interface to support architecturally compatible co-processors. One such co-processor is the floating point processor which performs floating point multiplications and divisions in hardware. Using co-processors is very advantageous as they are usually fast because of their special purpose nature, and also easy to integrate into the system. In the N.mPc implementation of the F100-L, co-processor were not to be implemented and, consequently, the necessary hardware features to support them were not included. However, incorporating them in future implementation should be relatively easy.

The Ferranti microprocessor manual [FERR81a] describes most of the features of the F100-L. Some of the F100-L's characteristics, however, deserve special mentions, especially in the context of the current implementation. Of particular interest

are the memory access mechanisms, the Direct Memory Access (DMA) arbitration and the interrupt mechanism. The memory access mechanisms will be covered first, since they are the lowest level of implementation detail to be discussed.

There are three basic mechanisms for accessing memory in an F100-L system: Read, Write, and Read Modify Write. The timing diagrams for all three mechanisms can be found in Figure 2.6. Co-ordinating memory processor interaction is carried out through the use of four lines: JACV, KACV, JPAS and KPAS. The suffix ACV indicates a signal line originating from the F100-L whereas PAS indicates a signal line destined to it.

Usually, the F100-L does not interact directly with memory modules, but rather with a memory interface module. Details of such a module will be covered later in the description of the system. An interesting feature of the F100-L is that, unlike microprocessors such as the Intel 8085 which only have Read and Write cycles, The F100-L has a Read Modify Write (RMW) facility. Using this RMW feature saves the time necessary to send the address a second time for the Write portion of the cycle. RMW is very useful when pointer operations with auto increment or decrement are being done. On the negative side, because of the need to maintain the address between the Read and the Write portions of the transfer, DMA (to be discussed next) cannot be allowed.

The F100-L has "built in" facility for the handling of DMA requests (DMARq) and grants (DMAAccept). DMA processing has been implemented as an autonomous activity, triggered by the action of the DMA request line. Figure 2.7 shows the timing

