intellitech

The Intelligent Use of Technology

(2)

VALIDATION OF N.mPc/N.2 MICROPROCESSOR SIMULATION

INT-84-38

(2)

VALIDATION OF N.mPc/N.2

MICROPROCESSOR SIMULATION

SEPTEMBER 1984

Prepared By: (1) Max Streit

Approved By: Dr. S.A. Mahmoud
Dr. C. Laferriere

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 - 84 - 041

DEPARTMENT OF COMMUNICATIONS - OTTAWA - CANADA

SPACE PROGRAM

TITLE: Validation of N.mPc/N.2 Microprocessor Simulation

AUTHOR(S): Max Streit
Intellitech Canada Ltd.
352 MacLaren St.
Ottawa, Ontario

ISSUED BY CONTRACTOR AS REPORT NO: INT-84-38

PREPARED BY: Max Streit

DEPARTMENT OF SUPPLY AND SERVICES CONTRACT NO: OER 83-05075

DOC SCIENTIFIC AUTHORITY: Michel Savoie
Communications Research Centre
Ottawa, Ontario

CLASSIFICATION: Unclassified

DATE: SEPTEMBER 1984

Summary

This document describes the validation of a hardware simulation facility and a microprocessor software development environment. The hardware simulation facility is provided by the CAE tool N.mPc whereas the software development environment is provided by a commercial cross software development package.

The validation scenario first involves the development of target microprocessor software in a high level language. For comparison the target software is then cross compiled and run on a simulation of the target hardware (itself running on the VAX 11-780), the actual target hardware and directly on the VAX 11-780.

The target software used is a simple spacecraft attitude control algorithm and the target hardware is an Intel 8086 based single board computer. The cross software development environment, the target software, and the target hardware simulation are described. This report is part of the work done under DSS contract OER 83-05075 for the Communications Research Centre of the Department of Communications, Government of Canada.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

1.1. Background

In recent years Computer Aided Engineering (CAE) tools have made it possible to choose a new, more flexible approach for the design of microcomputer systems. Traditionally, microcomputer based products are designed in the following fashion:

1. The necessary hardware components are built. This usually includes the microprocessor itself as well as other peripheral components.
2. Software programs are written for the target machine.
3. Software and hardware components are integrated and tested. Very frequently, the software is produced on a host machine using a cross development package, if available.

The development process usually involves many time consuming and costly iterations. A CAE tool such as N.mPc improves the situation by providing a simulation environment which is suitable for testing many design alternatives in a short period of time. The implications of using N.mPc are as follows:

1. It is no longer necessary to build the hardware components at the beginning of the design work. Instead, N.mPc provides what amounts to a micro-programmable, register transfer level machine which can be programmed to emulate the target hardware completely. In other words, a designer working on a VAX host, for example, could create a VAX executable program which, when run, would emulate the target hardware.

2. N.mPc provides a totally programmable cross development package for the software to be written in assembly language. The work documented in this report introduces an enhanced software development environment permitting to write software in the C high level language.

3. The rationale for using a tool such as N.mPc is that programmability implies flexibility. Given that a base exists, i.e. most of the hardware emulation is available as well as the cross development package, a designer can alter the design parameters with ease and test various alternatives without committing to any hardware choice.

The new approach to microcomputer design is of particular importance in a field like space technology where fault tolerant on-board processing is a very urgent need. Fault tolerance implies redundant multi-processor architectures of considerable complexity. Thus the development of a fault tolerant computer architecture for space purposes looks like an ideal application of the CAE Tool N.mPc.

In order to get a reliable, independent indication for the potential of CAE tools in multiprocessor design, the validation of the hardware simulation facility N.mPc and of an enhanced software development environment were included in the objectives of this work done for the Communications Research Centre of the Department of Communications.

1.2. Overview of N.mPc

It is assumed that the reader, having read the "N.mPc Detailed System Description" document [16], is already familiar with the elements of the N.mPc system. A short overview of N.mPc is given here as a

reminder.

N.mPc consists of six components used either to describe the hardware behaviour of a target system, or to execute the simulation of that system. Figure 1-1 illustrates the components of N.mPc and their interaction.

The Meta-micro assembler and the linking loader are used to generate the software which is to be executed by the simulated hardware components if these are programmable. Both are driven by a description of the instruction set of a target machine and can be made to generate code for either vertically or horizontally programmed machines. The linking loader produces code which is executed by a simulated processor or by an actual machine. The ISP´ compiler is used to produce simulation modules for individual processors and other hardware components of a system. The input language of the compiler is the ISP´ language which allows the specification of states for the implementation of processor registers and flags, memories for the simulation of memory, and ports which allow input to and output from simulated hardware.

The N.mPc ecologist and a simulated memory processor link the ISP´ processor modules with the linking loader outputs in order to form complete simulations. A run-time package is used to execute a simulation and to allow extensive interaction with the simulation.

1.3. Report Structure

Section 1 introduces the reader to background, motives and objectives of the work documented in this report. Section 2 outlines the approach chosen to validate an N.mPc microprocesor simulation. Section 3 describes the simulation of an Intel 86/12 SBC hardware on the N.mPc system. Section 4 covers the development of the test software run

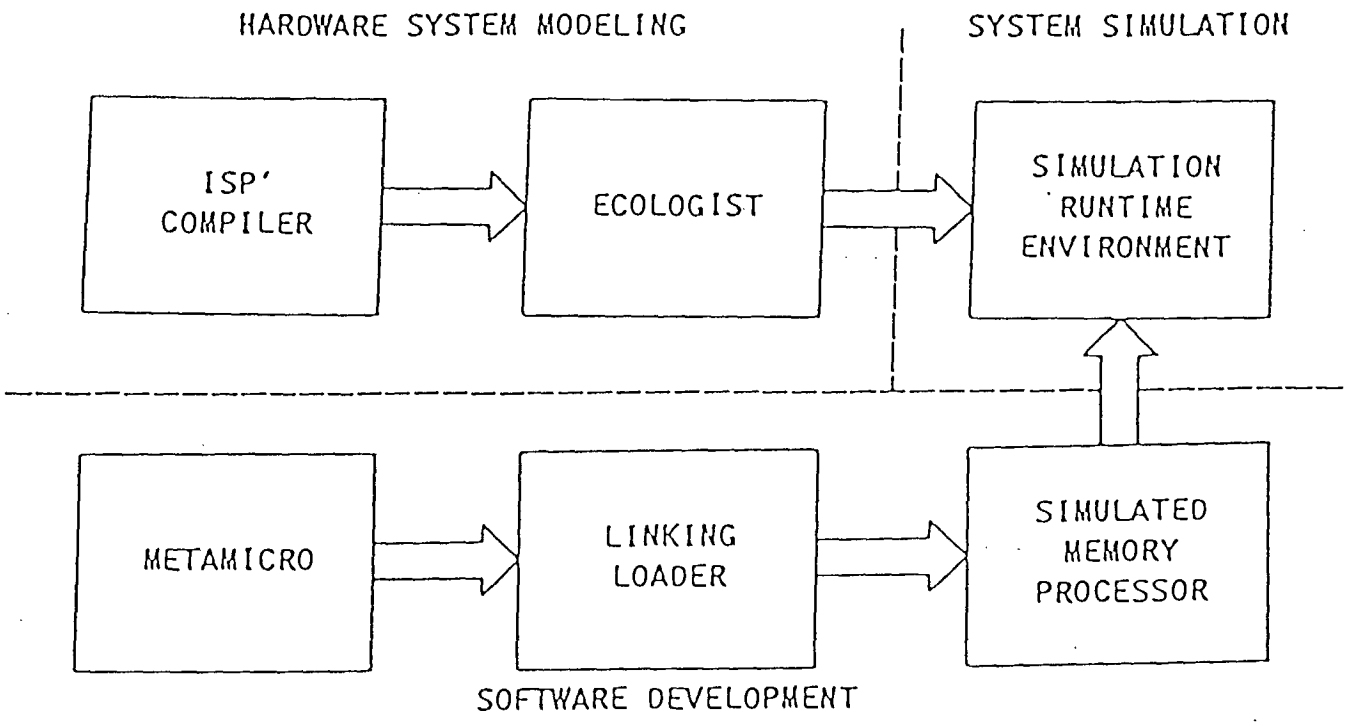


Figure 1-1 Elements of the N.mPc System

on the simulated and the actual hardware to perform the validation of N.mPc for microprocessor simulations. Section 5 gives some detailed information about how to run the validation test program on the actual and the simulated hardware. Section 6 presents the results from the validation test program and the performance test. Section 7 gives a summary of accomplishments and makes conclusions regarding the strong points and the perceived shortcomings of the CAE tool N.mPc.

1.4. Related Documentation

This report is one of the deliverables identified under contract OER 83-05075. Other reports submitted under the same contract include an N.mPc detailed system description document [16], an N.mPc User Manual [7], a report on the simulation of the SBP 9989 microprocessor [8], and two reports on a Fault Tolerant Computer Architecture and a Fault Tolerant Operating System [14], [20]. References to the original N.mPc documentation, provided by the vendor of the N.mPc package, are also listed.

2. VALIDATION SCENARIO

The primary goal of this work was to validate N.mPc as a CAE tool for microprocessor simulations. This includes the investigation of N.mPc's potential as a hardware and software development tool. The idea was not only to develop a simulation of a fairly complex hardware module but also to introduce the enhanced software development environment for 8086 based N.mPc simulations which is described in section 4.1. This would allow to do software development in a high level language and was considered to be quite an achievement in the N.mPc context. Besides the validation of N.mPc as a CAE tool for microprocessor simulations, the validation activities would have some useful by-products, namely the development of a reliable, powerful microcomputer simulation of the 86/12 SBC and the introduction of the enhanced software development environment mentioned above.

With the above goals in mind, the following activities were planned to validate N.mPc:

- The design of hardware modules simulating the Intel iSBC 86/12 single board computer.
- The use of an 8086 C cross software development package to develop software in C that could be run on the 86/12 simulation.
- The implementation of an algorithm, taken from a space attitude control system, on the simulated hardware using C as a high level programming language.
- Run a benchmark program on simulated and actual Intel SBC hardware in order to do a performance comparison.

The choice of simulating an 86/12 single board computer was made because its complexity represents a challenge in terms of N.mPc simulations and

because a description of the heart of the 86/12, the 8086 CPU, was contained in the N.mPc microprocessor description library. C was chosen for the use as a modern high level language as it allows for close interaction with the hardware.

Figure 2-1 shows the activities carried out for the validation of N.mPc as a CAE Tool for microprocessor simulation. The activities denoted as 1), 2) and 3) in Figure 2-1 have been described in sections 3, 4 and 5.

As a result of the above mentioned activities the simulation of the 86/12 SBC and the enhanced software development environment were used to build the "Validation Simulation". The goal was to execute a simple spacecraft attitude control algorithm, implemented in "C", on both the simulated 86/12 SBC hardware as well as on an actual Intel SBC machine. Results and performance from these tests were to be used to determine the validity of the CAE tool N.mPc for a specific simulation.

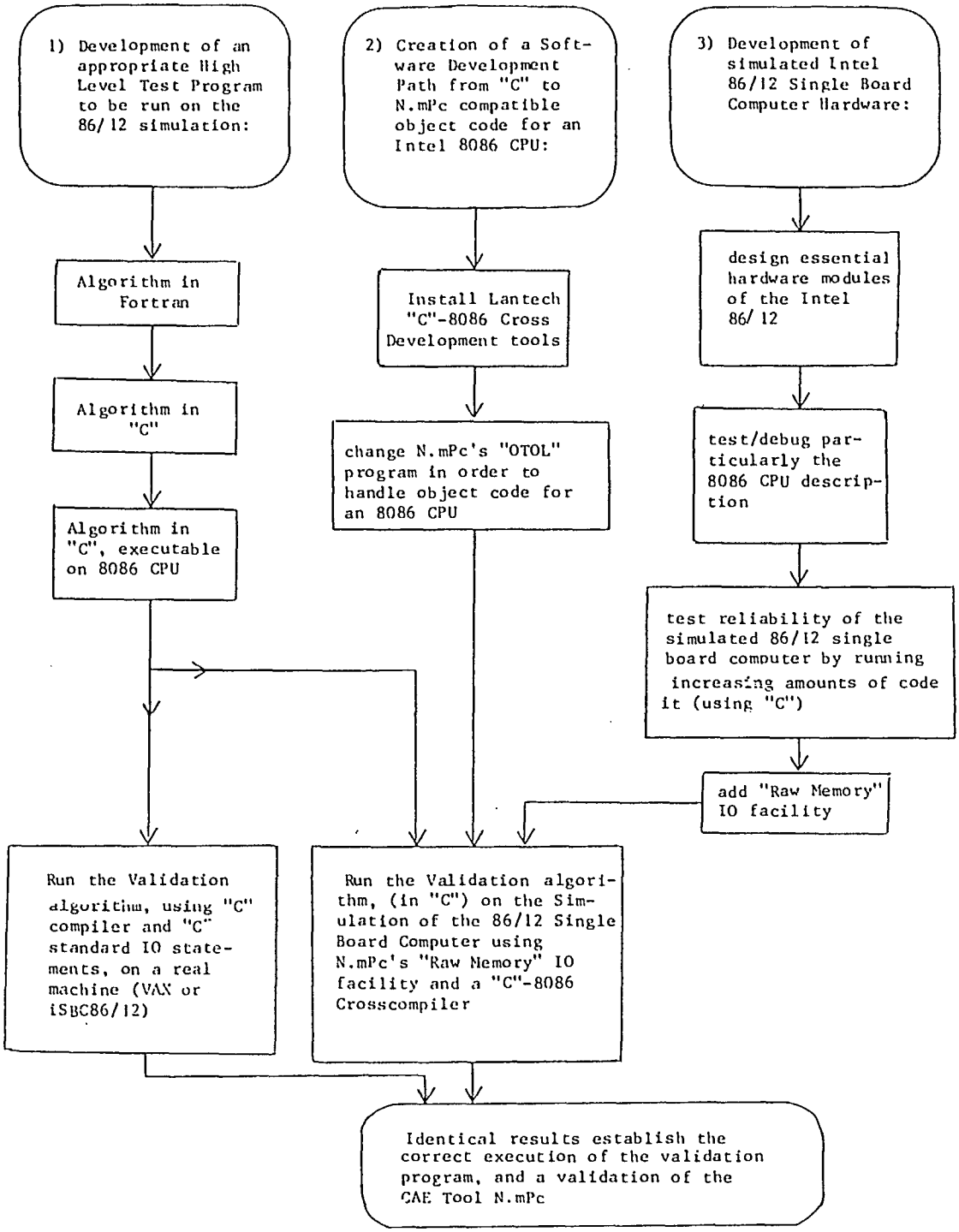


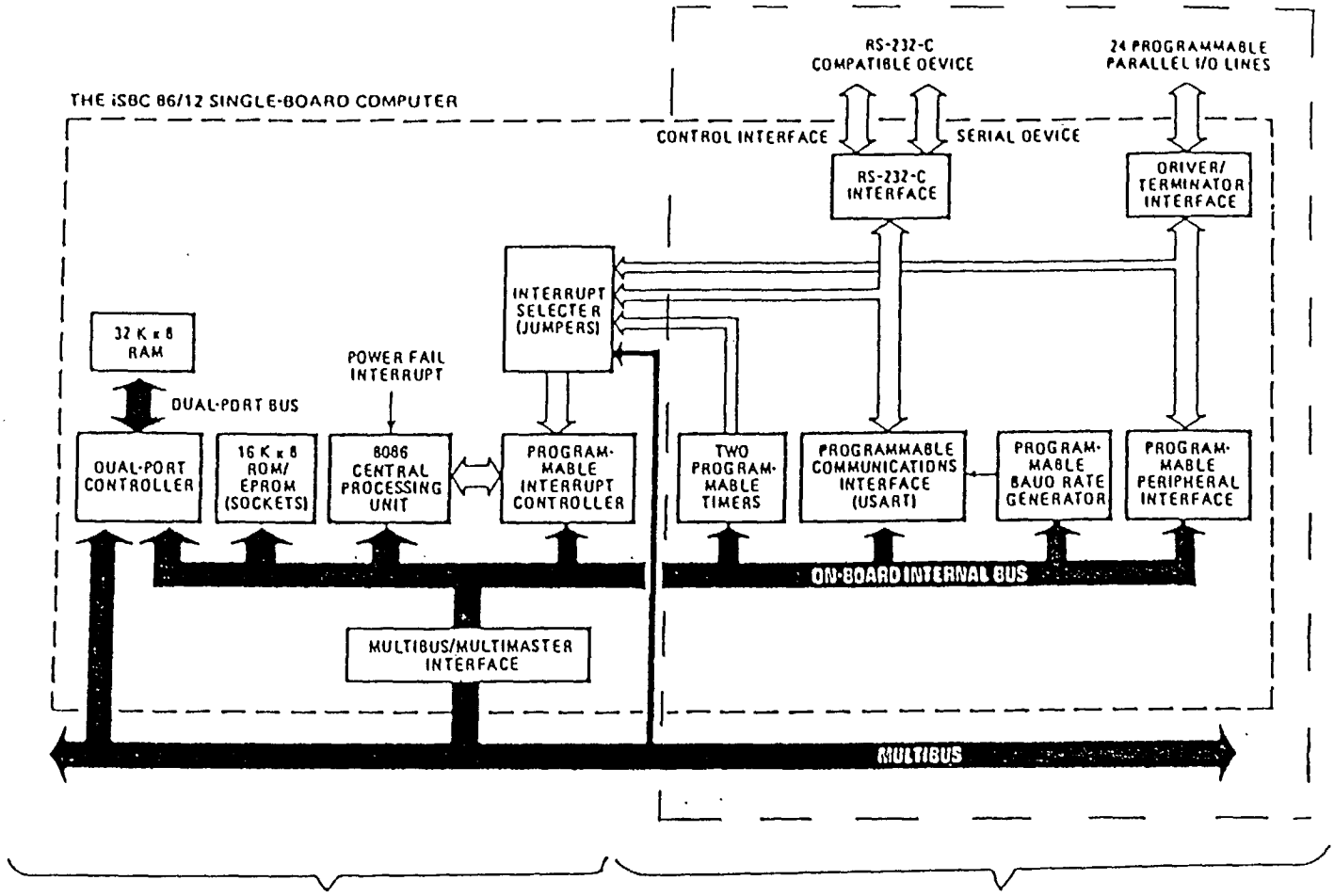
Figure 2-1: The Validation of N.mPc as a CAE Tool

3. DESCRIPTION OF THE INTEL 86/12 SBC SIMULATION

The Intel 86/12 is a single board computer that is based on the Intel 8086 CPU. The 86/12 SBC was chosen for simulation, mainly because a description of the 8086 CPU was available in the N.mPc microprocessor library (see [7], section 3). The 8086 also is an interesting processor for space related research as it is to be space qualified soon.

Figure 3-1 shows all the components of the 86/12 SBC. An 8086 CPU is the heart of the SBC structure. Other essential components are a programmable interrupt controller, memories (ROM, dual-port RAM) and the multibus interface. The Multibus allows the 86/12 SBC to communicate with other SBCs or with devices tied to the Multibus. Additional elements like the programmable communications interface, the baud rate generator, the peripheral interface, the RS-232-C interface and the driver/terminator interface provide various I/O functions. Programmable timers have also been included and may be used in certain applications.

To reduce the cost and the duration of the development of the simulation, only the necessary 86/12 components will be simulated. The essential subset of 86/12 elements to be simulated is shown in the left half of Figure 3-1. The complex I/O interfaces depicted in the right half of figure 3-1 are not needed for the intended simulation. N.mPc's "raw memory" feature will be used to do I/O operations. Figure 3-2 shows the structure of the simulated 86/12 SBC. The remainder of this section will describe each component of the 86/12 SBC simulation.



to be simulated using N.mPc

not needed for simulation purposes; N.mPc's "raw memory" feature used for I/O

Figure 3-1

The iSBC 86/12 Single-Board Computer (from Intel Application Handbook, Sept. 81)

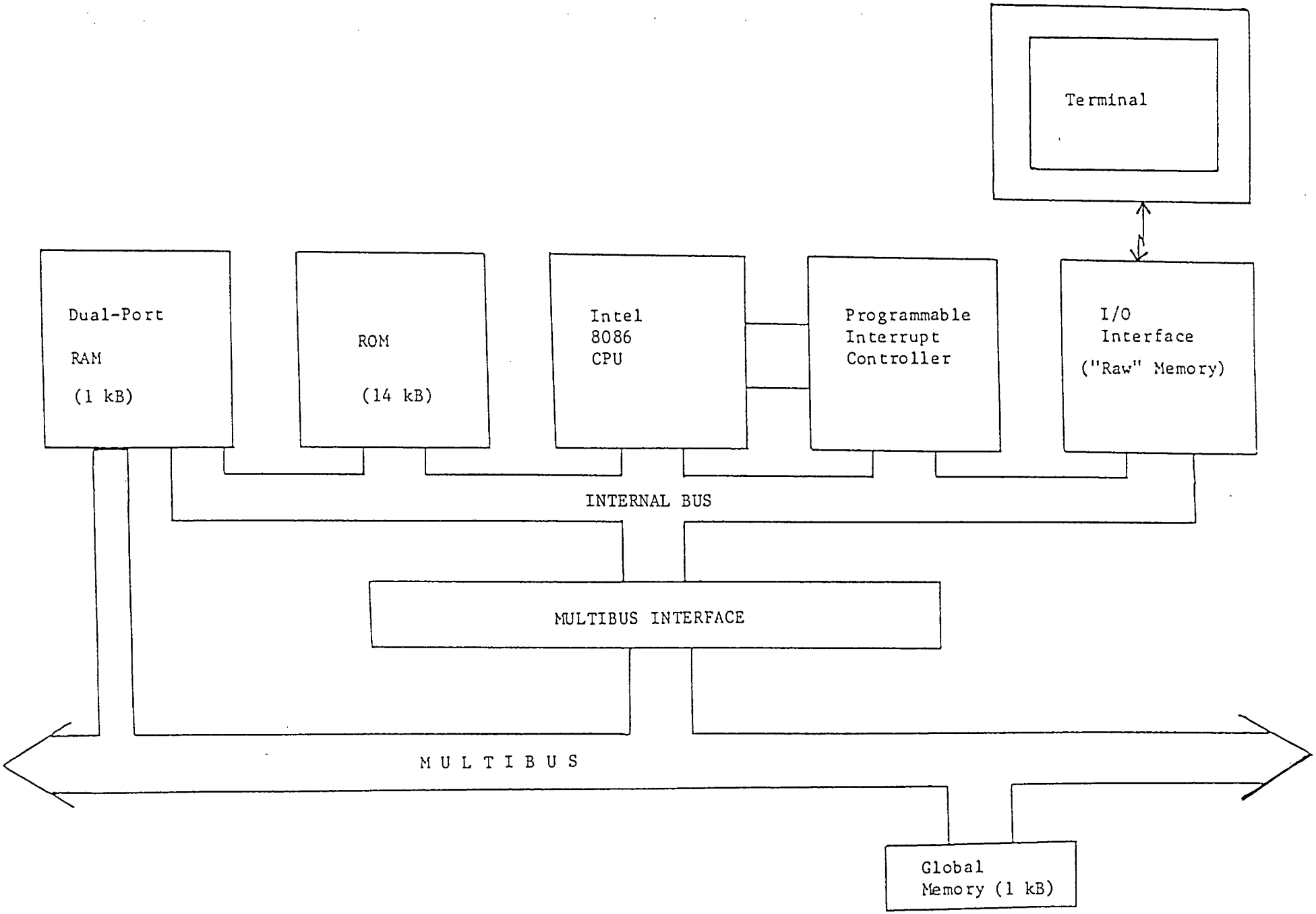


Figure 3-2: The Simulated 86/12 Hardware

3.1. The Intel 8086 CPU

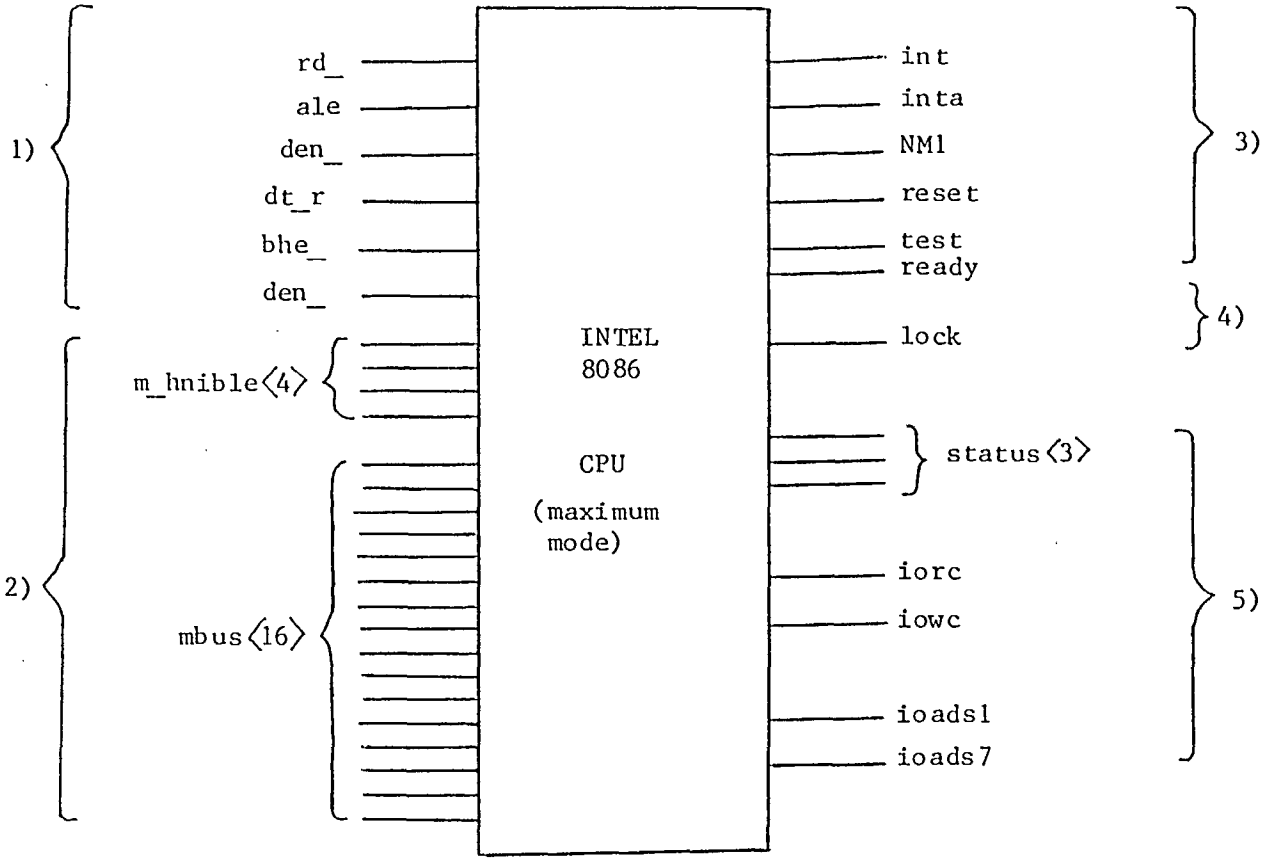
The starting point of the development of a hardware description for the Intel 8086 CPU was an ISP´ description of this microprocessor written by Y. Trivedi, Case Western University, for the N.mPc library. A few initial tests showed that this description required considerable testing/debugging in order to obtain the reliability needed for simulations of fault tolerant computer architectures. Therefore, the first and most important step was a thorough testing and debugging of the 8086 CPU description, a program of well over 3000 lines of code in size. The test program used for this purpose (see Appendix H) included every single 8086 instruction. The instructions were tested by executing them in various addressing modes.

Testing activities consisted of setting up the appropriate registers and/or memory locations for a particular instruction and checking these registers/memory locations after execution for correct results. If an instruction was not executed the way the Intel 8086 Hardware Manual prescribed, the error had to be found through a time consuming debugging effort. The usual errors found consisted in misuse of the ISP´ language or misunderstanding of how the 8086 hardware really worked. A list of the "bugs" found in the initial 8086 description is part of Appendix H. One type of error resulting from a misuse of the ISP´ hardware language deserves to be specially mentioned here. This error consists in wrongly determining the size of the operands of an instruction, for instance by testing a wrong bit. The result is a byte instruction trying to handle word operands (or the reverse), an ambiguity which cannot be properly handled by N.mPc and which causes the VMS operating system to halt operations. The fact that the simulation leaves the runtime mode altogether disables the usual debugging tools.

Only very thorough knowledge of the ISP' code in question, or iterative reduction of the user program until the fault-causing instruction is determined, can help. The N.2 system, which is to be released soon, should solve the above difficulty with VMS.

Figure 3-3 shows the communication ports used by the simulated 8086 CPU. The simulated 8086 is configured in "maximum mode" because the alternative "minimum mode" limits a CPU to standalone use. The "maximum mode" is therefore compulsory for multi-processor applications. As the goals of this work did not include the design of any co-processors (8087 Math Processor, 8089 IO Processor) for the 8086, the ports serving for this purpose were not implemented (request/grant lines, queue status lines). Some signals, which are usually generated on a separate bus controller chip because of lack of space on the 8086 chip, are part of the simulated 8086 (den_, dt_r, ale, inta).

Internally the simulated 8086 represents a very accurate model of the real 8086 chip. Like the "Bus Interface Unit" in the real 8086, a process called "BIU" constantly refills a queue of prefetched instructions which are to be executed by a second process called "EU (Execution Unit)". The "EU" process reacts to external events (interrupts, resets), fetches, decodes and executes instructions from the prefetch queue. At this point, it should be stressed that N.mPc hardware descriptions resemble black boxes behaving like real hardware when observed from the exterior but which are entirely different internally. Hardware descriptions designed using the N.mPc system can, and usually do, have an internal structure and complexity which is very similar to the actual hardware they model. In N.mPc this is true down to the register transfer level which strengthens N.mPc's potential in hardware design.



1) memory access signals

2) multiplexed address/data bus

3) control signals (interrupts etc.)

4) bus priority lock control

5) Multibus signals

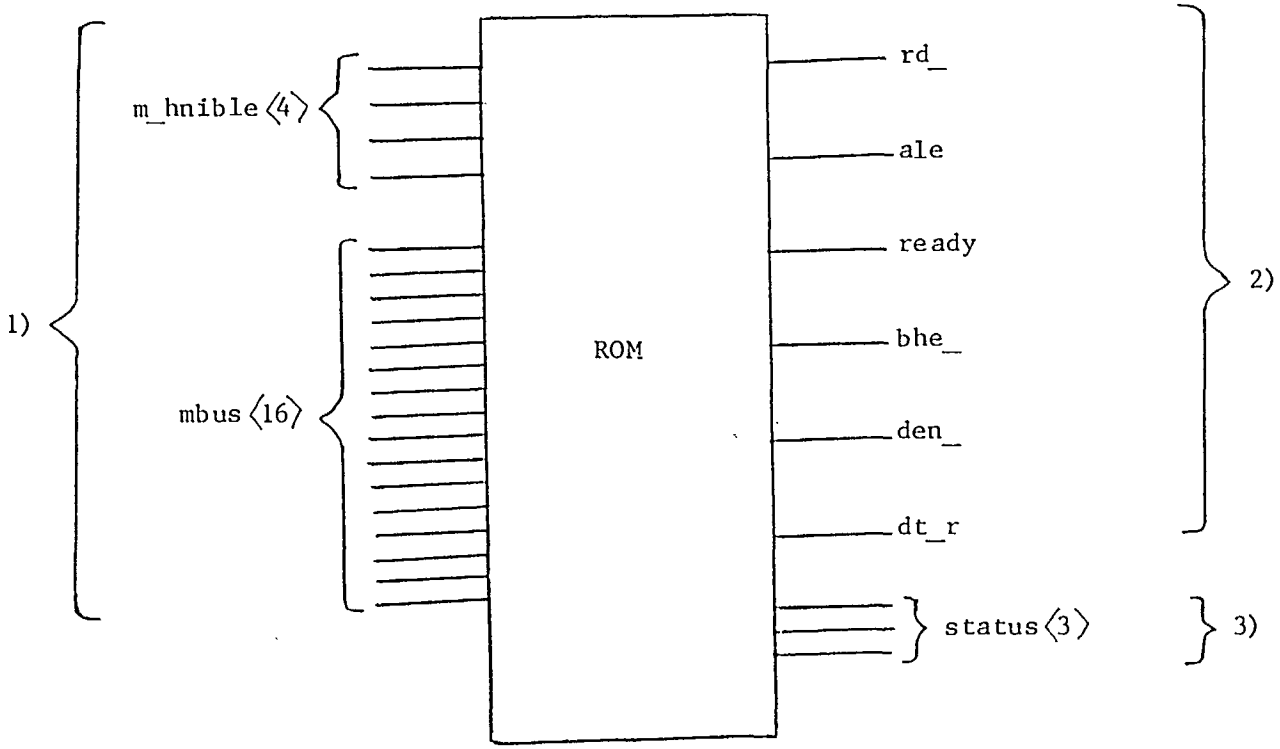
Figure 3-3: The Simulated 8086 CPU

Another aspect of the simulated 8086 is that the user has to initialize segment registers, stackpointer and instruction pointer himself. In real applications an operating system usually takes care of such details. As there is no operating system on top of this hardware simulation, the user is responsible for the proper initialization of the 8086 before running a program. Section 5 provides the details concerning the initialization of the simulated 8086 CPU.

3.2. The ROM

Three simulated memories are part of the 86/12 single board computer simulation (as shown by the memory map in Figure 3-11). This section will concentrate on the Read Only Memory (ROM). Figure 3-4 shows the address data lines and signals that allow an 8086 to access the ROM. It has to be added that for two reasons the "ROM" is not yet used in a read only mode in the context of the validation simulation (see *). Firstly, the N.2 system, which will soon fully supersede N.mPc, offers the possibility to declare ports that have attributes like "read only", "write only", "read/write" (see "N.2 ISP´ User´s Manual [15], page 13). Secondly, a ROM is not essential for the work described in this document as N.mPc can be made to perform a complete initialization of a RAM module at the beginning of the simulation. The fact that the RAM memory can still be accessed for write does not present any difficulty since such an operation can easily be disallowed.