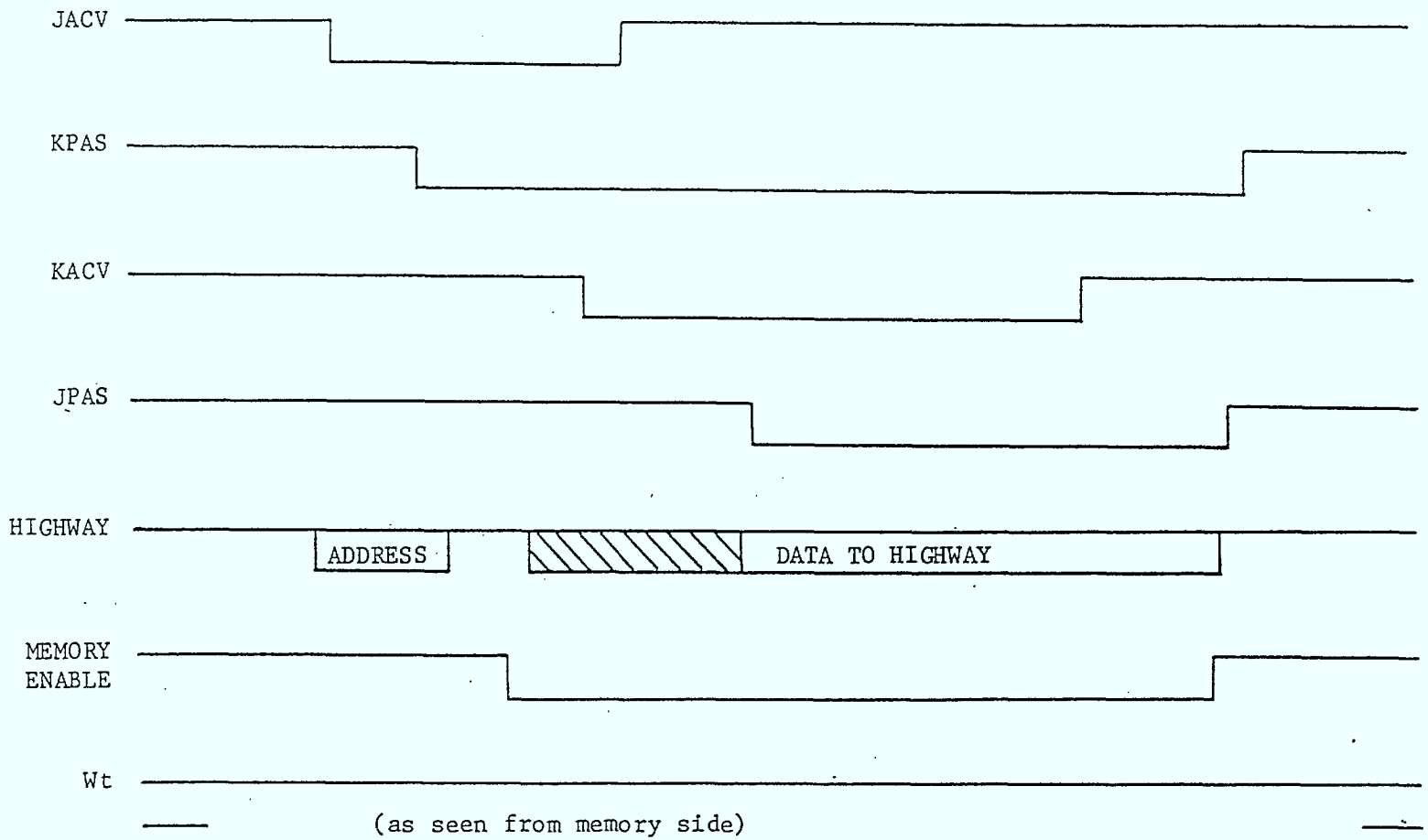

FIGURE 2.6 (a) Timing Diagram of Read Cycle

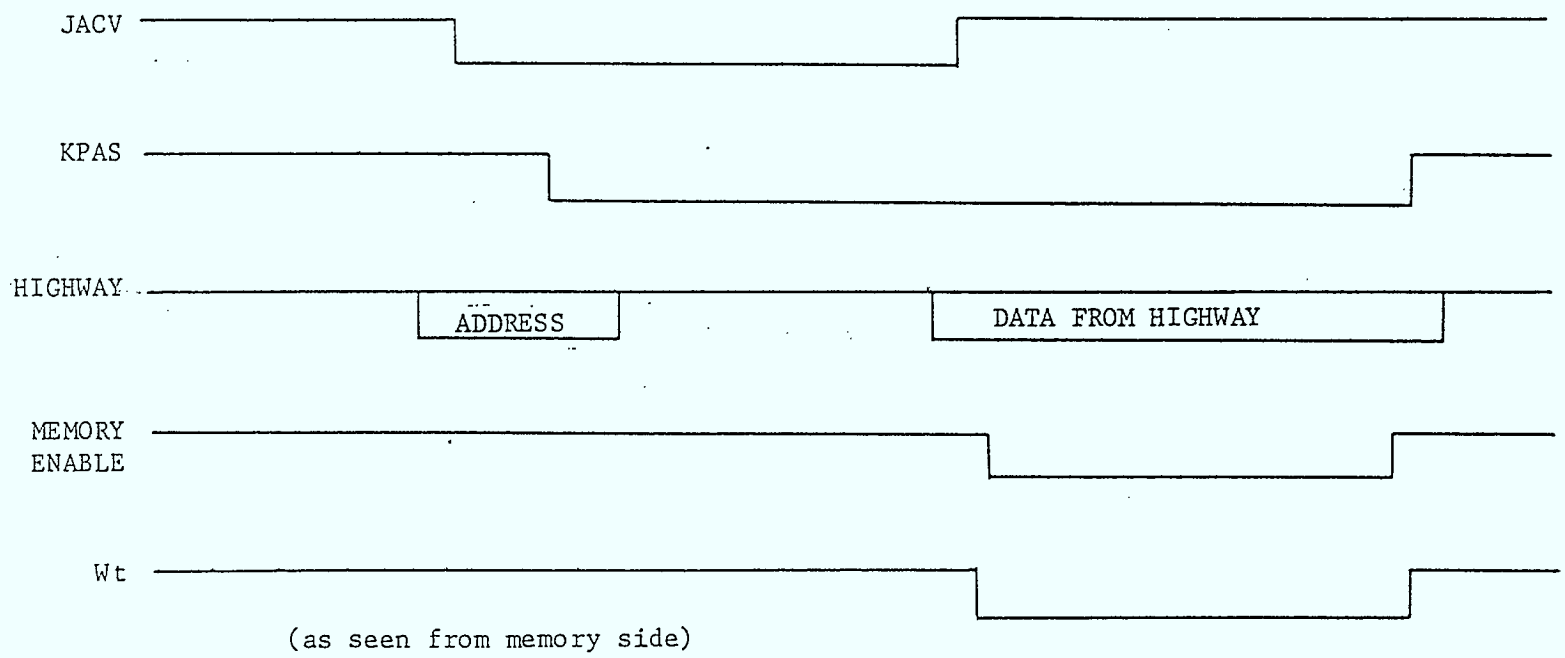

FIGURE 2.6 (b) Timing Diagram of Write Cycle

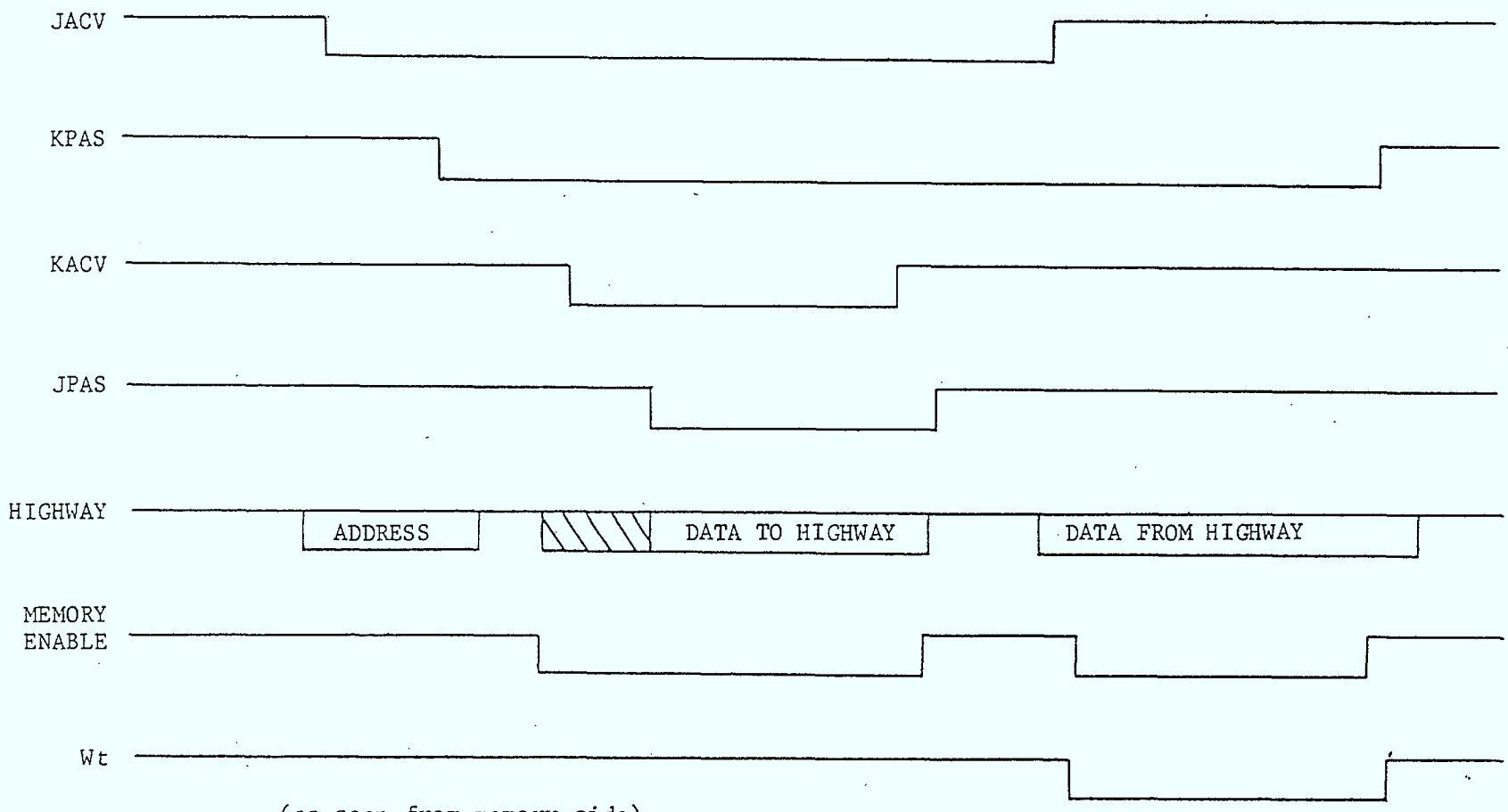

FIGURE 2.6 (c) Timing Diagram of Read/Modify/Write Cycle

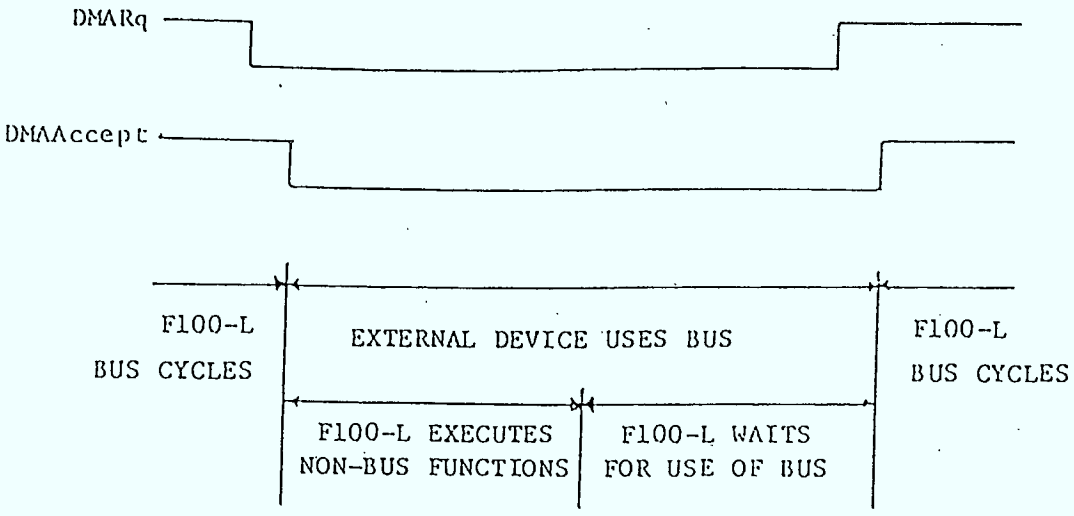

FIGURE 2.7 Timing Diagram For Direct Memory Access

diagram for DMA requests and grants. The memory access mechanisms (i.e. Read, Write, RMW) and the DMA process compete with each other for the utilization, or more precisely for the ownership of the highway. Mutual exclusion on the use of the bus is, of course, mandatory, since the bus is a unique physical resource which cannot support more than one user. In other words, should a CPU originated memory access occurr concurrently with a DMA device accessing memory, unpredictable effects would result. Mutual exclusion is therefore provided by two pairs of procedures (one pair for memory accesses, another for DMA) implementing DEKKER's algorithm. The algorithm guarantees that race condition will not happen and that starvation of one of the requesters will be avoided. It is interesting to note that DMA processing is essentially asynchronous and is used in the vectoring of interrupts which is the next item to be discussed.

Interrupt processing is a higher level function than both DMA arbitration and memory accesses. Two types of interrupts can be handled by the F100-L, vectored and non-vectored. A non-vectored interrrupt, will be triggered by the activation of the Program Interrupt Request line (PgItRq) which is scanned by the processor at the end of each instruction. If the interrupts were not specifically disabled by the program, the processor will grant the interrupt, thereby activating the Program Interrupt Acknowledge (PgItAccept) line in response to prior activity on PgItRq. The current status of the interrupted program is saved and a jump to the location of the interrupt service routine is executed. (In actual fact, the program counter gets 2050 and the interrupt service routine may reside there, or, as is more common, 2050

contains a jump instruction to the beginning of the routine).

Non-vectored interrupts have several drawbacks. The source of the interrupts has to be determined by polling the devices, a task which is usually done in software. The response time of non-vectored interrupts is thus slowed down by the software polling. The F100-L supports vectored interrupt provided that either the device interrupting or an interrupt controller is capable of requesting DMA, putting a channel number on the highway and strobing it (with ExtLdPgCt). The timing diagram of a vectored interrupt sequence is shown in Figure 2.8.