(*): Due to the flexibility inherent to N.mPc based simulations, certain liberties were taken with the Read Only Memory (ROM) module. In fact the ROM was used as RAM by the compiler for its stack and dynamic data area. This does not detract from the purpose and goals of the validation.



1) multiplexed address/data bus

2) memory access control signals

3) CPU status

Figure 3-4: The Simulated ROM

The memory address range used by the ROM is determined by the numerical values assigned the macros "LOROM", "UPROM" in the ISP´ description of the ROM. The address ranges of the other simulated memories in the 86/12 simulation are determined in a similar manner and it is therefore very easy to change the whole memory configuration. Currently, the ROM services the address range from 1024 (1k) to 14335 (14k-1) as shown in the memory map of Figure 3-11.

The name by which N.mPc´s "Simulated Memory Processor (SMP)" identifies the ROM among the three simulated memories that are part of a simulation is "romcore.;". If a user wants his program to be loaded into the ROM as an initial content, he simply creates a new version of the "1.out" formatted object version of his program and renames it to "romcore.;". The SMP will then know that the destination of the program is the ROM. The name "romcore.;" is chosen by the user but has to be previously declared in the topology file.

The simulated ROM is accessed exactly the same way as the real 8086 accesses memory: Words starting at even addresses are accessed in one memory cycle, words located at odd addresses need two memory cycles to be read. The CPU indicates whether a memory access or an IO operation is to take place by a "status" signal (see Figure 3-4). A "ready" signal holds the CPU until a memory access is over.

3.3. The Multibus Interface

In Figure 3-2 one can see that the simulated 8086 uses an "Internal Bus" to access memory directly. In multiprocessor configurations a bus like Intel´s "Multibus" will be used for interprocessor communication and resource sharing. As the "Multibus" is part of the intended 86/12 single board computer simulation, an interface between the "Internal

Bus" and the "Multibus" is needed. Figures 3-5, 3-1, 3-2, and 3-10 illustrate the Multibus interface.

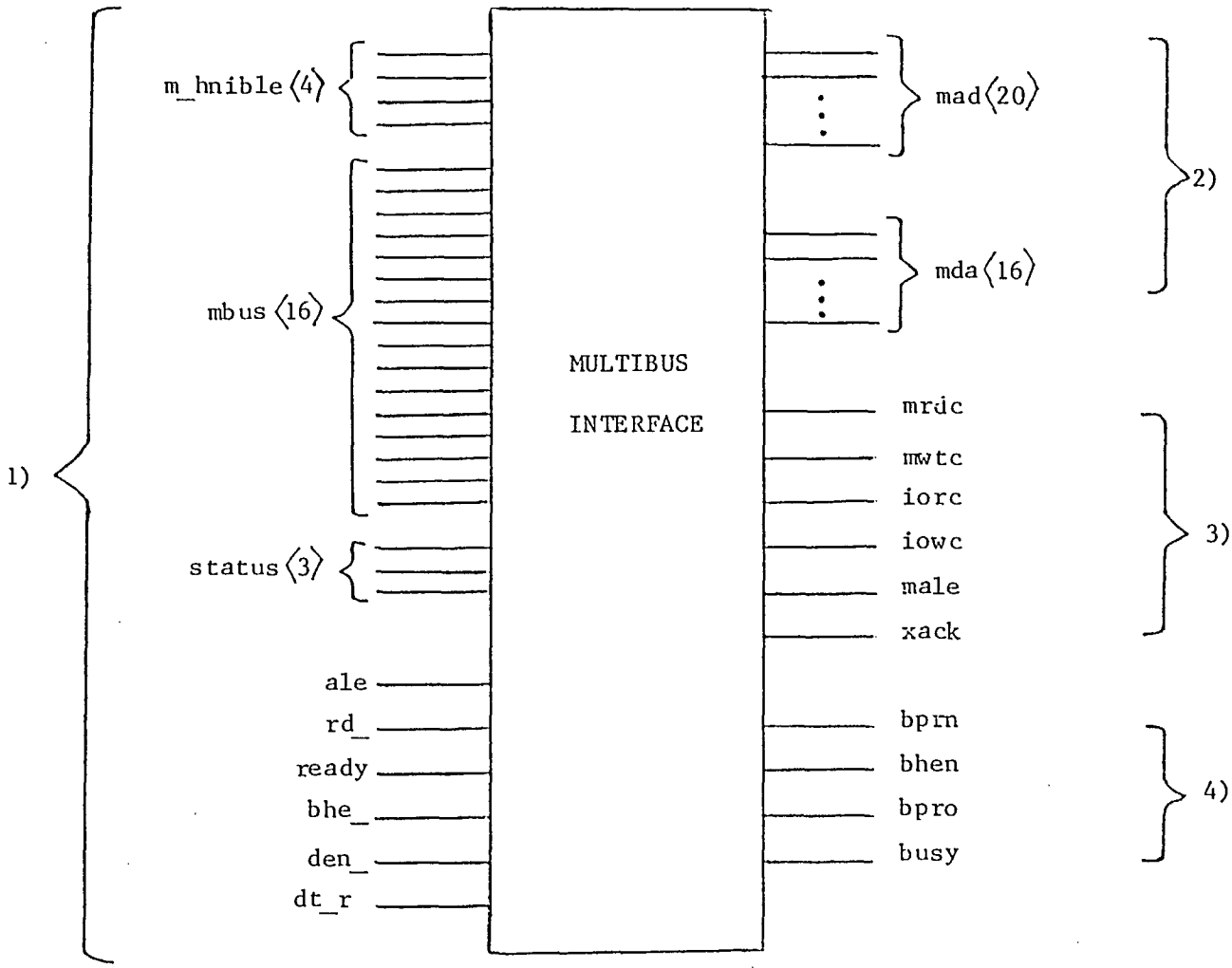
The Multibus interface will respond only to memory accesses in a range assigned to the Multibus. Like in the case of the ROM, two macros in the ISP´ description of the Multibus interface determine the address range. In the 86/12 simulation, the Multibus address range is set from 14336 (14k) to 16383 (16k-1) as shown in Figure 3-11.

In a multiprocessor configuration, several processors may want to use the Multibus at the same time. The Multibus interface not only translates the usual memory access of an 8086 (via internal bus) into a memory access following the Multibus protocol but also implements an arbitration scheme for the different Multibus masters. The method used relied on a serial priority scheme in which every processor has to wait for a bus grant signal to be activated. After using the Multibus, a processor passes the bus grant "token" on to the next one and so on.

A few Multibus lines were not relevant for simulation purposes and were therefore not implemented; among these lines were the Initialization signal, RAM/ROM Inhibit, Bus, Constant clock, Common Bus Request, Power Supply lines, etc.. It should be pointed out that the Multibus interface is transparent from the standpoint of a processor. The only noticeable sign of a Multibus use might be a longer wait when different Multibus masters are competing for the Multibus.

3.4. The Global Memory

The "Global Memory" is accessed exclusively via Multibus using the Multibus signals and protocols. A Multibus may have multiple masters thus making the global memory accessible to all processors tied to the Multibus. The global memory may be used for interprocessor



1) Internal Bus Side

2) Multibus Address Lines (20) and Data Lines (16)

3) Multibus Signals

4) Multibus Arbitration Lines

Figure 3-5: The Multibus Interface

communication (mailbox) or simply as a shared resource.

The address range of the global memory has to be within the address range assigned to the Multibus (see section 3.3). In the 86/12 simulation, the global memory is assigned the address range from 14336 (14k) to 15359 (15k-1) as shown in Figure 3-11. The signals, address lines and data lines used by the global memory are all part of the Multibus lines and are shown in Figure 3-6. The global memory is identified by N.mPc's "Simulated Memory Processor" by the name "gblcore.;".

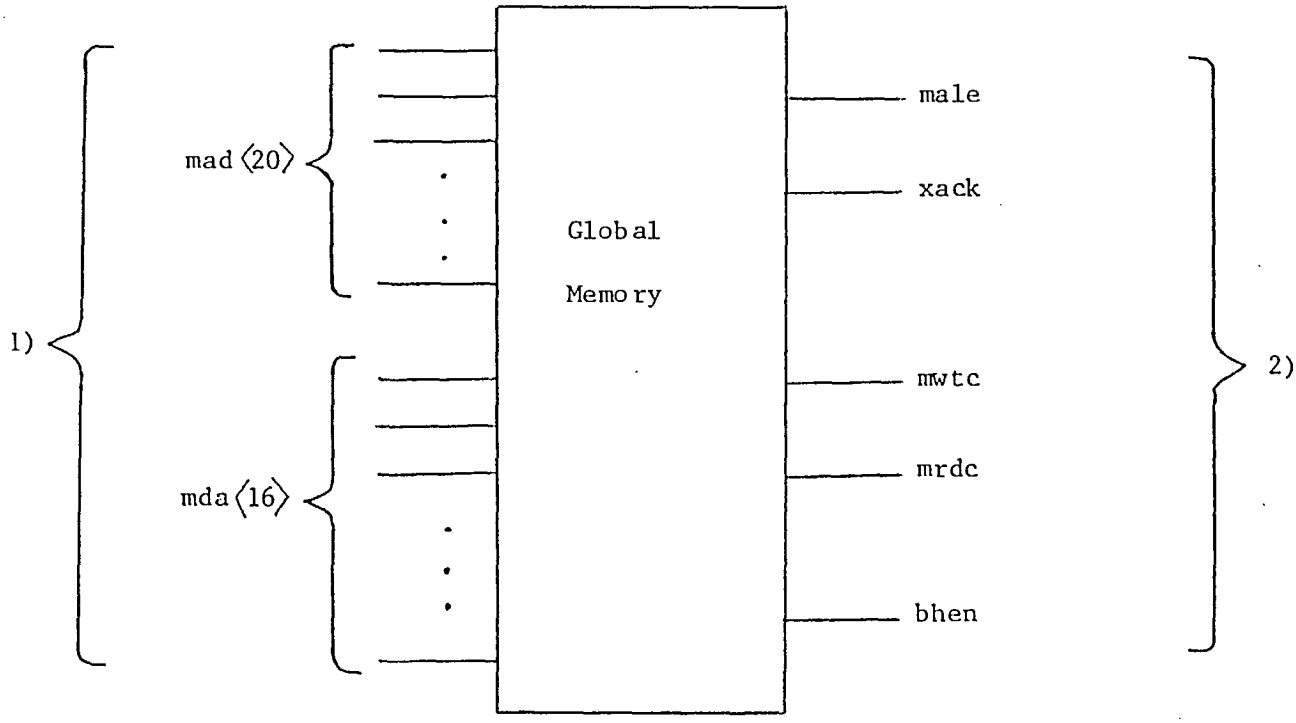
3.5. The Dualport RAM

The Dualport RAM is a combination of a RAM and a global memory allowing to access a given set of memory locations either via the Multibus or directly via the Internal bus. In the 86/12 simulation, the CPU accesses its RAM directly via the internal bus using the address range from 0 to 1023 (1k-1). On the other hand, the CPU (or any other Multibus master) can also access the same memory locations using the (Multibus) address range from 15360 (15k) to 16383 (16k-1). The Multibus addresses of the RAM are translated down into the 0 to 1023 range.

The Dualport RAM delays access attempts via Multibus while it is being accessed via the internal bus and vice versa. The internal bus has priority over the Multibus. Figure 3-7 shows the two buses connected to the Dualport RAM and Figure 3-11 shows the two different address ranges used to access the Dualport RAM.

3.6. The Programmable Interrupt Controller (PIC)

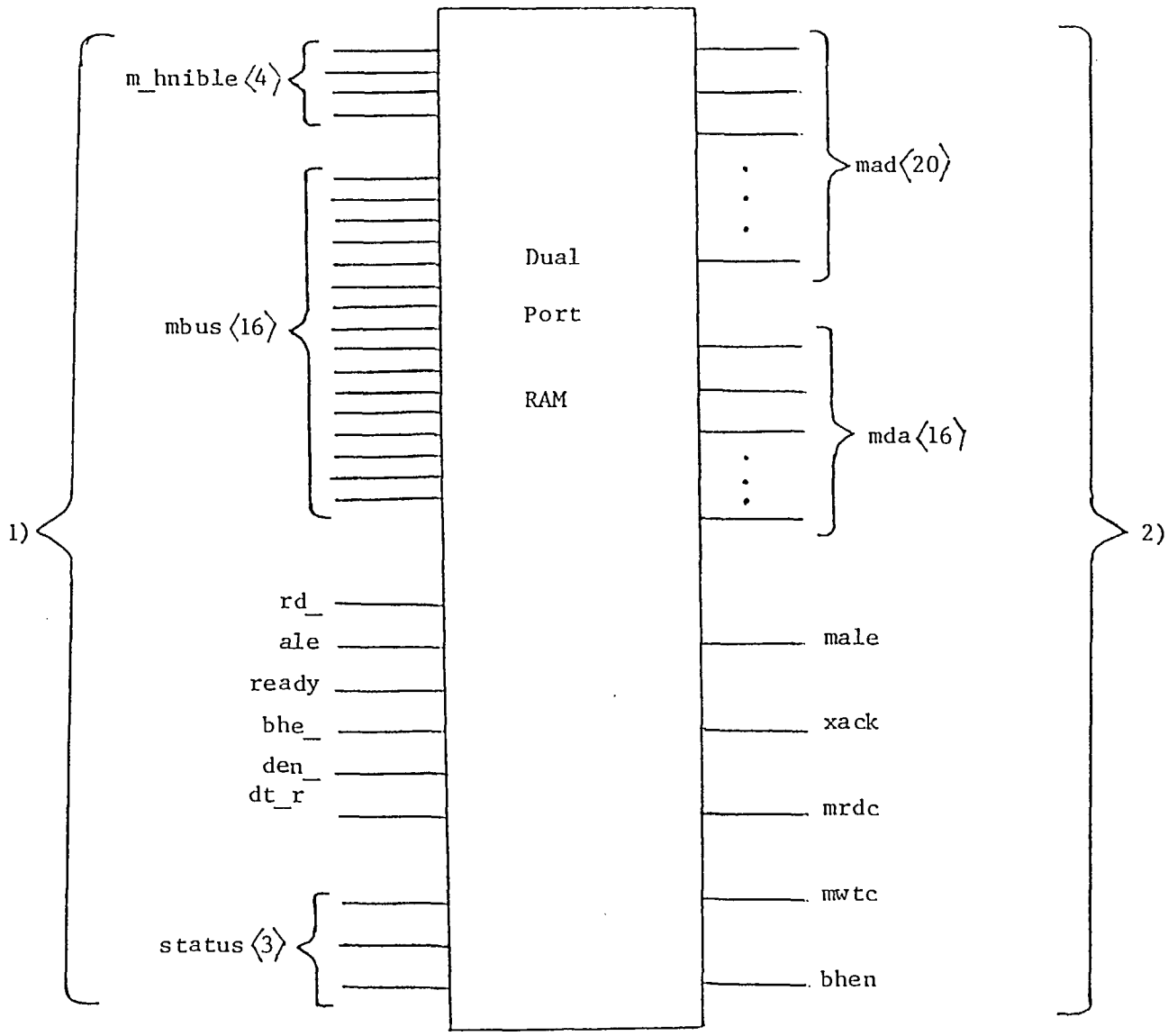
The PIC designed for the 86/12 simulation represents a subset of the functionality of the real Intel 8259 chip. The full set of



1) Multibus Address and Data Lines

2) Multibus Control Signals

Figure 3-6: The Global Memory



1) ports for access of RAM by CPU

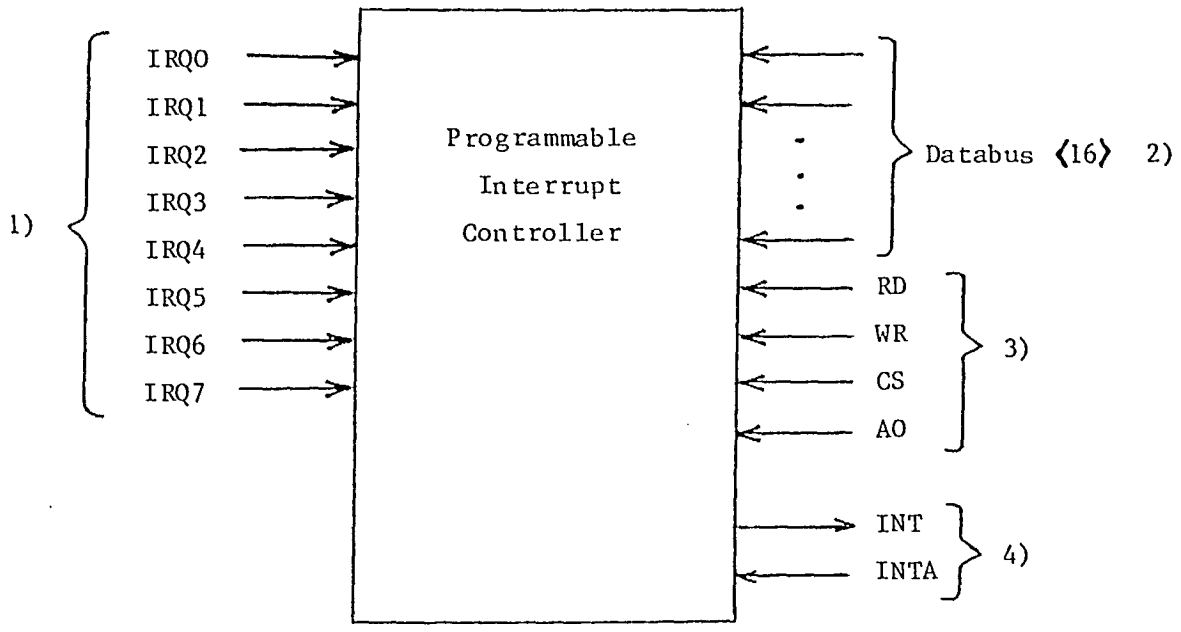
2) ports for access of RAM via Multibus

Figure 3-7: The Simulated Dualport RAM

different hardware functions and priority resolution schemes is not needed in a simulation context. The basic idea was to implement a fully functional PIC but only for the given 86/12 hardware and one single interrupt priority resolution scheme. The priority resolution scheme chosen scans the 8 interrupt request lines from line 0 to line 7 continuously. If an interrupt request line is scanned while being active the request is serviced, the request line is reset and the scanning restarts again at line 0. A request line is activated by the leading edge of a strobe on that line and stays activated until reset. It can be seen that the scanning is not cyclic and gives the lower numbered request lines a higher priority.