Strictly speaking, the device sends twice its pre-assigned channel number to the F100-L. The F100-L, after having saved the status of the interrupted program, will load the program counter with 2050 offset by the channel number of the device. Therefore, to each channel is assigned a couple of words which contain a jump to the appropriate interrupt service routine.

2.4.2.2 SYSTEM DESCRIPTION

In order to obtain a working system that can be used in a simulation, the F100-L has to be connected with support chips. The tasks performed by those chips are as follows:

1. Memory Interface. This kind of support chip is essential to decode the signals for Read, Write, Read Modify Write and to demultiplex the data/address highway.

2. Interrupt Controller. This support chip(s) is necessary if vectored interrupts are required. The controller may be a distributed entity (e.g. F111 + F112) or a single central controller chip (e.g. the interrupt controller in the listings).

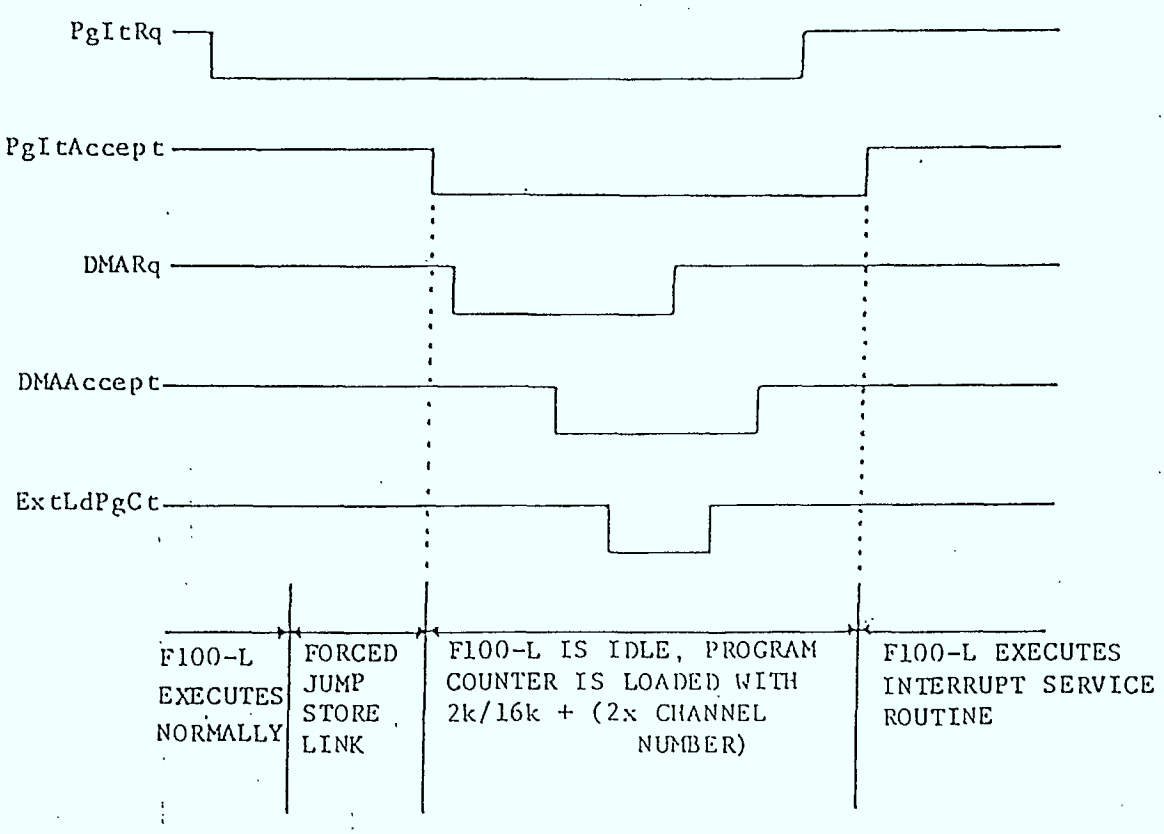

FIGURE 2.8 Timing Diagram of Vectored Interrupt

3. Direct Memory Access (DMA) Controller. If this function is desired in the system, a DMA controller is necessary. As in the previous case, control can be distributed (e.g. F111 + F112 again) or centralized. DMA by itself was not needed in the multiprocessing examples and, although the implementation of the F100-L supports DMA, a DMA controller was not provided.

In the early stages of system design, two possible ways to design a system were considered:

1. A simple approach was to use an F113 memory interface chip and implement it in ISP. The F113, to be described shortly, is a passive device (i.e.it does not initiate DMA transfer) and is used solely to interface the F100-L to some memory modules. Interrupts being an essential feature, the simple approach proposed building an interrupt controller in ISP. The design of the controller was to be similar to that of an Intel 8259.

2. A more complicated approach, albeit more versatile, was to use the standard Ferranti interface set, which is made out of one F111 control chip and two F112 data chips. This interface set is very powerful; it can handle interrupt and DMA requests, and it supports a daisy chain priority scheme for interrupt and DMA requests. Higher priority is given to interrupting devices which are closer to the processor. Finally, it can function as a multimode device with all the modes supported by the interface set being described in [FERR81c]. Of particular interest, is the fact that this standard Ferranti interface set can also be used in multiprocessing systems.