Figure 3-8 shows the simulated PIC and its ports. A "Signal Generator" was designed for convenient testing of the interrupt facilities in the 86/12 simulation. It periodically generates a "pulse" on each interrupt request line. This is an asynchronous way of triggering all the interrupts for testing purposes.

An interrupt is serviced in exactly the same manner as in the real 86/12 single board computer. The positive edge of a pulse on an interrupt request line sets the corresponding bit in the PIC's Interrupt Request Register (IRR). The interrupt may have to wait until it becomes the highest priority request when other interrupts have been activated as well. Once the interrupt has the highest priority, the PIC goes ahead and activates the CPU's "INT" line to signal the interrupt to the CPU. The CPU responds with a succession of two pulses on the "INTA" (interrupt acknowledge) signal, thereby reading the interrupt type which is put on the lower half of the data bus by the PIC. The 8086 then gets the interrupt vector (new instruction pointer, new CS register content) stored at the four memory locations starting at (4 * interrupt type).



1) Eight interrupt request inputs.

2) The lower half of the databus is used to send the interrupt type to the CPU.

3) Signals used for PIC programming via CPU.

4) Interrupt request lines.

Figure 3-8: The Programmable Interrupt Controller

The CPU then executes the interrupt service routine corresponding to the interrupt vector it acquired and, upon completion, returns to the execution of the interrupted program.

It is possible for the CPU to access the PIC's internal registers by doing IO operations using addresses assigned to the PIC by the designers of the 86/12. The following PIC register accesses may be carried out on the simulated PIC:

- Set Interrupt Mask Register:

```
MOV al, 0Fh
OUT 0C2h, al
```

  These instructions mask interrupt lines 0 to 3.

- Read Interrupt Mask Register:

```
IN al, 0C2h
```

- Read Interrupt Request Register:

```
MOV al, 0Ah
OUT 00C0h, al
IN al, 00C0h
```

- Read In-Service Register:

```
MOV al, 0Bh
OUT 00C0h, al
IN al, 00C0h
```

It is also possible to have the CPU initialize the simulated PIC but this is in fact irrelevant as the simulated PIC has been dedicated to a single priority resolution mode and to the 86/12 hardware.

3.7. The IO Facility

The IO module "terminal.isp", shown in Figure 3-9, makes it possible for N.mPc simulations to read and write directly from/to the working terminal. This "Raw Memory" has a structure almost identical to that of the ROM described in Section 3-4. The "Raw Memory", though, has

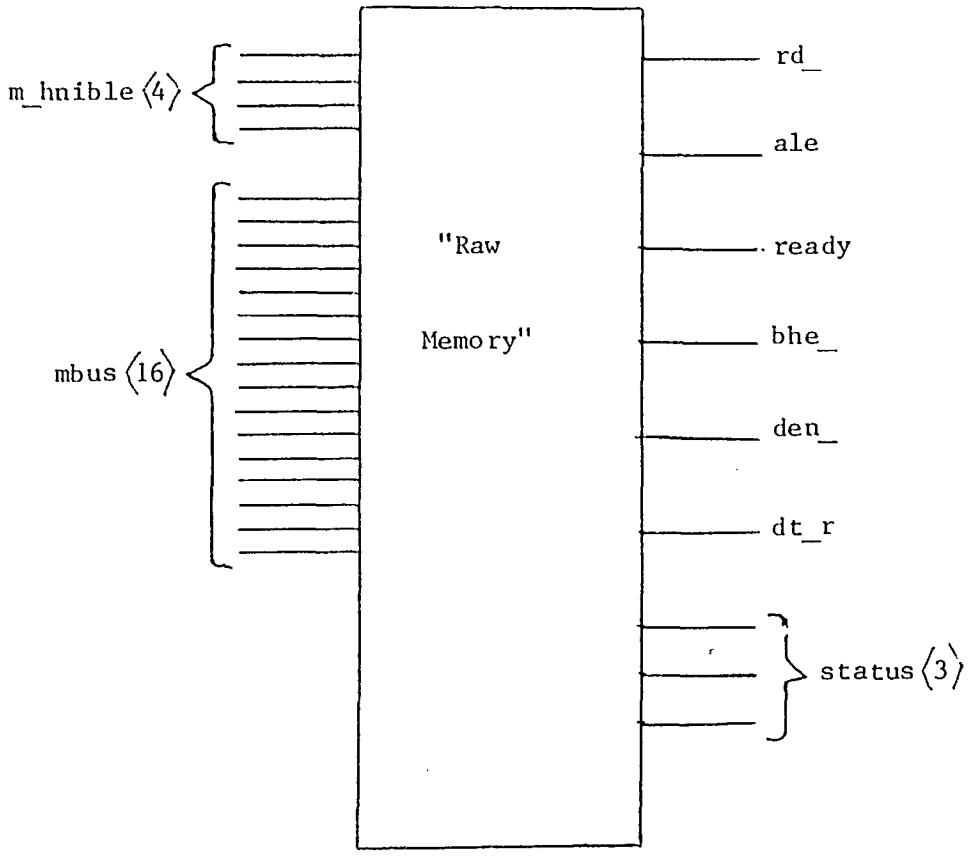


Figure 3-9 : The "Raw Memory" IO Facility

just one memory location and only responds to IO calls to address 0. In order to have the "Raw Memory" address the terminal driver, the name assigned to the working terminal under the VMS operating system has to be declared in the topology file. This name is "tt:" under VMS and is not compatible with the syntax used in the topology file. To resolve this situation a new logical name compatible with the topology file syntax has been defined ("wtty" for "tt:") in one of the login files (logsys.com). For details about "raw memories" refer to the "Ecologist User's Manual" ([4], Section 2.1.6). The following examples show how IO operations using the "terminal.isp" raw memory are carried out:

a) output to working terminal:
```
MOV al, #97
OUT al, #0
```

These instructions write the letter "a" (=97 in ASCII) on the screen of the working terminal.

b) input a character from the working terminal: IN al, #0

NB: The ASCII code for the first character on the working terminal's screen will be put into al.

The "Raw Memory" feature provides N.mPc simulations with a simple IO mechanism that is sufficient for the needs of most simulations.

3.8. The Simulated iSBC 86/12 Single Board Computer

The components of the simulated 86/12 SBC as well as the interconnection of their communication ports is shown in great detail by Figure 3-10. For a simulation, the information about a set of simulated hardware modules and their interconnection is stored in a topology file. Appendix F contains the listing of the topology file for the simulation of the 86/12 SBC. Detailed information about how to write topology files is found in the "Ecologist User's Manual" [4]. Figure 3-11 is a memory map of the three contiguous, simulated memories used in the 86/12

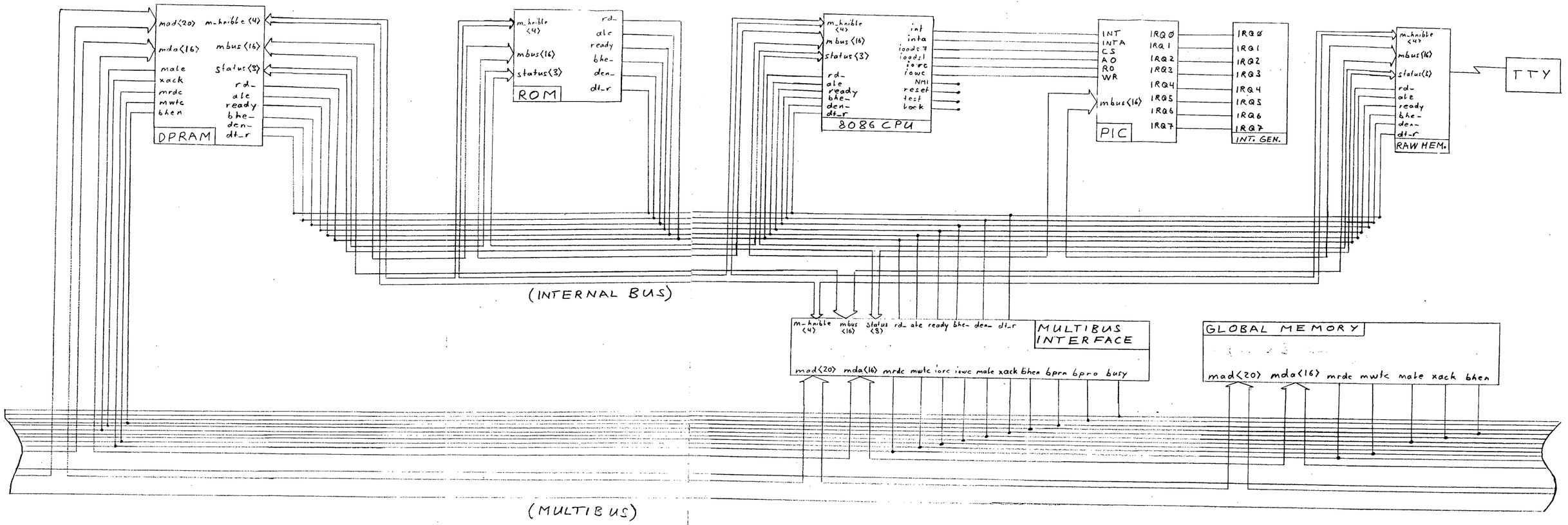


Figure 3-10: Schematic Diagram of the Simulated 86/12 Hardware

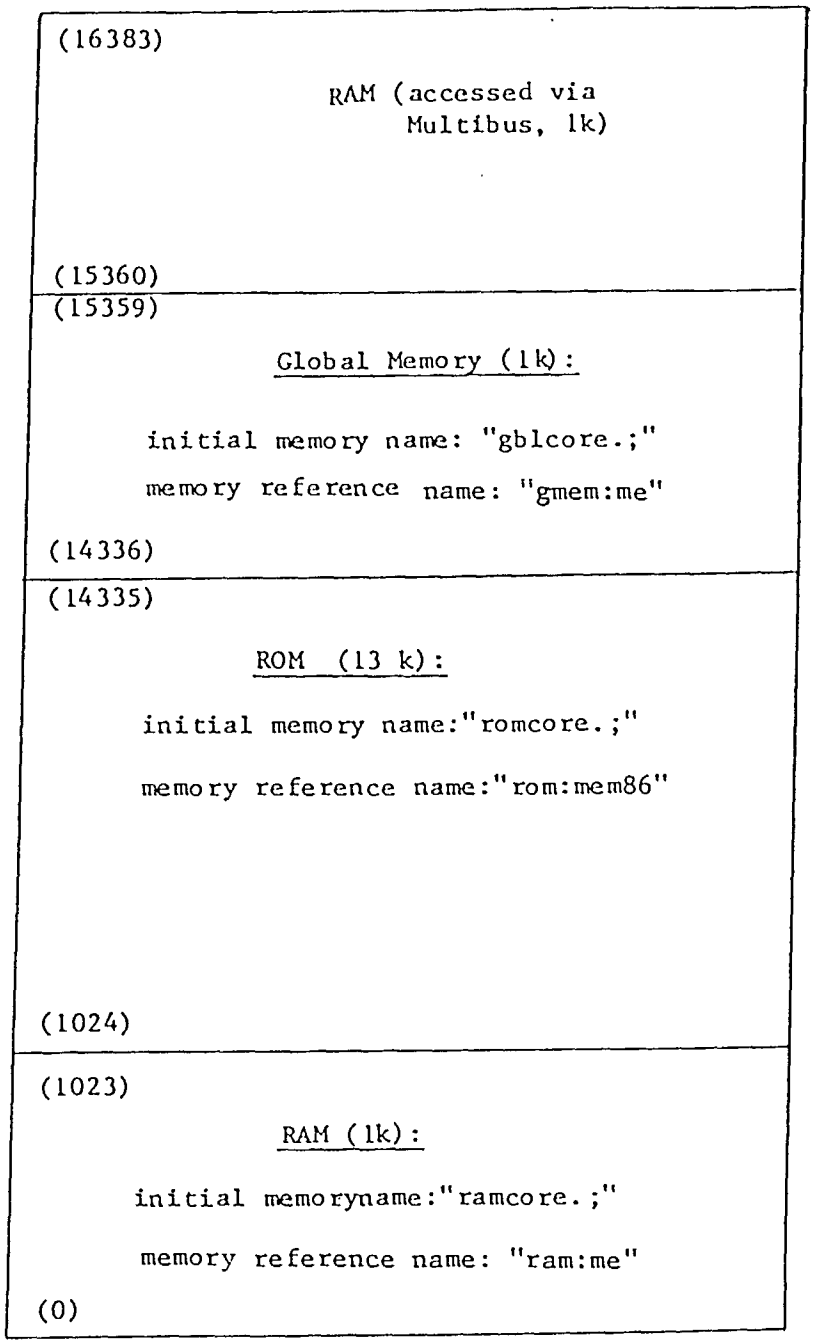


Figure 3-11 Memory Map of the Simulated 86/12 Hardware

simulation: RAM, ROM and Global Memory. For simulation of multiprocessor configurations one would simply have to tie multiple 86/12 modules onto the Multibus and set up the serial arbitration scheme for each processor's Multibus interface.

4. TEST SOFTWARE DEVELOPMENT FOR SIMULATED AND ACTUAL HARDWARE

This section describes the development of the validation test software for the simulated and the actual Intel SBC hardware as well as the software development tools involved and their use. All test simulations involving the simulated 86/12 SBC hardware are also described in the N.mPc User's Manual [7] (section 2) and in textfiles in the corresponding directories.

4.1. The Enhanced Software Development Environment for 8086 Based N.mPc Simulations

Initially, the only way to develop software to be run on N.mPc simulated hardware involved the use of a programmable "Metamicro" assembler that was not entirely compatible with commercial 8086 assemblers. The use of this "standard" method of software development for N.mPc simulations is documented in Intellitech's N.mPc User Manual [7] and, as a reminder, is illustrated again in Figure 4-1.

Following current trends, a need was felt for using high level programming languages for more convenient software development. The use of this method for developing software to be run on simulated hardware is also documented by examples in Intellitech's N.mPc User Manual [7]. An overview of a simulation in the case of the "C" language and a simulated Intel 86/12 single board computer is given by Figure 4-2. The rest of section 4.1 will describe the use of "C" and commercial cross-development tools in an N.mPc simulation context.