The second approach is clearly more flexible in general purpose systems. However, spacecraft systems are of a special purpose nature and furthermore, the F111 + F112 interface set is very complex due to its multimode functionality. For these reasons, and also because of the limited time available, the simple approach was chosen.

A complete F100-L based system using the F113 memory interface, a special memory module, and an interrupt controller, is shown in figure 2.9. In that system, the co-ordination between the F100-L and the F113 is as described in the discussion of memory

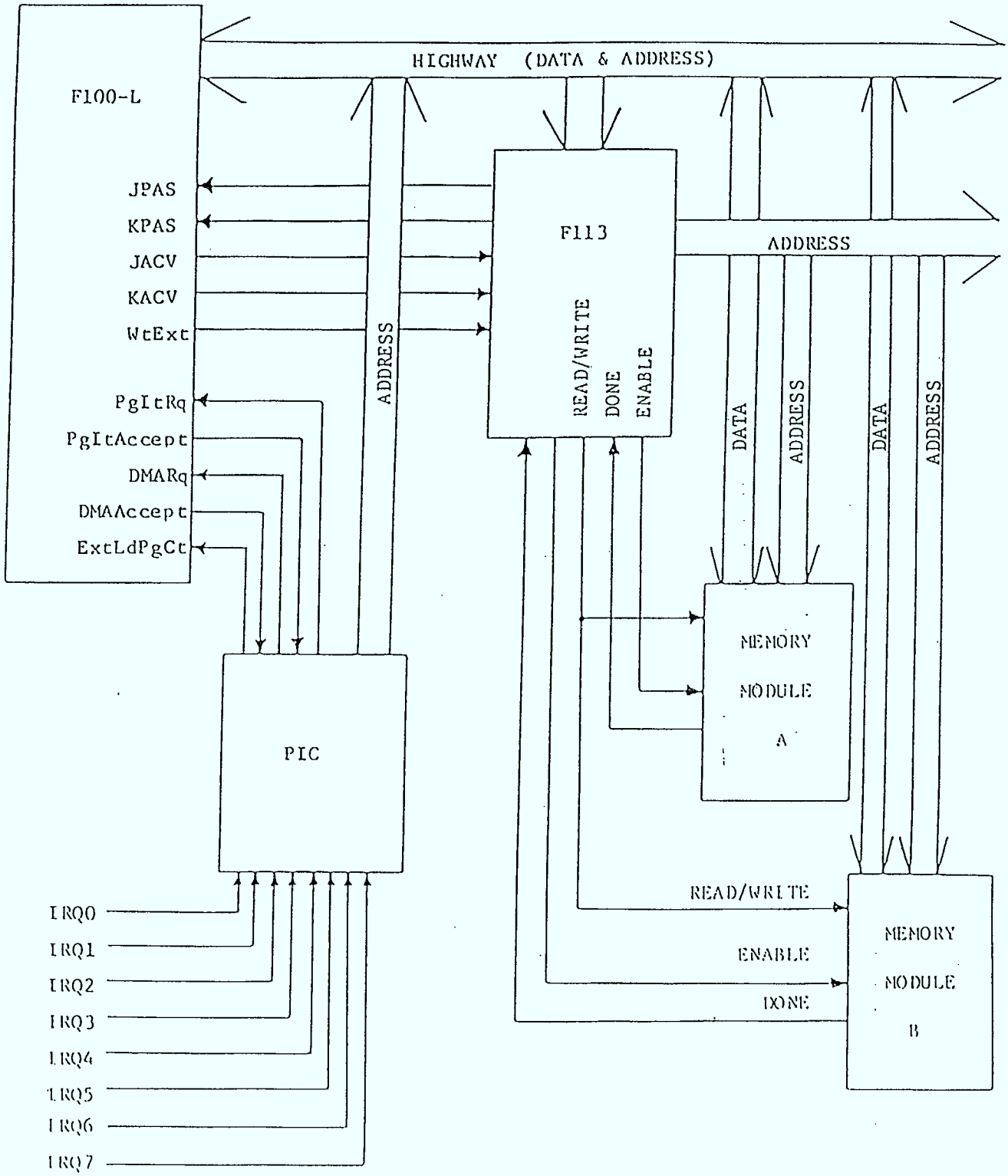

FIGURE 2.9 Basic F100-L Based System

timing.

The F113 memory interface ISP' implementation is slightly different from the real F113. The difference is that in systems where the F113 is normally used, an address register demultiplexes and holds the address selected by the processor. This register is external to the F113 but, for ease of implementation and also to simplify the interconnection of system elements, the register and its functions were incorporated in the F113.

Another characteristics of the F113 is the provision for two external RC circuits to serve as timing indicators. Usually, interaction between a memory interface and a memory module is accomplished by an "enable" signal from the interface and a "done" signal from the memory module. In the case of the F113, an "enable" signal is used to trigger the memory module but the "done" signal from the memory is absent. Instead, the F113 uses an RC circuit timer to produce the equivalent of a "done" signal from memory. There are two such circuit connections on the F113, one for each of the two memory channels supported by the interface.

The current implementation of the F113 does not use RC timing circuit equivalent. Instead, it uses the more traditional enable/done arrangement described previously, thus making memory access co-ordination easier to visualize. When the system is part of a multiprocessor simulation, a bus coupler/arbiter is used and the bus coupler has to be able to delay the interface and the processor. This is so because global memory is accessed through the bus coupler/arbiter and the access time is dependent upon the amount of contention on the global bus. If RC timing circuits were used, an upperbound on global memory access time would have to be

chosen with a corresponding decrease in performance.

The memory modules used in the F100-L based system are of a straightforward design. They respond to an "enable" signal from the memory interface, read the read/write line to determine which operation to perform and signal completion of read or write by a "done" signal. The address selected by the processor is available directly from the memory interface and is stable throughout the operation. In the case of a Read Modify Write cycle, the address is stable from the beginning of the Read to the end of the Write. (Of course, the memory modules would not know whether a Read Modify Write is in progress or not, since they only see Reads or Writes).

The last component of the system is the Interrupt controller. The internal architecture of the controller is shown in Figure 2.10 and is, in fact, quite simple. A pair of registers is used to notify the controller of the occurrence of an interrupt, (Interrupt Request Register) and to "double buffer" those same requests (In Service Register). The controller supports eight independent request lines and implements a line scanning scheme which emulates the daisy chaining of request lines. Line 0 has the highest priority and line 8 the lowest. The controller uses DMA to transmit the channel number (in fact, twice the channel number) to the F100-L over the highway. The controller meets all F100-L timing requirements for interrupt processing. It is interesting to point out that, following an interrupt request and grant, the F100-L will wait for a few clock cycles for a DMA request to occur. This expected DMA request should be from the controller as a first step to sending the device channel number.