4.1.1. The "C"-Crosscompiler

As mentioned above "C" was chosen to be the high level language in which a user would write programs. The 8086 "C" Cross Software Tools by Lantech Systems Inc. are described in the corresponding Lantech Manual

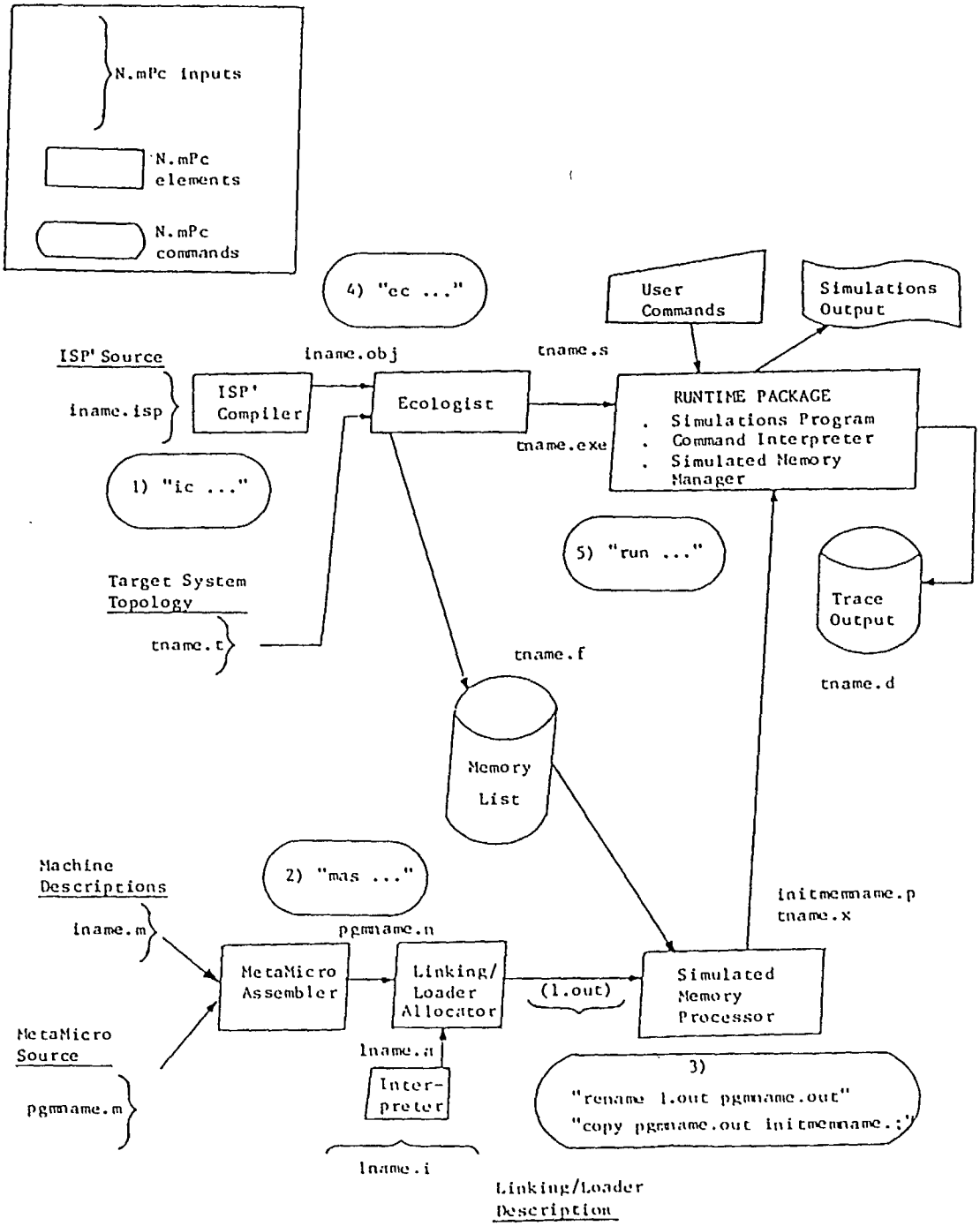


Figure 4-1 : Standard Software Development in N.mPc

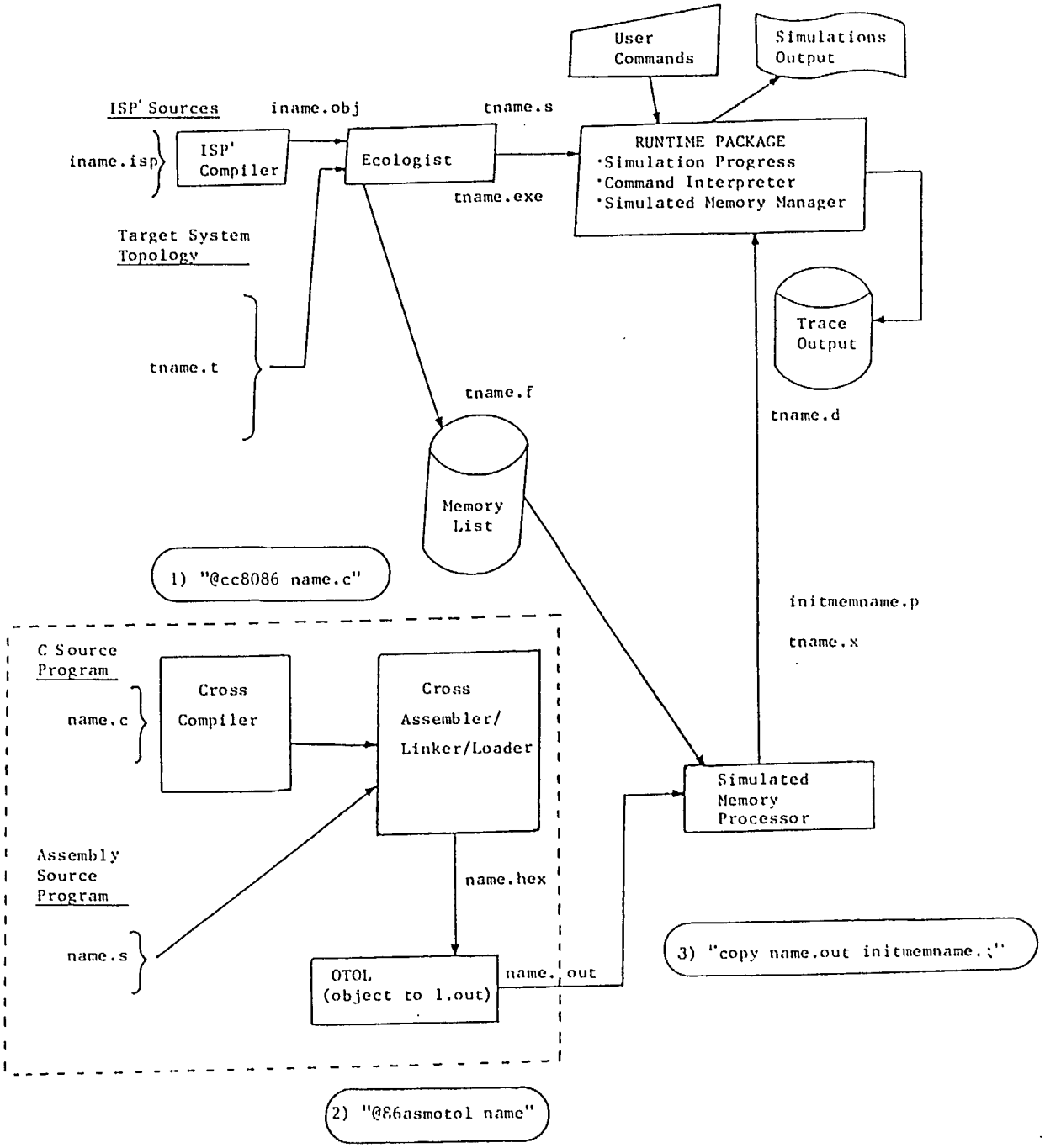


Figure 4-2: High-Level Software Development In N.mPc

[19]. The package includes a "C" Cross Compiler and a "C" Cross Assembler/Linker/Librarian/Downline Loader for Intel 8086/8087/8088 processors.

The steps necessary to run a program written in "C" on a simulated 86/12 hardware are shown in detail in Figure 4-3. The situation depicted in Figure 4-3 shows the path of a program written in "C" (name1.c) and a program written in Intel's ASM86 assembler (name2.s) through all the development stages. The two programs in the example are finally loaded into two different simulated memories of an N.mPc simulation by the "Simulated Memory Processor" (SMP). Figure 4-3 also shows how to do software development in an orderly manner. It is advisable to do the cross compiling of "C" programs in one directory ("c86") and the cross assembling/linking/loading plus the "otol" invocation in another ("as86"). The product of all software development (a ".out" file) is then copied into the actual simulation directory where it is loaded into some simulated memory. Doing software development in separate directories avoids having too many files in the simulation directory.

The Lantech Cross Development package was installed on the VAX/VMS environment which is host to the N.mPc system and is invoked using the commands declared in a login command file ("LOG86.COM", see User Manual [7], Appendix D).

Step 1 in Figure 4-2 shows how to invoke the cross compiler in order to create an 8086 assembly version of a C program. The operation involves several steps (preprocessor, parser, code generator, postprocessor) which are executed by a command file called "cc8086". This command file is located in the "C86" directory, which is reserved for high level software development. The command below is used to cross

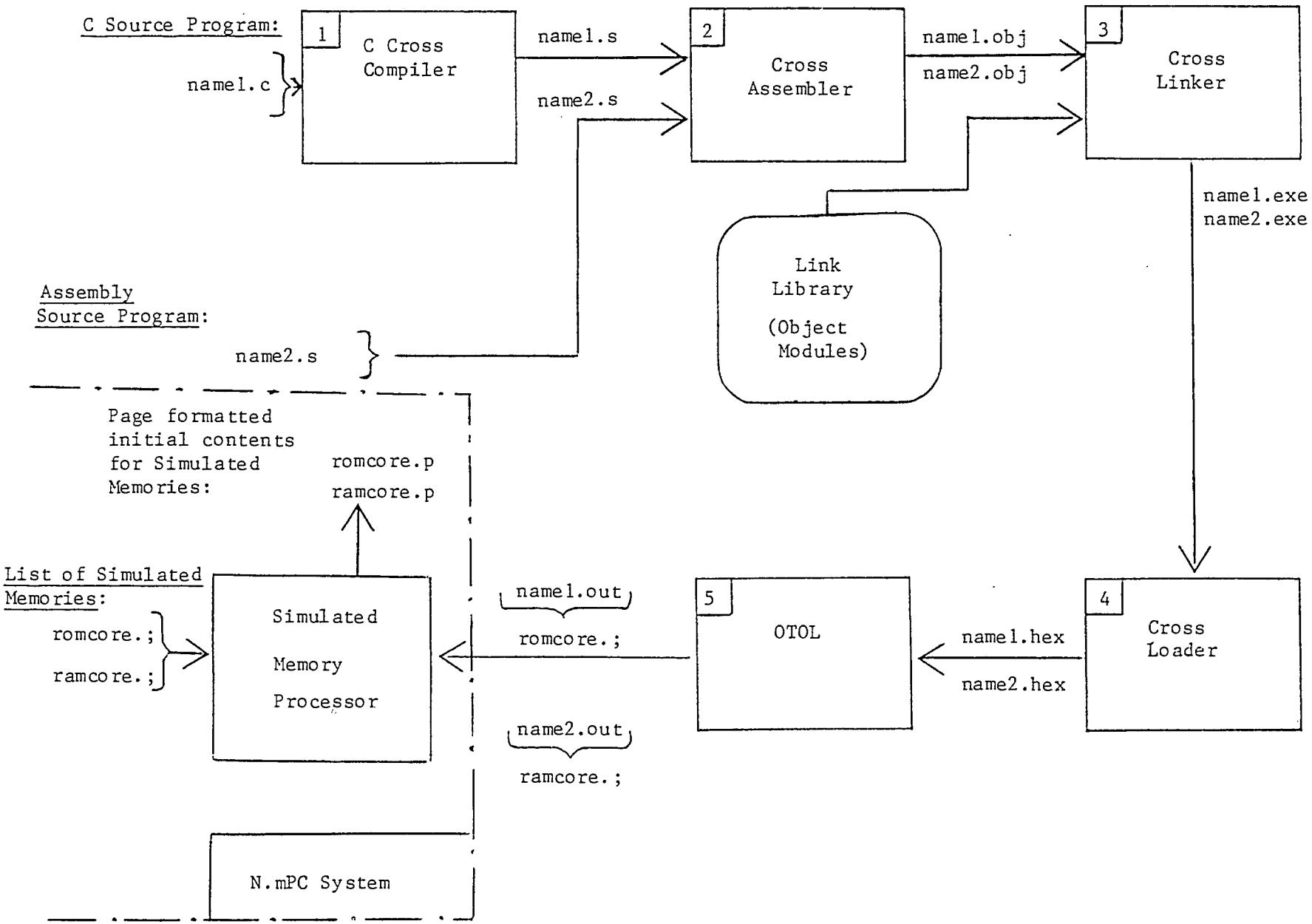


Figure 4-3 : Detailed Look at High-Level Software Development in N.mPc

compile the C program "name1.c" in Figure 4-3:

" @cc8086 name1"

The output produced is a file with a ".s" extension and the same name as the input file: "name1.s".

It should be remembered that the Lantech cross assembler/linker/loader assumes that both an 8086 CPU and an 8087 math processor are to be found in the target system. It therefore relies on the floating point arithmetic instructions implemented by the 8087, whenever floating point operations are required in a C program.

The assembly programs produced by the C cross compiler are stack oriented. If one wants to write assembly routines that can be called by C programs, detailed knowledge of the use of the stack by the code produced by the cross compiler is required. The Lantech documentation [19, chapter 7] contains detailed information on this subject.

4.1.2. The Cross Assembler/Linker/Loader

For details concerning the Lantech cross software development tools, the corresponding user guide [19] should be consulted. To illustrate briefly the application of the Lantech cross assembler/linker/loader, the steps 2 to 4 in Figure 4-3 will be explained using a cross compiled C program named "name1.c" as a general example.

Step 2 assembles assembly source programs written in a syntax which is very similar to the Intel ASM86 assembly language. Unfortunately, complete compatibility has not been achieved. The assembly source program may have been created either by the C cross compiler (Example: "name1.s") or directly by the user (Example: "name2.s"):

"asm86 -l -o name1.obj name1.s"

- The option -l generates a listing of the assembly program.
- The option -o is used to have the output named name1.obj

Step 3 in Figure 4-3 resolves references to library routines in the main program. For details on libraries refer to [19]. The cross linker links the main object module with other object modules stored in the link library to form a final object module:

"lkr86 -l -o name1.exe name1.obj llib:lib8086.a"

- The option -l is used to list tables of external references.
- The option -o is used to have the output file named to "name1.exe".
- "name1.obj" is the object module of the main program.
- "llib" is the logical name for the library (directory) in which source and object modules of some assembly routines are stored; at present the library only contains the routines necessary to process doubleword operands (corresponds to "long integer" variables in C).

Step 4 in Figure 4-3 determines where in a (simulated) memory the linked object module is to be loaded. The loader reserves two areas in memory: A "Code Block" and a "Data Block". It is important to know that the "Data Block" is reserved for data (global labels, ...) and the stack. The following command invokes the Lantech cross loader:

"ldr86 -l [-c ...] [-d...] -o name1.hex name1.exe"

- The option -l generates a listing displaying information about starting addresses and sizes of reserved memory areas. This listing is important as it contains the necessary information for the initialization of the 8086 CPU.
- The -c and -d options allow the user to choose the starting addresses (=hex numbers after -c, -d) of the memory areas to be reserved for program code (-c) and data, stack (-d). If these options aren't used the loader by default loads the program code starting at memory location 1024 (400 hex) and starts the data block at the next paragraph location after the end of the code. A paragraph is an address that is a multiple of 16.

4.1.3. The "OTOL" Program

The "OTOL" ("object to l.out") program transfers object code from different formats (Motorola, Intel Hex, ...) into the "l.out" format used by N.mPc. For this application the "OTOL" program had to be enhanced as it could not handle the concept of "segmented" memory (see Intel 8086 documentation). The 8086 has four segment registers and addresses its megabyte of address space in 64k byte segments.

With reference to Step 5 of Figure 4-3, the "l.out" formatted files produced by "OTOL" have to be renamed so that the "Simulated Memory Processor" can direct their contents to the appropriate simulated memory. In the example of Figure 4-3, the object version of the "namel.c" program is to be made the initial content of the simulated ROM. Therefore "namel.out", the object version of "namel.c", is renamed "romcore.;" (the name "romcore.;" is declared in the topology file) and

similarly "name2.s" is directed to the simulated RAM by renaming "name2.out" to "ramcore.;".

Step 5 in Figure 4-3 is executed by the following command:

"otol - id name1.hex name1.out [-a "initrecord"]"

- The i option tells "otol" that the object file to be transferred into "1.out" format is in Intel Hex format.
- The -d option produces a listing of starting addresses of data records.
- "name1.out" is the name assigned to the output file in "1.out" format. The ".out" extension reflects the fact that the file is in the "1.out" format required by N.mPc. The default name of the output file is "1.out".
- The "initrecord" option may be used to initialize areas of memory with certain values.

    Example: "-a2048-3000$00" initializes the memory locations from 2048 to 3000 with the value 0.

Other object code formats which can be handled by "OTOL" include:

- MOTOROLA;
- Tektronix Hexadecimal;
- RCA Cosmac;
- MOS Technology;
- Signetics Absolute Object;
- Fairchild Fairbug.

A short documentation on "OTOL", written by the N.mPc developers is to be found in Appendix G.

4.1.4. Command Files For The Cross Software Tools

As it is rarely necessary to invoke the cross assembler/linker/loader in a step-by-step fashion, several command files have been created to make the use of the Lantech cross software development tools easier. The command files are located in the "AS86" directory, which is reserved for software development. The name and purpose of each of the two command files currently used with the cross assembler/linker/loader are listed below along with an example for their use:

1) "86asmotol.com": This command file invokes the cross assembler/linker/loader and the "OTOL" utility. It uses none of the -c, -d loader options so that the code is loaded starting at address 1024 and the data immediately after the code. Again, the input file is assumed to have a ".s" extension. The output file is given a ".out" extension as it is a file in N.mPc's "l.out" a format.

   Example: "@86asmotol name1"

2) "c0asmotol": This is exactly the same command file as the "86asmotol" but the -c option is used when invoking the loader in order to have the code loaded starting at address 0. (instead of the 1024 default).

   Example: "@c0asmotol name1"

NB: If the reader at this point should want to add further to his knowledge of the use of the Lantech 8086 C cross software development tools, the reader is referred to the examples documented in section 6 of the N.mPc User Manual [7].

4.2. Software Development for an actual Intel SBC

The actual hardware used was an Intel 86/30 SBC running iRMX86. The 86/30 SBC is essentially an 86/12 with additional on-board memory and for the purpose of this test can be considered identical to an 86/12 SBC. The test programs for the actual hardware, written in C, were compiled using Intel's C-86 compiler. Appendix C shows the steps to generate the code necessary to execute a test program on the actual Intel SBC hardware.