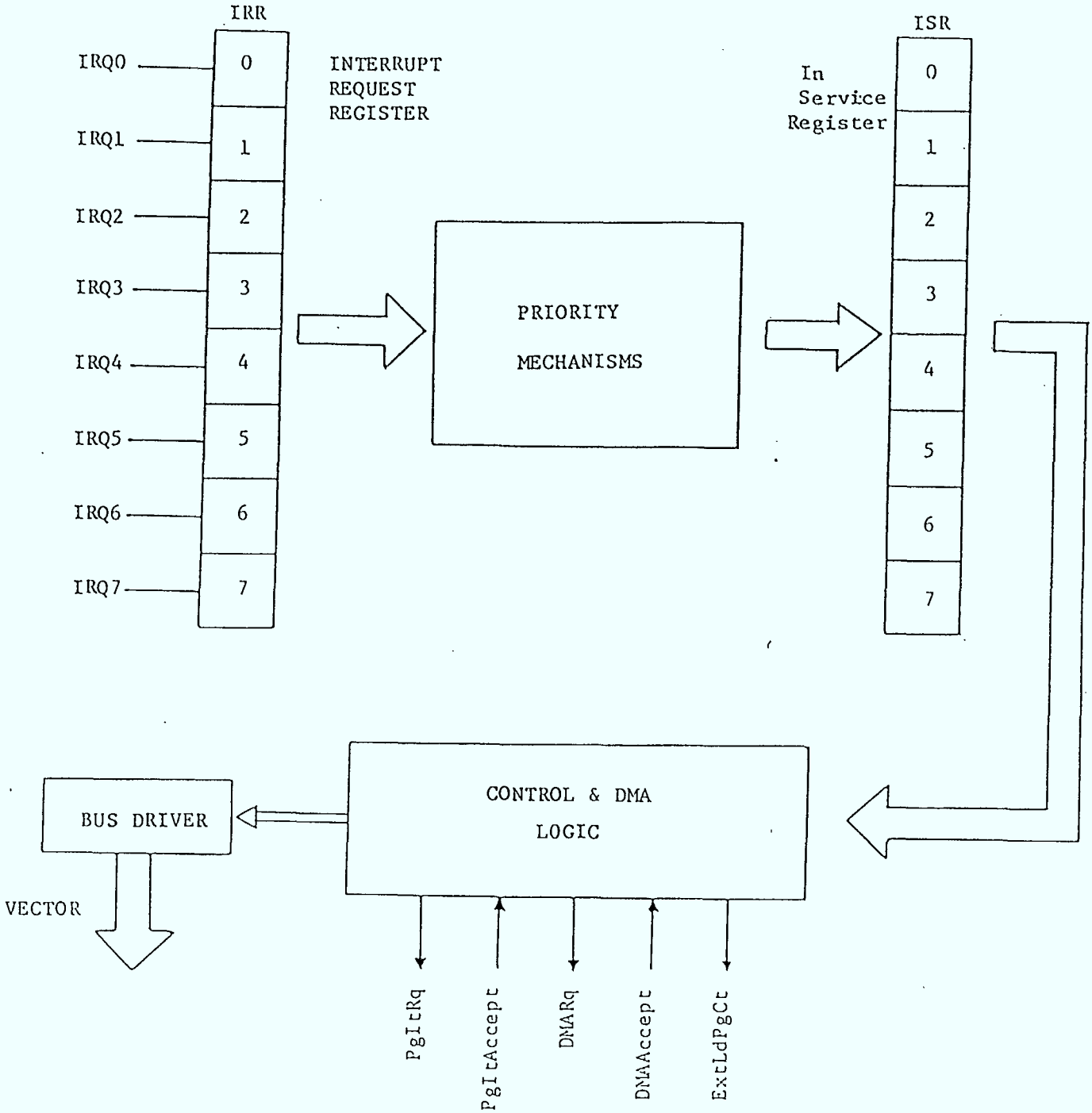

FIGURE 2.10 Internal Architecture of Interrupt Controller

Incidentally, the channel number is strobed by the controller by activating the ExtLdPgCt line.

In the case of a system where vectoring of interrupts is not necessary, no DMA request should be made following an interrupt acknowledge. The F100-L, noticing the absence of DMA request, will assume that no vectorry will take place and will simply load 2050 into the program counter.

It is obvious that following an interrupt acknowledge, no DMA request other than that of the interrupt controller should be allowed until the end of the interrupt processing sequence (as signified by PgItRq going to inactive followed by PgIt Accept). If it were not so, it could happen that another device would make a DMA request which would be interpreted by the processor as being from the interrupt controller. The processor would then mistakenly wait forever on the strobe (ExtLdPgCt) signal from the controller.

In the system of Figure 2.9, the interrupt controller is the only device using DMA. If some DMA devices were incorporated in the design, the interrupt controller would have to be modified to take care of this potential deadlock problem. In the standard Ferranti interface set (F111 + F112) the problem is handled by daisy chaining all DMA and interrupt requests through the same chips and by disabling the DMA request chain as soon as an interrupt request is posted. This arrangement is shown in Figure 2.11. The F111 control chip is the only part of the interface set involved in the daisy chain and its functional description does not cover the difficulty with DMA request timing.

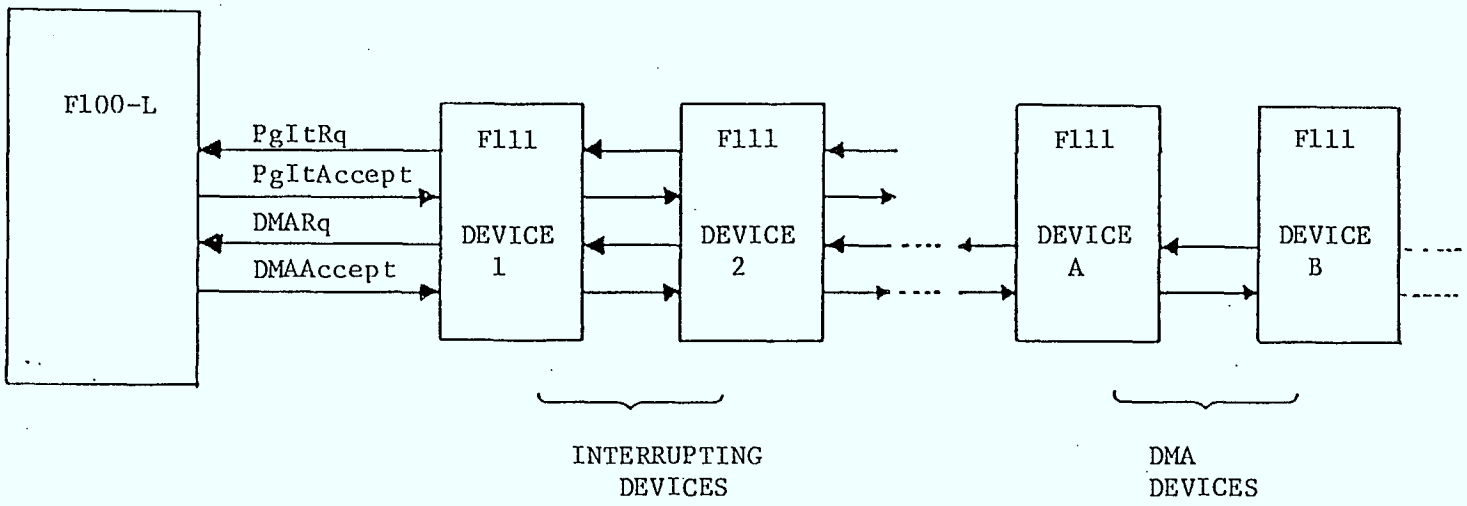

FIGURE 2.11 Daisy Chaining in Standard Interface Set

As mentioned before, modifications should be made to the controller if other controllers or purely DMA devices are used in the system. In fact, an arrangement very similar to that of the standard Ferranti should be adopted. Two possibilities come to mind:

1. Coupling of DMA and interrupt request lines can be achieved in exactly the same fashion as the F111 + F112 interface set. This means that all controllers (DMA and interrupt are daisy chained together.

2. Alternatively, a central DMA/interrupt super controller could be designed and implemented in ISP´. This super controller would perform DMA and interrupt requests arbitration.

2.4.2.3 ISP´ PROGRAM LISTINGS

This section introduces and comments on the ISP´ program listings which are to be found in Appendix III of [LAFE83]. The listings are, in order of appearance:

1. Ferranti F100-L ISP´ description,
2. Ferranti F113 ISP´ description,
3. Memory Module ISP´ description,
4. Interrupt Controller ISP´ description.

The ISP´ programs are all extensively documented, and their purpose and functions should be readily understood. ISP´ is not, however, a traditional language but rather a register transfer language. Differences and peculiarities exist which make ISP´ programs look strange in some instances. For example, the statement "next" is required to make previous assignments take place. As well, a call by value structure forces the use of functions instead of procedures. The inability of ISP´ to accept the value of bit field as parameters results in awkward constructs

as found in the functions "bit-assign" and "bit-test" in the F100-L program.

2.4.3 METAMICRO ASSEMBLER

The metamicro assembler is a programmable assembler that allows users to define an assembler for a target machine. Using the meta micro, an assembler for the Ferranti F100 was developed. To use the assembler, a user writes his program in the target assembly language, concatenates his program to the F100 metamicro description file using the "include" statement and invokes the metamicro assembler. The output of the assembler is a nodal file that contains the machine code representation of the assembly language memonics and the data constants.

Label addresses are not resolved during the assembly and spaces for those labels are reserved by the metamicro. The task of address resolution is left to the linking/loader. A user's guide for the F100 metamicro assembler can be found in Appendix III of [LAFE83]. The guide contains a detailed description of the assembler syntax.

Due to some limitations of the metamicro and also partly to the fact that the F100 assembler [FERR81b] includes some features that are not standard in North American made assemblers, entire syntax compatibility can not be achieved. However, the UNIX operating system provides a tool called YACC (Yet Another Compiler Compiler) which is suitable for writing a pre-processor. This pre-processor could be used to modify the syntax of F100 source programs written for the special purpose F100 assembler, and to make it compatible with the F100 metamicro assembler.

2.4.4 LINKING LOADER

The main function of the linking loader is to allocate and resolve addresses that are found in the nodal files. The output of the linking loader is a loadable memory module for a target machine and this loadable memory module may include more than one nodal file. Assembly listings for each source file are also produced by the linking loader.

The linking loader requires a description file supplying information on how to resolve different addressing modes. For example, some F100 assembly instructions have four modes to address their operand(s):

1. Short Direct Addressing Mode: The address of the operand is the lower order 10 bits of the instruction word.

2. Long Direct Addressing Mode: The address of the operand is in the memory location next to that of the instruction word.

3. Immediate Addressing Mode: The operand is located in the memory location next to the instruction word.

4. Indirect Pointer Mode: The lower 8 bits of the instruction word form an address to a pointer that points to the operand. The pointer can optionally be auto incremented or decremented.

The linking loader must be able to recognize each addressing mode, check for the proper address values and produce an error message if any address value overwrites the op-code of the instruction.

The simulated memory where the program is to be loaded does not have to be contiguous. List of memory ranges can be defined in the linking loader description file and jump instructions to other available memory regions are automatically inserted by the linking loader. If the program requires more memory that is

defined in the description file, an error message is produced. Listing of the linking loader description file can be found in Appendix III of [LAFE83].

2.5 MULTIPROCESSOR SIMULATION II

The simulation described in section 2.3 is repeated with Ferranti F100-L processors replacing the Intel 8085's used previously. Some modifications have been made to the bus coupler and other supporting devices because of the F100's 16 bit architecture and its different memory cycles.

Listings of the simulation include the topology file and the source programs for each processor. Here again, two types of simulation were carried out, one performing co-ordination by polling, the other by interrupt. It was found in the simulation with interrupt that the simulation program grew too large and went beyond the 64K bytes memory limit imposed by the PDP-11 architecture and the UNIX operating system. The number of processors in the simulation had to be reduced by eliminating one processor and combining the sum and difference functions into one program.

As mentioned earlier, the F100 uses different memory access cycles, and therefore requires a special memory interface circuit. The block diagram of a F100 based computer module is shown in figure 2.9. It consists of the F100 micro processor a F113 memory interface, a programable interrupt controller and a memory module. Also present in the system but not shown in the diagram, is the bus coupler. The F113 was modified to accommodate a third channel so that "raw memory" could be used as well. For the polling version,

the interrupt circuit is redundant. A block of the simulation is shown in figure 2.12.