4.3. Description of the Target Software for the Validation

The previous two sections described the software development tools involved when developing test software for the actual and the simulated Intel SBC hardware. The test software was to be run on both the actual and the simulated Intel SBC hardware in order to validate N.mPc's microprocessor simulation capabilities. The starting point was the choice of a "Simple Attitude Control Algorithm" (SACL) for implementation as a test program. Appendix B contains a mathematical description of the SACL algorithm and listings of the programs implementing it. The SACL algorithm was discretized using the Z-transform and implemented as a Fortran program named "SACLZ.FOR". At this point, the fact that SACLZ.FOR used real (floating point) variables had to be taken into account. Floating point variables result in 8087 floating point arithmetic instructions as the C cross compiler assumes the presence of an 8087 math processor. The simulated 86/12 single board computer was not intended to include such mathematical convenience and the alternative was to use "integer" instead of the floating point variables in the SACLZ.FOR program. This resulted in an assembler

program with instructions that could be executed by an 8086 alone. By scaling some variables, an implementation of the SACL algorithm using only integer variables was produced and given the name "SACLZI.FOR". In order to produce a SACL version in C that could be developed on the Lantech cross software development package used with the simulated SBC hardware, SACLZI.FOR was translated into C and called "CMD.C". The fact that the Lantech cross software development package did not include an "ABS" (absolute value) routine in the link library was taken into account by adding an "ABS" subroutine to the "CMD.C" program. In this form, CMD.C can be run on the actual Intel SBC hardware.

It was stated earlier that N.mPc's raw memory feature was to be used in order to simplify I/O operations for simulations. Two I/O procedures ("IN", "PRINT") have been written in assembly language and added to the link library of the Lantech cross development tools in order to replace the standard I/O procedures ("SCANF", "PRINTF") called in CMD.C. The program using the new "IN" and "PRINT" procedures for I/O in N.mPc simulations is called "VALCMD.C" and can be run on the simulated 86/12 hardware using the Lantech cross software development tools. As VALCMD.C uses "long integer" variables, the 8086 CPU has to handle double word operands. The necessary assembly language routines are contained in the link library which is part of the Lantech Cross software tools. Listings of the two assembly routines used to perform the I/O for the validation simulation are found in Appendix I. The total size of the validation code amounts to 1680 bytes which is very considerable in an N.mPc simulation context. This code corresponds to about 300 8086 instructions or a C program of about 40 lines.

The steps to build the simulation of an Intel 86/12 SBC and to run the validation testprogram ("VALCMD.C") on the simulated hardware are shown in detail in Appendix D. How to run the intermediate programs ("SACLZ.FOR", "SACLZI.FOR") produced when developing the validation testprogram ("VALCMD.C") is also indicated in Appendix D.

5. VALIDATION/TEST PROCEDURES

In the case of the actual Intel SBC hardware it was very easy to run the validation testprogram as the 86/30 SBC is running the iRMX86 operating system. Appendix C shows that the actual running of the validation testprogram CMD.C (here named "CMVALID.C86") is done by a single command.

In the case of the simulated 86/12 SBC hardware things are not so easy. The user has to initialize some registers of the 8086 CPU in order to have the 8086 start executing at the beginning of a program and to set up a stack and data area in memory. The use of the Lantech cross software development tools introduces a restriction for the initialization of the 8086 which will be discussed in detail.

The Lantech 8086 C cross software development tools are intended to develop software to be run on hardware consisting of an 8086 CPU and an 8087 math processor. In usual applications one just develops the software in high level language and the operating systems of the host and target systems will take care of everything else. In an N.mPc simulation, there is no operating system on top of the simulated hardware so that the user has to initialize the simulated hardware before he runs a program. The initialization of a simulated 8086 CPU serves the following purposes:

1) It sets the instruction pointer (ip) and the code segment register (cs) to point to the starting address of the code.
2) It places the data area and the stack area appropriately into the data block reserved in memory by the loader. The Lantech cross software development tool requires that the 8086 stack and data segment registers be initialized with the same value.

To perform a proper initialization of the 8086 CPU one has to remember the following loading information from the previous development steps:

i) the address where the loaded program starts

ii) the beginning and the end of the common memory block reserved by the loader for the data and stack areas

In order to determine those values one has to look at the listing produced by the cross loader when using the -l option and find the starting addresses and sizes of code and data blocks. The end of the reserved memory area is verified by inspecting the corresponding simulated memory when the simulation is in runtime mode. The following example was made up to illustrate the initialization procedure discussed above.

1) Information from the cross loader (obtained when the -l option was used with cross loader):

- The program code block starting address is 400 hex (=1024).
- The data block starting address is 470 hex (=1136), the end of the reserved memory area is found at 600 hex (=1536). Memory areas not reserved by the loader are characterized by an "illegal memory access" message when being inspected. Use the "initrecord" option in OTOL to reserve a bigger memory space (see section 4.1.3).

2) Program counter initialization:

- Set the combination of instruction pointer (ip) and code segment register (cs) to the starting address of the code block.

NB: program counter = (16 * (cs)) + (ip)

Example: starting address = 1024 (=400hex)
- "deposit 1024 :ip"
- or: "deposit 64 :cs", "deposit 0:ip"
- or: "deposit 32 :cs
deposit 512 :ip"
- etc ...

3) Data segment, stack segment initialization:

- It should be noted that the Lantech 8086 C cross software development tools do not make any distinction between stack segment and data segment; the starting addresses of stack and data segments have to be identical which means that the "ds" and "ss" segment registers have to be initialized with identical values (see [19], page 3-6).
- Have the data and stack segment registers (and the extra segment register if it is used) point to the starting address of the data block: - "deposit 0x47 :ds"
- "deposit 0x47 :ss"
(- deposit 0x47 :es) (if "es" used)
- The stack grows downwards from its top; it is therefore reasonable to put the top of the stack near or at the end of the memory block reserved for data and stack by initializing the stack pointer with an appropriate value.
Example: - (end of reserved stack, data memory area - its base address) = 600 hex - 470 hex = 190 hex: "deposit 0x190:sp"

The initialization information as well as all the other essential

information about all "ready-to-run" simulations is found in the "0readme.fst" textfiles put into each simulation directory.

The step by step description of how to build and run the validation testprogram "VALCMD.C" on simulated 86/12 SBC hardware is found in Appendix D. The execution of the testprogram ("SIEVE.C") on the actual Intel SBC hardware is handled in a similar fashion to the execution of the validation testprogram shown in Appendix C.

With the exception of the initialization step, the execution of the "sieve.c" test is also analoguous to the execution of the validation testprogram. The "OREADME.FST" textfile in the simulation directory (86sieve) describes how to carry out the performance test.

6. COMPARISON OF SIMULATION AND ACTUAL RESULTS

6.1. Interpretation of the Validation Results

Figures 6-1a and 6-1b show the input and the output produced by running the validation testprogram ("VALCMD.C", see Appendix B) on the simulated 86/12 single board computer. The validation of N.mPc as a CAE tool for microprocessor simulations is done by comparing these results to the ones produced when "CMD.C" (or "HEXCMD.C") is run on the actual Intel SBC hardware. The program "CMD.C" only differs from "VALCMD.C" in the routines used for I/O operations. The results are identical if identical "angles" are entered by the user when running "CMD.C" on the simulated hardware or "CMD.C" (or "HEXCMD.C" for output in hex numbers) on the actual Intel SBC hardware. Figures 6-2a and 6-2b show the results obtained when running "HEXCMD.C" (see Appendix B) on an actual Intel 86/30 SBC using the same input angle as the one used to produce the simulation output shown in Figures 6-1a and 6-1b.

Thus the validation testprogram, written in the high level language C; and implementing a discrete, scaled version of the "Simple Attitude Control Algorithm" (SACL), was developed and run on both simulated and actual Intel SBC hardware, producing identical results.

```
  sho def
  USER$DISK1:[ICLI.NMPC.READYSIN.86VAL]
$ set val
$ run val
        Welcome to N.aPc/VMS
N.aPc: val
# dep 1024 :ip
# dep 0xaa :ss
# dep 0xaa :ds
# dep 280 :sp
# ru
MAX 11:45:34 VAL        CPU=00:00:31.97 PF=1926 IO=2421 MEM=307
Enter angle(degrees,2 digits):01
01
0000 00000000 00000000 0000 0000 0000

0001 00000020 00000004 6451 6451 1600

0002 00000020 0000000C 3CD3 3CD3 1600

0003 00000020 00000014 2522 2522 1600

0004 00000020 0000001C 16EB 16EB 1600

0005 00000020 00000024 0E64 0E64 0E64

0006 00000020 0000002C 0946 0946 0946

0007 00000020 00000034 0635 0635 0635

0008 00000020 0000003C 045E 045E 045E

0009 00000020 00000044 0344 0344 0344

000A 00000020 0000004C 029B 029B 029B

000B 00000020 00000054 0236 0236 0236

000C 00000020 0000005C 01F9 01F9 01F9

000D 00000020 00000064 01D5 01D5 01D5

000E 00000020 0000006C 01C0 01C0 01C0

000F 00000020 00000074 01B4 01B4 01B4

0010 00000020 0000007C 01AD 01AD 01AD

0011 00000020 00000084 01A9 01A9 01A9

0012 00000020 0000008C 01A7 01A7 01A7

0013 00000020 00000094 01A6 01A6 01A6

0014 00000020 0000009C 01A5 01A5 01A5

0015 00000020 000000A4 01A6 01A6 01A6

0016 00000020 000000AC 01A6 01A6 01A6

0017 00000020 000000B4 01A7 01A7 01A7

0018 00000020 000000BC 01A7 01A7 01A7

0019 00000020 000000C4 01A8 01A8 01A8

001A 00000020 000000CC 01A9 01A9 01A9

001B 00000020 000000D4 01AA 01AA 01AA

001C 00000020 000000DC 01AA 01AA 01AA

001D 00000020 000000E4 01AB 01AB 01AB

001E 00000020 000000EC 01AC 01AC 01AC

001F 00000020 000000F4 01AD 01AD 01AD

0020 00000020 000000FC 01AE 01AE 01AE

0021 00000020 00000104 01AE 01AE 01AE

0022 00000020 0000010C 01AF 01AF 01AF

0023 00000020 00000114 01B0 01B0 01B0

0024 00000020 0000011C 01B1 01B1 01B1

0025 00000020 00000124 01B2 01B2 01B2

0026 00000020 0000012C 01B2 01B2 01B2
```

Figure 6-1a: Running the Validation Program on the Simulated 86/12

```
0026 00000020 0000012C 01B2 01B2 01B2
0027 00000020 00000134 01B3 01B3 01B3
0028 00000020 0000013C 01B4 01B4 01B4
0029 00000020 00000144 01B5 01B5 01B5
002A 00000020 0000014C 01B6 01B6 01B6
002B 00000020 00000154 01B6 01B6 01B6
002C 00000020 0000015C 01B7 01B7 01B7
002D 00000020 00000164 01B8 01B8 01B8
002E 00000020 0000016C 01B9 01B9 01B9
002F 00000020 00000174 01BA 01BA 01BA
0030 00000020 0000017C 01BA 01BA 01BA
0031 00000020 00000184 01BB 01BB 01BB
0032 00000020 0000018C 01BC 01BC 01BC
0033 00000020 00000194 01BD 01BD 01BD
0034 00000020 0000019C 01BE 01BE 01BE
0035 00000020 000001A4 01BE 01BE 01BE
0036 00000020 000001AC 01BF 01BF 01BF
0037 00000020 000001B4 01C0 01C0 01C0
0038 00000020 000001BC 01C1 01C1 01C1
0039 00000020 000001C4 01C2 01C2 01C2
003A 00000020 000001CC 01C2 01C2 01C2
003B 00000020 000001D4 01C3 01C3 01C3
003C 00000020 000001DC 01C4 01C4 01C4
003D 00000020 000001E4 01C5 01C5 01C5
003E 00000020 000001EC 01C6 01C6 01C6
003F 00000020 000001F4 01C6 01C6 01C6
0040 00000020 000001FC 01C7 01C7 01C7
0041 00000020 00000204 01C8 01C8 01C8
0042 00000020 0000020C 01C9 01C9 01C9
0043 00000020 00000214 01CA 01CA 01CA
0044 00000020 0000021C 01CA 01CA 01CA
0045 00000020 00000224 01CB 01CB 01CB
0046 00000020 0000022C 01CC 01CC 01CC
0047 00000020 00000234 01CD 01CD 01CD
0048 00000020 0000023C 01CE 01CE 01CE
0049 00000020 00000244 01CE 01CE 01CE
004A 00000020 0000024C 01CF 01CF 01CF
004B 00000020 00000254 01D0 01D0 01D0
004C 00000020 0000025C 01D1 01D1 01D1
004D 00000020 00000264 01D2 01D2 01D2
004E 00000020 0000026C 01D2 01D2 01D2
004F 00000020 00000274 01D3 01D3 01D3
0050 00000020 0000027C 01D4 01D4 01D4
Enter angle(degrees,2 digits):
MAX 09:00:14 VAL        CPU=04:50:17.19 PF=1938 IO=5890 MFH=319
q
^Y
$
```

Figure 6-1b: Running the Validation Program on the Simulated 86/12

```
  run hexcmd
Enter the commanded angle :
1
 0 0 0 0 0 0
 1 20 4 6451 6451 1600
 2 20 c 3cd3 3cd3 1600
 3 20 14 2522 2522 1600
 4 20 1c 16eb 16eb 1600
 5 20 24 e64 e64 e64
 6 20 2c 946 946 946
 7 20 34 635 635 635
 8 20 3c 45e 45e 45e
 9 20 44 344 344 344
 a 20 4c 29b 29b 29b
 b 20 54 236 236 236
 c 20 5c 1f9 1f9 1f9
 d 20 64 1d5 1d5 1d5
 e 20 6c 1c0 1c0 1c0
 f 20 74 1b4 1b4 1b4
 10 20 7c 1ad 1ad 1ad
 11 20 84 1a9 1a9 1a9
 12 20 8c 1a7 1a7 1a7
 13 20 94 1a6 1a6 1a6
 14 20 9c 1a5 1a5 1a5
 15 20 a4 1a6 1a6 1a6
 16 20 ac 1a6 1a6 1a6
 17 20 b4 1a7 1a7 1a7
 18 20 bc 1a7 1a7 1a7
 19 20 c4 1a8 1a8 1a8
 1a 20 cc 1a9 1a9 1a9
 1b 20 d4 1aa 1aa 1aa
 1c 20 dc 1aa 1aa 1aa
 1d 20 e4 1ab 1ab 1ab
 1e 20 ec 1ac 1ac 1ac
 1f 20 f4 1ad 1ad 1ad
 20 20 fc 1ae 1ae 1ae
 21 20 104 1ae 1ae 1ae
 22 20 10c 1af 1af 1af
 23 20 114 1b0 1b0 1b0
 24 20 11c 1b1 1b1 1b1
 25 20 124 1b2 1b2 1b2
 26 20 12c 1b2 1b2 1b2
 27 20 134 1b3 1b3 1b3
 28 20 13c 1b4 1b4 1b4
 29 20 144 1b5 1b5 1b5
 2a 20 14c 1b6 1b6 1b6
 2b 20 154 1b6 1b6 1b6
 2c 20 15c 1b7 1b7 1b7
 2d 20 164 1b8 1b8 1b8
 2e 20 16c 1b9 1b9 1b9
 2f 20 174 1ba 1ba 1ba
 30 20 17c 1ba 1ba 1ba
 31 20 184 1bb 1bb 1bb
 32 20 18c 1bc 1bc 1bc
 33 20 194 1bd 1bd 1bd
 34 20 19c 1be 1be 1be
 35 20 1a4 1be 1be 1be
 36 20 1ac 1bf 1bf 1bf
 37 20 1b4 1c0 1c0 1c0
 38 20 1bc 1c1 1c1 1c1
 39 20 1c4 1c2 1c2 1c2
 3a 20 1cc 1c2 1c2 1c2
 3b 20 1d4 1c3 1c3 1c3
 3c 20 1dc 1c4 1c4 1c4
 3d 20 1e4 1c5 1c5 1c5
 3e 20 1ec 1c6 1c6 1c6
 3f 20 1f4 1c6 1c6 1c6
 40 20 1fc 1c7 1c7 1c7
 41 20 204 1c8 1c8 1c8
 42 20 20c 1c9 1c9 1c9
 43 20 214 1ca 1ca 1ca
 44 20 21c 1ca 1ca 1ca
 45 20 224 1cb 1cb 1cb
 46 20 22c 1cc 1cc 1cc
 47 20 234 1cd 1cd 1cd
 48 20 23c 1ce 1ce 1ce
 49 20 244 1ce 1ce 1ce
 4a 20 24c 1cf 1cf 1cf
 4b 20 254 1d0 1d0 1d0
 4c 20 25c 1d1 1d1 1d1
 4d 20 264 1d2 1d2 1d2
 4e 20 26c 1d2 1d2 1d2
 4f 20 274 1d3 1d3 1d3
 50 20 27c 1d4 1d4 1d4
```

Figure 6-2a: Running the Validation Program on the actual Intel SBC Hardware (output in hex)

```
run cmd
Enter the commanded angle :
1
 0 0 0 0 0 0
 1 32 4 25681 25681 5632
 2 32 12 15571 15571 5632
 3 32 20 9506 9506 5632
 4 32 28 5867 5867 5632
 5 32 36 3684 3684 3684
 6 32 44 2374 2374 2374
 7 32 52 1589 1589 1589
 8 32 60 1118 1118 1118
 9 32 68 836 836 836
 10 32 76 667 667 667
 11 32 84 566 566 566
 12 32 92 505 505 505
 13 32 100 469 469 469
 14 32 108 448 448 448
 15 32 116 436 436 436
 16 32 124 429 429 429
 17 32 132 425 425 425
 18 32 140 423 423 423
 19 32 148 422 422 422
 20 32 156 421 421 421
 21 32 164 422 422 422
 22 32 172 422 422 422
 23 32 180 423 423 423
 24 32 188 423 423 423
 25 32 196 424 424 424
 26 32 204 425 425 425
 27 32 212 426 426 426
 28 32 220 426 426 426
 29 32 228 427 427 427
 30 32 236 428 428 428
 31 32 244 429 429 429
 32 32 252 430 430 430
 33 32 260 430 430 430
 34 32 268 431 431 431
 35 32 276 432 432 432
 36 32 284 433 433 433
 37 32 292 434 434 434
 38 32 300 434 434 434
 39 32 308 435 435 435
 40 32 316 436 436 436
 41 32 324 437 437 437
 42 32 332 438 438 438
 43 32 340 438 438 438
 44 32 348 439 439 439
 45 32 356 440 440 440
 46 32 364 441 441 441
 47 32 372 442 442 442
 48 32 380 442 442 442
 49 32 388 443 443 443
 50 32 396 444 444 444
 51 32 404 445 445 445
 52 32 412 446 446 446
 53 32 420 446 446 446
 54 32 428 447 447 447
 55 32 436 448 448 448
 56 32 444 449 449 449
 57 32 452 450 450 450
 58 32 460 450 450 450
 59 32 468 451 451 451
 60 32 476 452 452 452
 61 32 484 453 453 453
 62 32 492 454 454 454
 63 32 500 454 454 454
 64 32 508 455 455 455
 65 32 516 456 456 456
 66 32 524 457 457 457
 67 32 532 458 458 458
 68 32 540 458 458 458
 69 32 548 459 459 459
 70 32 556 460 460 460
 71 32 564 461 461 461
 72 32 572 462 462 462
 73 32 580 462 462 462
 74 32 588 463 463 463
 75 32 596 464 464 464
 76 32 604 465 465 465
 77 32 612 466 466 466
 78 32 620 466 466 466
 79 32 628 467 467 467
 80 32 636 468 468 468
```

Figure 6-2b: Running the Validation Program on the actual Intel SBC Hardware (output in decimal)

6.2. Performance of Simulated and Real Hardware

As another application for the high level software capabilities of the 86/12 simulation developed during this work, it was decided to run performance tests using a high level benchmark program in C. This program could be run on any machine equipped with a C compiler. The machines being compared in this case were the VAX 11-780, the iSBC 86/12 single board computer and a simulation of the 86/12 using N.mPc on the VAX 11/780. To get reasonable and measurable execution times for the "sieve" benchmark, the program was executed 100,000 times on the VAX and on the 86/12 while one single execution of the "sieve" on the 86/12 simulation was sufficiently long. The factor 100,000 was later taken into account when comparing the relative performance of each processor. The "sieve" benchmark uses the "long integer" data type for variables on all three machines. The execution times on the VAX were measured in CPU time. The details about how the "sieve" benchmark was run on different machines are to be found in Appendix D.2. The performance comparison of real machines to a simulated one shows of course a significant performance penalty due to the simulation overhead. The following execution times were measured when running the "sieve" benchmark:

- VAX 11/780 : 17 sec. for 100,000 runs of "sieve"
- iSBC86/12 : 211 sec. for 100,000 runs of "sieve"
- Simulation of the iSBC 86/12 using N.mPc on a VAX 11-780: 402 sec. for one run of "sieve"

Expressed in performance ratio (P for Performance) this means:

$P_{VAX} : P_{8612} : P_{Simulation\ 8612} \approx 2{,}364{,}000 : 190{,}000 : 1$

Figure 6-3 shows the same results in a graphical form. These performance comparisons only serve to get a notion of the order of magnitude of the performance penalty caused by the N.mPc simulation overhead and do not claim to be exact as compiler differences may influence the results. See Appendix D or the "readme" file in the "86sieve" directory for details on the performance comparisons.

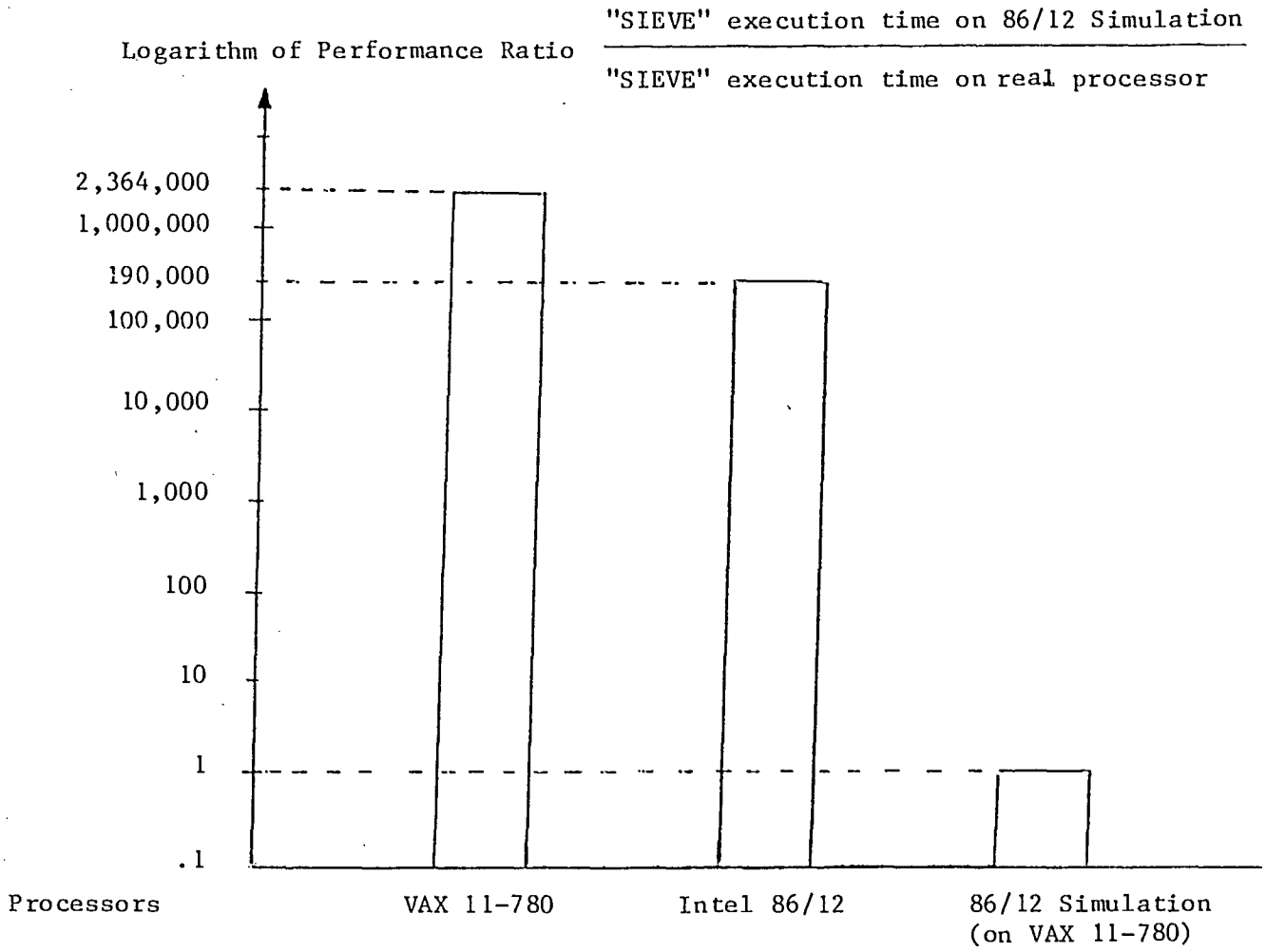


Figure 6-3

Performance Penalty due to Simulation using N.mPc

7. SUMMARY AND CONCLUSIONS

The following is a summary of the main results of the study presented in this report:

- A simulation of an Intel iSBC 86/12 single board computer was performed successfully. It included an 8086 CPU, a dual port RAM, a ROM, a programmable interrupt controller (PIC), a Multibus interface, a global memory and an I/O facility based on N.mPc's "raw memory" feature. These modules were designed and thoroughly tested and debugged during the course of the work.
- The program "OTOL" was used to transfer object code produced by the cross compiler to the "l.out" format needed by N.mPc . The program was subsequently upgraded in order to handle code intended to be run on an 8086 CPU. This made it possible to do software development for the 86/12 simulation in the high level language "C", using the C 8086 cross software development tools by Lantech Inc.
- The validation of N.mPc as a CAE tool for microprocessor simulations was performed by implementing a "Simple Attitude Control Algorithm" as a "C" program, which was successfully run on the simulated 86/12 single board computer as well as on the actual Intel SBC hardware. The validation not only established the reliability of the simulated hardware but also demonstrated the I/O capabilities of the 86/12 simulation. The development of the validation program in "C" demonstrated the potential of the high level software development path introduced as a result of this work.

The work reported in this document was an opportunity to acquire considerable expertise with the CAE tool N.mPc. As a result, it has

been felt that N.mPc is in fact a valuable tool in the development of a computer system.

For the case of the Intel 86/12 simulation the following observations were felt to be strong points of the CAE tool N.mPc:

- The design of the hardware modules of the simulation of the 86/12 SBC showed N.mPc's flexibility; making hardware changes due to new requirements or correcting design errors could be performed easily and efficiently.
- The debugging of the description of the 8086 CPU demonstrated the power of N.mPc's monitoring, control and debugging features which allow the tracing of errors in hardware descriptions.
- This work showed the feasibility of a high level language software development for N.mPc microprocessor simulations.
- The major effort of writing a hardware description of the 8086 CPU could be reduced by using a description from the library of descriptions of existing microprocessors (this library was delivered with the N.mPc system).

The following observations are based upon the 86/12 simulation work and outline some difficulties experienced with N.mPc:

- The C programs which were executed on the simulated 86/12 SBC were limited to be of a moderate size. This restriction was due to the simulation performance penalty resulting from the simulation overhead.
- Errors in the 8086 CPU description from the N.mPc's microprocessor description library required a considerable debugging effort. Just as the testing of VLSI chips is a major

problem of today's semiconductor industry, the same problem occurs in N.mPc descriptions of complex hardware modules. As in all software endeavours, the larger the programs one can run successfully, the more confidence one can have in a microprocessor description, for example. At the present time, there is no straightforward, systematical way to find all the possible bugs in a complex hardware description.

- The VMS version of the N.mPc run-time environment behaved strangely, at times, when faced with some subtle hardware description errors. The result was a catastrophic exit to VMS which left one unable to ascertain the cause of the failure systematically.

Some general conclusions can be drawn from the experience gained during the work documented in this report. Some of the strong points of a design approach using computer aided design tools such as N.mPc are listed below:

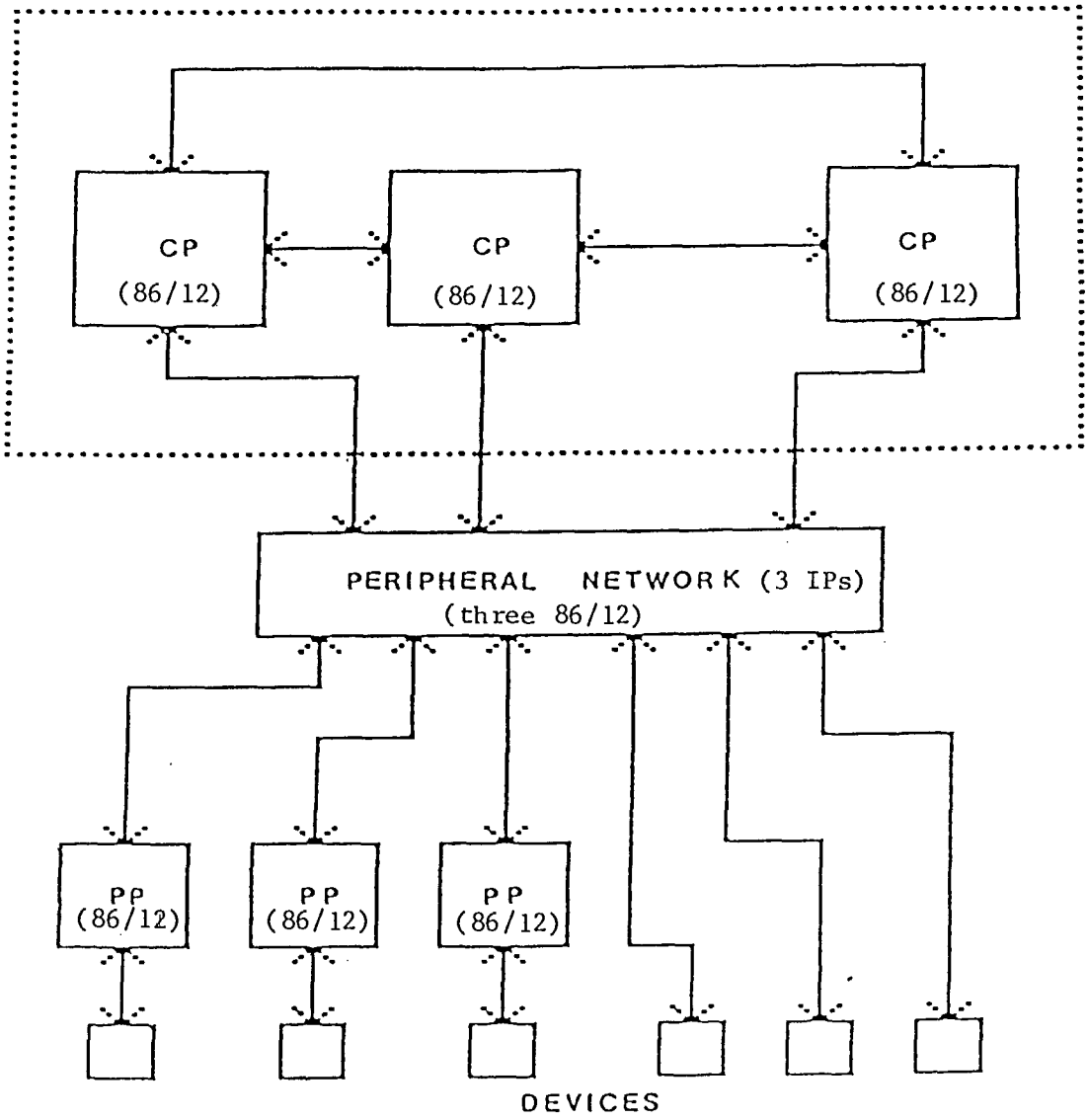
- N.mPc introduces the inherent flexibility of software into hardware design.
- Complex computer systems can quickly be simulated by taking a few hardware descriptions from a library and "interconnecting" them in a topology file.
- N.mPc separates the logical hardware design problems from the technical ones (timing, etc.) so that they can be solved separately.
- N.mPc not only offers an entirely programmable software development environment for microprocessor simulations but also allows to use of commercial cross software development

packages. These packages make it possible to develop software for microprocessor simulations in a high level language.

Of course, the advantages gained by the use of CAE tools come at a price:

- The performance penalty paid by N.mPc microprocessor simulations is significant and clearly limits the size of software to be run on a simulated processor. This simulation performance penalty will be alleviated by the ever increasing performance and decreasing cost of host computer systems.
- N.mPc's library of hardware descriptions should be improved in order to contain descriptions of existing hardware elements (microprocessors, etc.) with a reliability that is similar to the actual hardware. The hardware designer using the N.mPc could then directly use these descriptions without having to worry about debugging.

It is interesting to note that the design of a fault tolerant computer architecture for space applications (see Figure 7-1) was part of the work done under this contract. In that section of the work, a fault tolerant multiprocessor architecture was developed and it was decided to simulate the operations of such an architecture with N.mPc. The 8086 CPU was chosen as the generic processor for the elements of the multiprocessor system. N.mPc makes possible the simulation of several 8086 based processing elements at a very low incremental effort. This is where a CAE tool like N.mPc really shows its inherent power.



CP = Central Processor
IP = Interface Processor
PP = Peripheral Processor

Figure 7-1: Fault Tolerant Computer Architecture

REFERENCES

[1] Ordy, G.M., "N.mPc: Runtime User's Manual," Department of Computer Engineering and Science, Case Western Reserve University, 1979.

[2] Ordy, G.M. and Rogers, L.A., "N.mPc: MetaMicro User's Manual," Department of Computer Engineering and Science, Case Western Reserve University, 1979.

[3] Rogers, L.A., "N.mPc: Linking Loader User's Manual," Department of Computer Engineering and Science, Case Western Reserve University, 1979.

[4] Ordy, G.M., "N.mPc: Ecologist User's Manual," Department of Computer Engineering and Science, Case Western Reserve University, 1979.