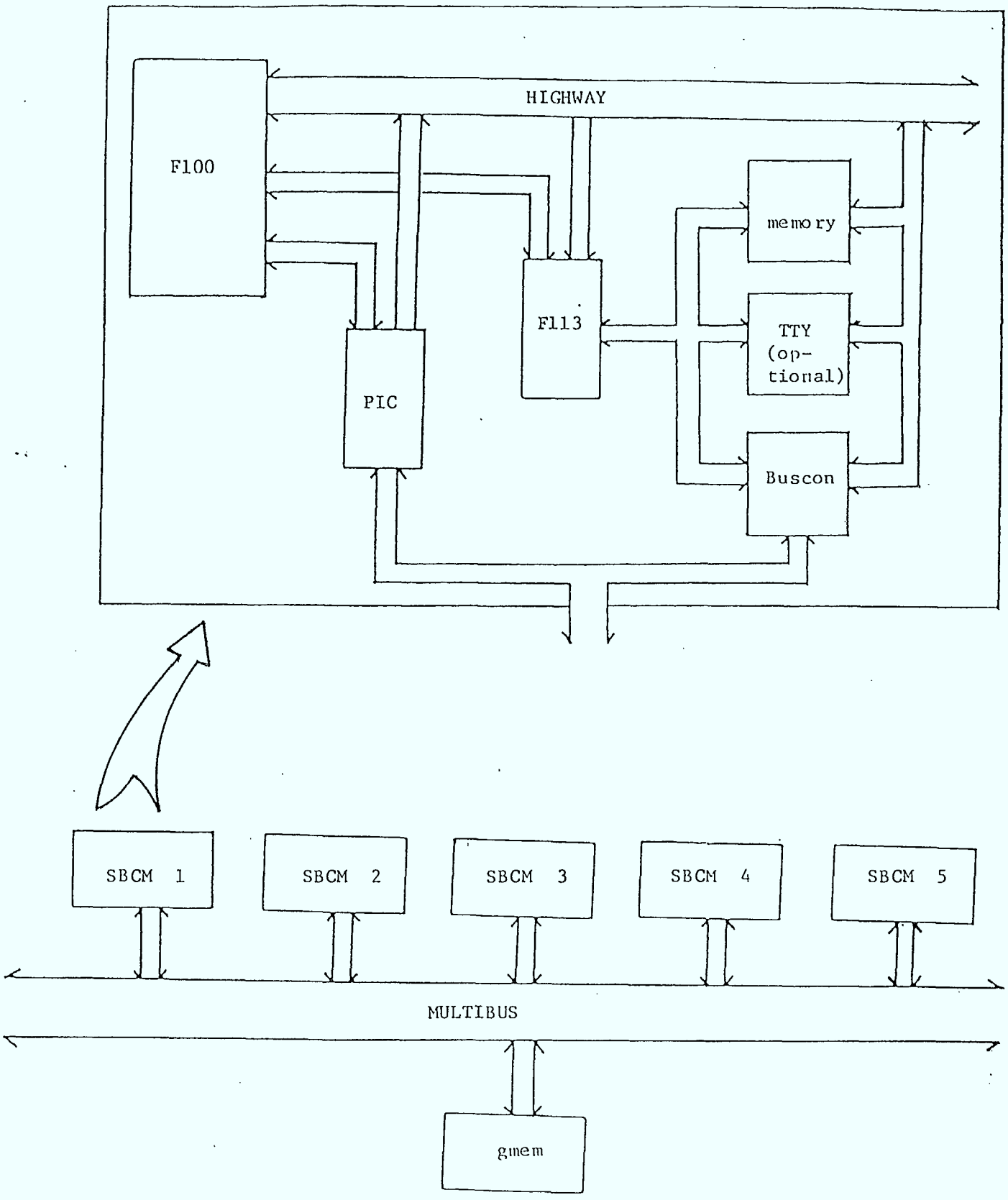

Figure 2.12 BLOCK DIAGRAM OF MULTIPROCESSOR SIMULATION II

3.0 UTILITY OF N.mPc

3.1 COMMENTS ON INDIVIDUAL COMPONENTS

3.1.1 METAMICRO ASSEMBLER AND LINKING LOADER [ROGE80]

The assembler developed using the metamicro has some limitations. It does not allow arithmetic expressions in the operand field of a mnemonic, which is a standard feature in many other special purpose assemblers. The assembler is not, therefore, as powerful as those special purpose assemblers.

Another minor limitation is that the metamicro assembler requires that the operand field of a mnemonic be enclosed in parentheses. This limitation, as well as others, make it difficult to develop an assembler that achieves full syntax compatibility. However, it is relatively easy to use the metamicro to develop an assembler for a new machine and the development time varies from a few man days to produce a basic working version, to a few weeks if comprehensive error detection and full compatibility (e.g. using YACC) are required.

3.1.2 ISP´ COMPILER [STRA78]

Various features of the ISP´ hardware description language and its N.mPc compiler have been commented upon previously. It may be worthwhile at this point to review them and see how they impact the building of simulations.

1. ISP´ is very versatile and allows a designer to implement existing hardware structures and invent his/her own special purpose devices.

2. The use of ISP´ is relatively straightforward, albeit slightly disconcerting at first, due to its register transfer nature. The parameter passing mechanism also requires some care.

3. ISP´ offers facilities for a module to communicate with the outside through "ports" and to keep track of its execution through control variables stored in internal "states".

4. ISP´ provides for the asynchronous triggering of some pre-determined processes when certain events occur. The occurence of those events can be detected through ports connected to devices generating the events.

Hardware models can also be easily debugged using the trace option. Although very useful, this option should be used sparingly since it drastically affects the overall size of the simulation.

3.1.3 ECOLOGIST AND RUN TIME KERNEL [ORDY78],[ORDY79a]

The Ecologist together with the simulated memory manager are used to build a simulation, that is to produce a set of processes and data that, when loaded, will execute the simulation. The ecologist and simulated memory processor combination works well, although a few minor irritants should be mentioned:

1. The page option determines the size of the actual physical pages containing the programs in object code. Two of those pages are present within the simulation, with the rest being held in one of two simulated memory managers. The recommended page size is 256 bytes and when overall simulation size problems were encountered, varying the size of those pages did not produce anything significant.

2. As mentioned in item 1, there are two simulated memory managers. Each of them can hold up to 50K bytes of memory space and they communicate with the simulation over UNIX "pipes". The total size of the simulated memory is therefore 100K bytes. This total size is not program size but rather the size of the various memory structures supported by the simulation components. For example, if an Intel 8080 is simulated with a full house of memory, the corresponding memory module is 64K byte long. The program to be run by the 8080 may only be 20K byte long, but the whole 64K structure has to be available to the processor (i.e. 8080) should it decide to access it. In other words a simulation of two 8080´s with full memory totalling 128K bytes cannot be done.

3. The size of the memory structure of a given processor is defined in the linking/loader command file for that

particular processor. A common mistake is to define the memory as the total address space of the processor. This results in a simulation program that is too large to run.

The run time kernel oversees the simulation currently running. It has been found very flexible especially the way monitors (i.e. probes), breakpoints and other facilities are implemented. On the negative side, however, it happened that some simulations suffered abnormal termination. This situation was narrowed down to the case of simulations whose size was very close to the maximum permissible size. Since communications between simulation components is over UNIX pipe, it happened that not enough memory could be allocated to buffer space for pipes. It ensued that during a simulation, UNIX took over with the "Read Over Empty Pipe" diagnostic. UNIX then proceeded to do a core dump.

Another minor point is that the run time kernel assigns a monitor number to commands such as display, states, break point, etc. Those monitor numbers are not re-used and even though the upper limit is a large number, it could happen that the user would run out.

3.1.4 POST PROCESSOR

The run time kernel can be instructed to gather statistics on the simulation. In this fashion several variables (states, ports, simulated memory location) can be monitored and their history traced. The run time kernel puts all this information in trace files, in a format compatible with the post processor. The functions of the post processor are to read these files and to present the results in a readable form.

Unfortunately, the post processor and its utility cannot be commented upon since it is not working on UNIX V7 running on PDP-11 without separate I/D spaces. The N.mPc people are supposed to rectify the situation.

3.2 USEFULNESS OF N.mPc

Several advantages result from the use of N.mPc in system design and development. The most obvious one is the capability of designing complex hardware and software structures totally in software. In fact, with N.mPc, it is possible to design, implement, test and completely debug a system prior to hardware implementation. This is of great importance since it allows the designer to assess the correctness of a design through simulation.

Following the initial design and implementation phase, alternatives and/or refinements can be considered. Here again, N.mPc provides a suitable environment for implementing those alternatives and/or changes and also, N.mPc makes possible the comparing of different versions of a system in order to evaluate performance. Performance evaluation can be done according to two criteria:

1. It can be based upon probabilistic performance, or in other words, it can study bus loading, processor(s) utilization, etc., which are phenomena that are probabilistic in nature. To this end, N.mPc incorporates facilities to interpret trace data gathered at run time. These facilities take the form of a "Post-Processor".

2. The evaluation can also be based upon deterministic performance, that is correctnes. Correctness encompases many levels among which: correctness of hardware connections, correctness of robust/reliable schemes, correctness of software etc. It should be stressed, however, that in the present context, correctness does not imply any mathematical proofs but rather some form of correctness evaluation gained through testing.

The use of N.mPc in those stages of development brings extra benefits in terms of the software that is written to drive the simulations. The simulation software is either the real software that will be used in the target system or a skeletal software which roughly follows the same algorithmic steps as the real software. In the latter case, it may be possible to expand and modify the skeletal software so that it meets operational criteria (i.e. becomes application software). Therefore, when N.mPc serves as a development system, it allows various design and development activities to run concurrently.

Within the framework of a complete design methodology, the usefulness of N.mPc is limited by its inability to go beyond the processor/memory/bus level. If a design methodology using Ada in the higher level design stages is desired, interface mechanisms will have to be designed and put in place so that a link-up with N.mPc is achieved. The lack of extensive software capabilities in N.mPc hinders the design process. N.mPc does not provide any high level language programming facilities. The metamicro assembler supplied with N.mPc only allows programming in assembly language and does not support advanced features usually found in dedicated Macro Assemblers. Programming in assembler is fine for system dependent, time critical portion of the code. Otherwise, the use of a high level language is preferable. At present, the only solution to this problem is to use existing compilers or cross compilers for the desired target machine and load the object code produced by the compilers into simulated memory(ies). This would be an adequate solution although it involves obtaining a suitable

compiler for a given target processor and will also require some system programming work in order to effect the loading of simulated memories.

In conculsion, it can be said that N.mPc is not without shortcomings or limitations (see next section for description of shortcomings), but on the whole, the concept of computer assisted design as implemented by N.mPc gives a designer a tremendous help in the design of systems, be they for spacecraft applications or for general purpose systems. The two investigators on this contract felt that N.mPc was easy to use and at the same time very powerful in handling simulation testing and debugging.

3.3 LIMITATIONS OF N.mPc

N.mPc suffers from two important limitations. The first one is N.mPc's inability to support high level languages. The effects of this limitation were discussed in the preceding section.

Of a more serious nature is the second limitation which is concerned with the practical details of execution speed, maximum simulation size, etc. The version of N.mPc currently in use is running on Digital Equipment Corporation LSI 11/23. This machine being relatively slow, execution speed of certain simulation suffers greatly. Moving to a faster PDP-11 (or a VAX) would really help.

Besides execution speed, there is a problem of simulation size which is common to all PDP-11's. The total address space of a task on a PDP-11 cannot exceed 64K bytes. Complex systems such as N.mPc solve this problem by breaking up the components of the simulation into smaller chunks and by setting up interprocess communication through UNIX pipes. There is a limit to how much

decomposition can be done, however, since pipes introduce memory overhead and slow down execution appreciably.

The size of the simulation itself without run time environment is restricted to 64K bytes on PDP-11 without split I/D spaces. The size of the simulated processor's memory space is also limited to 100K bytes which is what the two simulated memory managers can hold. Going to a split I/D machine (e.g. PDP 11/70, 11/45) alleviates the problem somewhat, but real relief from memory constraints can only come from the use of a VAX. In fact, for any serious work such as simulation of a multiprocessor system for spacecraft, the use of a VAX-11 is highly recommended, especially in the later phases of development.

4.0 SUGGESTIONS FOR FUTURE WORK

Investigating computer assisted design tools is of great importance in view of the impact those methods have on the design of general purpose multi-processor systems and also of specialized hardware. N.mPc is at the present time the only such system in existence at the processor/memory/bus level. A new version of N.mPc is currently under preparation and is expected to be released soon. This new version runs on a VAX-11 and is free from the limitations of speed and memory size that plagued the PDP-11 version. The ecologist will be of a new design and the metamicro assembler/linking loader combination will be greatly enhanced. It would be advantageous to use the new version of N.mPc and it would also be important to have a working version of the "post-processor".

With the above tools, a complete simulation of a real satellite multiprocessor system could be done in detail. All input sensors, output activators and various processing systems could be simulated using adequate software. Performance evaluation could be done as well as testing of some recovery schemes for fail safe operations.

Irrespective of the choice of hardware/software development tool, interface constructs will be necessary in order to achieve proper link up with the Ada based high level design stages. Work in that area will be done under the present contract.

REFERENCES

[BART81] J.Barthmaier, "Intel Multibus Interfacing", ISBC Applications Handbook, Intel Corporation, September 1981.

[FERR81a] "F100-L Micro Processor Specification", Ferranti Computer Systems, September 1981.

[FERR81b] "F100-L Assembly Language Programming", Ferranti Computer Systems, September 1981.

[FERR81c] "F100-L Interface Set Specification", Ferranti Comuputer Systems, June 1981.

[LAFE82a] C. Laferriere, W.T. Brown, J.G. Ouimet, S.A. Mahmoud, "The Definition and Specification of an Integrated Set of CAE Tools for Spacecraft Multiprocessor System Design", Report No. INT-82-16, Intellitech Canada Limited, March 1982.

[LAFE82b] C. Laferriere and S.A. Mahmoud, "Trade-Off Study Report", Intellitech Canada Limited, August 1982.

[LAFE83] C. Laferriere and A. Lam, "Program Listings of Simulations Implemented Using N.mPc", Technical Report INT-83-47/2, Intellitech Canada Limited, January 1983.

[MAHM82a] S.A. Mahmoud, J.G. Ouimet, C. Laferriere, W.T. Brown, "A Survey of Computer Aided Engineering tools for the Design and Simulation of Multi Processor Systems", Technical Report No. INT-82-15, Intellitech Canada Limited, March 1982.

[MAHM82b] S.A. Mahmoud and C. Laferriere, "CAE Tools for Spacecraft Multi Processor Design: Progress Report", Intellitech Canada Limited, September 1982.

[ORDY78] G.M. Ordy, "N.mPc Ecologist User's Manual", Dep't of Computer Engineering, Case Western Reserve University, Cleveland, Ohio, Spring 1978.

[ORDY79a] G.M. Ordy, "N.mPc Runtime User's Manual", Dep't of Computer Engineering, Case Western Reserve University, Cleveland, Ohio, Spring 1979.

[ORDY79b] G.M. Ordy and F.I. Parke, "An evaluation of the N.mPc Design System", Proceedings of the 16th Design Automation Conference (IEEE), pp. 537-541, June 1979.

[ORDY80] G.M. Ordy, "N.mPc Release 2 Installation", Department of Computer Engineering, Case Western Reserve University, Cleveland, Ohio, November 1980.

[OUIM82] J.G. Ouimet, C. Laferriere, S.A. Mahmoud, T.F. Martin," Review of Multiprocessor Systems and Their Spacecraft Applications", Report No. INT-82-14, Intellitech Canada Limited, March 82.

[PARK79a] F.I. Parke, "An Introduction to the N.mPc Design Environment", Proceedings of the 16th Design Automation Conference (IEEE), pp. 513-519, June 1979.

[PARK79b] F.I. Parke et al., "The N.mPc Runtime Environment", Proceedings of the 16th Design Automation Conference, (IEEE), pp.529-536, June 1979.

[ROGE80] L.R. Rogers and G.M. Ordy, "N.mPc MetaMicro User's Manual, version 3.1", Department of Computer engineering, Case Western Reserve University, Cleveland, Ohio, July 1980.

[ROSE79] C.W. Rose et al., "The N.mPc System Description Facility", Proceedings of the 16th Design Automation conference (IEEE), pp. 520-528, June 1979.

[STRA78] R. Straubs, "N.mPc ISP User's Manual", Dep't of computer Engineering, Case Western Reserve University, Cleveland, Ohio, 1978.

intellitech
Intellitech Canada Ltd
352 MacLaren Street,
Ottawa, Ontario
K2P 0M6
(613)235-5126