[5] Leffler, S.J., "PP: A Post-Processor for N.mPc," Department of Computer Engineering and Science, Case Western Reserve University, 1979.

[6] Rogers, L.A., "A Generalized Linking/Loader for the Allocation of Code in Vertical and Horizontal Machines," Master of Science Thesis, Department of Computer Engineering and Science, Case Western Reserve University Report CES-79-6, August 1978.

[7] Streit, M., "VAX 11-780 CAE Tools for Multiprocessor Simulation: N.mPc User's and Application Manual and Installation Guide", a Report prepared by Intellitech, September 1984.

[8] Streit, M., "Simulation of the SBP 9989 Microprocessor Using the Computer Aided Engineering Tool N.mPc on a VAX 11/780", a Report prepared by Intellitech, September 1984.

[9] Parke, F.I., "An Introduction to N.mPc Design Environment", Proceedings of the ACM/IEEE Design Automation Conference, June 1979.

[10] Rose, C.W., Rogers, L.A., and Straubs, R.V., "The N.mPc System Description Facility," Proceeding of ACM/IEEE Design Automation Conference, June 1979.

[11] Hewitt, D.C., Parke, F.I., and Rose, C.W., "The N.mPc Runtime Environment," Proceedings of the ACM/IEEE Design Automation Conference, June 1979.

[12] Hewitt, D.C., "The Runtime Environment for N.mPc, An Adaptable System to Support the Development of Microprocessor-Based Systems", Master of Science Thesis, Department of Computer Engineering and Science, Case Western Reserve University Report CES-79-7, January 1978.

REFERENCES CONTINUED

[13] Jiang, W., "A Distributed Kernel Runtime Environment for Large N.mPc System Simulation", Master of Science Thesis, Department of Computer Engineering and Science, Case Western Reserve University Report CES-82-7, August 1982.

[14] Boucouris, S., "Design and Analysis of Fault Tolerant Architectures for Multi-Microprocessor Systems", Intellitech Technical Report, October 1984.

[15] Ordy, G., "N.2 ISP´ Users´s Manual", January 1984.

[16] Mahmoud, S.A., "VAX 11/780 CAE Tools for Multiprocessor Simulation - N.mPc Detailed System Description", September 1984.

[17] Straubs, R., "ISP´ User´s Manual", 1978.

[18] "Introduction to N.mPc System Programs", Technical Report, Case Western University, 1980.

[19] Lantech Systems Inc., "8086 C Cross Software Tools", 1983.

[20] Boucouris, S., "Conceptual Design of a Fault Tolerant Multiprocessor Operating System and the Implementation of a Prototype Kernel", Intellitech Technical Report, October 1984.

[21] Ordy, G., "N.mPc under VMS-Preliminary Paper", 1984.

[22] Ordy, G., "A Simple VAX N.mPc Post Processor", January 1984.

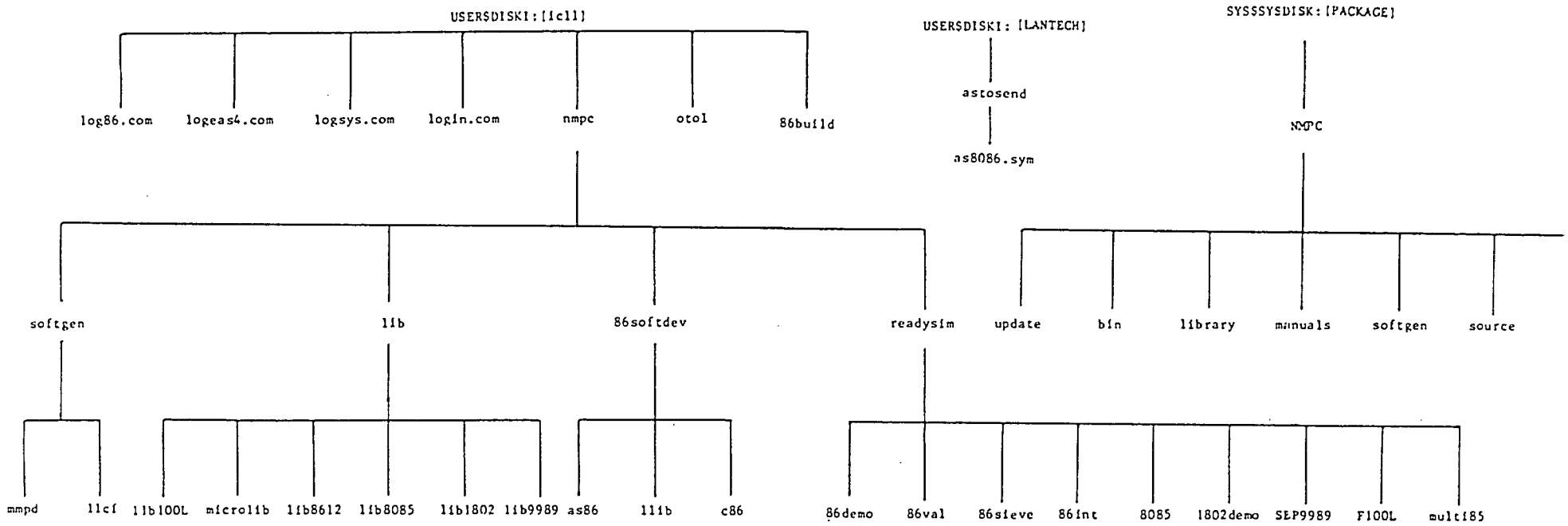
APPENDIX A:

Complete Directory of Intel :SBC 86/12 Files

The listings of the hardware descriptions that are part of the simulated 86/12 single board computer are too long to be appended to this manual. The following list indicates the file names used for each element of the 86/12 model and the names of the directories in which these elements can be found. The ISP´ source files containing the descriptions of the elements of the simulated Intel 86/12 single board computer are:

- 8086 CPU: "max86cpu.isp"
- ROM: "max86mem.isp"
- Dualport RAM: "dpram.isp"
- PIC: "pic.isp"
- Interrupt Generator: "interrupt.isp"
- Raw Memory: "terminal.isp"
- Multibus Interface: "multint86.isp"
- Global Memory: "globalmem.isp"
- Description file of 86/12 topology: "val.t"

The above listed files of the simulated 86/12 single board computer are found in the following directories in the present N.mPc directory structure presented on the next page: "lib 8612", "86 demo", "86 val", "86 sieve". Directory listings of these four directories are also part of this Appendix.



The N.mPc Directory Structure on VAX/VMS

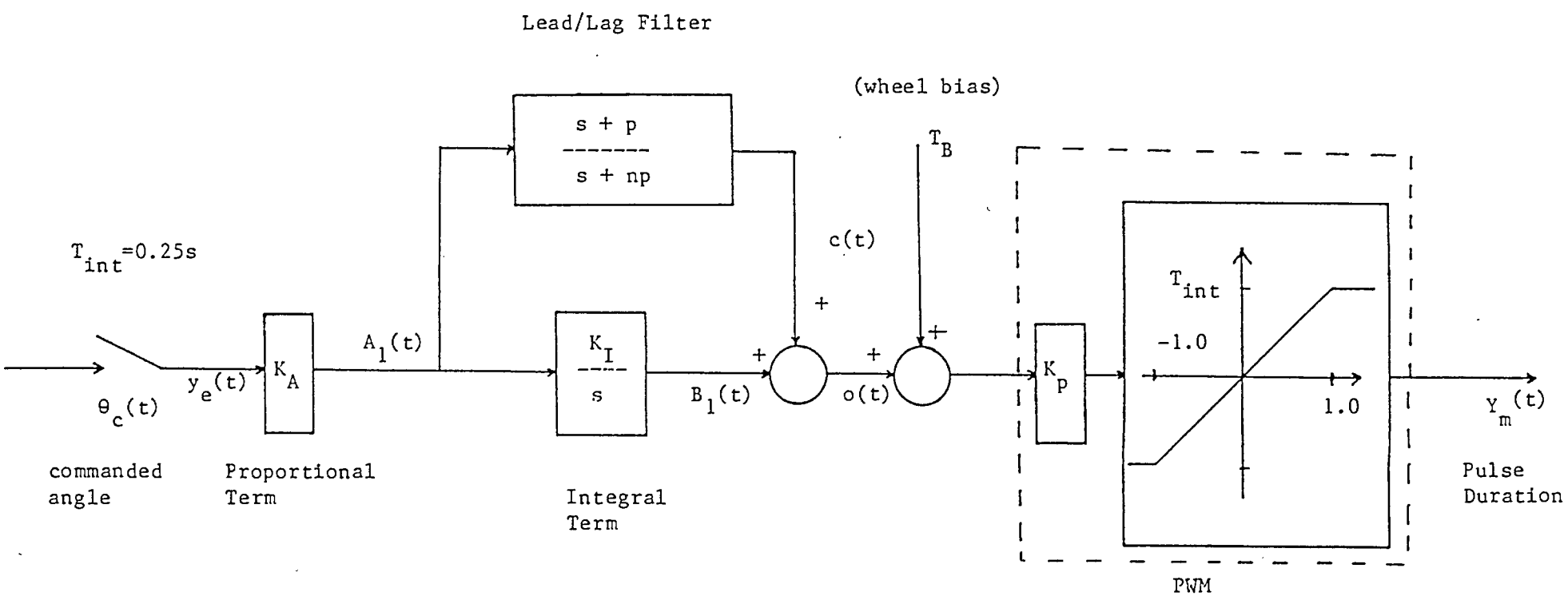
```
  set def 86val
$ dir3

Directory USER$DISK1:[ICL1.NMPC.READYSIM.86VAL]

OREADME.FST;9          CMD.C;2              CMD.EXE;2
CMD.MAI;2              CMD.MP1;2            CMD.OBJ;1
DPRAM.ISP;11           DPRAM.OBJ;6          EDTINI.EDT;1
GBLCORE.;4             GBLCORE.P;219        GLOBALMEM.ISP;10
GLOBALMEM.OBJ;7        HEXCMD.C;2           HEXCMD.EXE;1
HEXCMD.OBJ;1           IN.S;3               INTERRUPT.ISP;7
INTERRUPT.OBJ;7        MAX86CPU.ISP;5       MAX86CPU.OBJ;5
MAX86MEM.ISP;36        MAX86MEM.OBJ;44      MULTINT86.ISP;27
MULTINT86.OBJ;21       PIC.ISP;24           PIC.OBJ;20
PRINT.S;3              RAMCORE.;98          RAMCORE.P;607
ROMCORE.;77            ROMCORE.P;185        SACLZ.DAT;3
SACLZ.EXE;1            SACLZ.FOR;2          SACLZI.DAT;5
SACLZI.EXE;2           SACLZI.FOR;3         TERMINAL.ISP;9
TERMINAL.OBJ;5         VAL.D;257            VAL.EXE;44
VAL.F;46               VAL.S;46             VAL.T;5
VAL.X;184              VALCMD.C;2           VALCMD.OUT;3
VALCMD.S;1

Total of 49 files.
$ set def 86demo
$ dir3

Directory USER$DISK1:[ICL1.NMPC.READYSIM.86DEMO]

OREADME.FST;5          86ASMOTOL.COM;8      86ASS.OUT;1
86ASS.S;3              CC8086.COM;1         DPRAM.ISP;11
DPRAM.OBJ;6            EDTINI.EDT;1         GBLCORE.;4
GLOBALMEM.ISP;10       GLOBALMEM.OBJ;7      INTERRUPT.ISP;1
INTERRUPT.OBJ;5        MAX86CPU.DAT;2       MAX86CPU.ISP;246
MAX86CPU.OBJ;183       MAX86MEM.ISP;36      MAX86MEM.OBJ;44
MULTINT86.ISP;27       MULTINT86.OBJ;21     PIC.ISP;1
PIC.OBJ;19             PRIME.C;4            PRIME.HEX;3
PRIME.OUT;3            PRIME.S;5            PRIME.S;4
PRIME.S;3              PRIME.S;2            RAMCORE.;75
ROMCORE.;20            ROMCORE.;19          TERMINAL.ISP;1
TERMINAL.OBJ;4         VAL.EXE;17           VAL.F;18
VAL.S;18               VAL.T;4

Total of 38 files.
$
```

```
  set def 86int
$ dir3

Directory USER$DISK1:[ICL1.NMPC.READYSIM.86INT]

OREADME.FST;2          EDTINI.EDT;1          GBLCORE.;4
GBLCORE.P;82           INTDEMO.OUT;5         INTDEMO.S;1
MAX86CPU.DAT;1         MAX86CPU.OBJ;181      RAMCORE.;91
RAMCORE.P;470          ROMCORE.;14           ROMCORE.P;83
VAL.D;143              VAL.EXE;35            VAL.F;37
VAL.S;37               VAL.T;5               VAL.X;118

Total of 18 files.
$ set def lib8612
$ dir3

Directory USER$DISK1:[ICL1.NMPC.LIB.LIB8612]

OREADME.FST;2          DPRAM.ISP;11          DPRAM.OBJ;6
GLOBALMEM.ISP;10       GLOBALMEM.OBJ;7       INTERRUPT.ISP;7
INTERRUPT.OBJ;7        MAX86CPU.DAT;3        MAX86CPU.ISP;246
MAX86CPU.OBJ;184       MAX86MEM.ISP;36       MAX86MEM.OBJ;44
MULTINT86.ISP;27       MULTINT86.OBJ;21      PIC.ISP;24
PIC.OBJ;20             TERMINAL.ISP;9        TERMINAL.OBJ;5

Total of 18 files.
$ set def 86sieve
$ dir3

Directory USER$DISK1:[ICL1.NMPC.READYSIM.86SIEVE]

OREADME.FST;4          86SIEVE.C;2           86SIEVE.S;1
EDTINI.EDT;1           GBLCORE.;4            GBLCORE.P;36
NMPCSIEVE.C;6          NMPCSIEVE.OUT;6       NMPCSIEVE.S;3
RAMCORE.;75            RAMCORE.P;424         ROMCORE.;15
ROMCORE.P;37           SIEVE.C;1             SIEVE.EXE;2
SIEVE.OBJ;1            VAL.D;80              VAL.EXE;12
VAL.F;12               VAL.S;12              VAL.T;5
VAL.X;72               VAXSIEVE.C;4          VAXSIEVE.EXE;6
VAXSIEVE.OBJ;1

Total of 25 files.
$
```

APPENDIX B:

1) Mathematical Basiss of the "Simple Attitude Control Algorithm"

Simple Attitude Control Loop(SACL):



Input: $\theta_c(t)$, Commanded Angle

Output: $Y_m(t)$, Pulse Duration

Z transformation based on Tustin's Method

let $s = \frac{2}{T}\frac{(z-1)}{(z+1)}$ where $T = Tint = 0.25s$

let $F = \frac{2}{T}$

(i) lead/lag filter

$$G_1(s) = \frac{s+p}{s+np},\ G_1(z) = \frac{\frac{2}{T}\frac{(z-1)}{(z+1)} + p}{\frac{2}{T}\frac{(z-1)}{(z+1)} + np} = \frac{F(z-1) + p(z+1)}{F(z-1) + np(z+1)}$$

$$= \frac{(F+p)\ z+p-F}{(F+np)\ z+np-F}$$

$$= \frac{a_{11}\ z+ a_{10}}{b_{11}\ z+ b_{10}}$$

where $a_{10} = p-F$, $a_{11} = F+p$

$b_{10} = np-F$, $b_{11} = F+np$

(ii) integral term

$$G_2(s) = \frac{K_I}{s}\quad G_2(z) = \frac{K_I}{\frac{2}{T}\frac{(z-1)}{(z+1)}} = \frac{K_I\ (z+1)}{F\ (z-1)} = \frac{K_I\ z+ K_I}{Fz-F}$$

$$= \frac{a_{11}\ z+ a_{20}}{b_{21}\ z+ b_{20}}$$

where $a_{20} = K_I$, $a_{21} = F_I$

$b_{20} = -F$, $b_{21} = F$

difference equations

$$G_1(s) = \frac{C\ (s)}{A_1(s)} \qquad G_2(s) = \frac{B_1(s)}{A_1(s)}$$

$$\frac{C(z)}{A_1(z)} = \frac{a_{11}\ z+ a_{10}}{b_{11}\ z+ b_{10}} \qquad \frac{B_1(z)}{A_1(z)} = \frac{a_{21}\ z+ a_{20}}{b_{21}\ z+ b_{20}}$$

$$C(z)\ (b_{11}z + b_{10}) = A_1(z)\ (a_{11}z + a_{10})$$

$$b_{11}C\ (K+1) + b_{10}C\ (K) = a_{11}\ A_1(K+1) + a_{10}A_1(K)$$

$$C(K+1) = (\ a_{11}\ A_1(K+1) + a_{10}A_1(K) - b_{10}C(K))\ /b_{11}$$

$$C(K) = (\ a_{11}A_1(K) + a_{10}A_1(K-1) - b_{10}C(K-1))\ /b_{11}$$

$$B_1(z)\ (b_{21}z + b_{20}) = A_1(z)\ (a_{21}z + a_{20})$$

$$b_{21}B_1(K+1) + b_{20}B_1(K) = a_{21}A_1(K+1) + a_{20}A_1(K)$$

$$B_1(K+1) = (a_{21}A_1(K+1) + a_{20}A_1(K) - b_{20}B_1(K))/b_{21}$$

$$B_1(K) = (a_{21}A_1(K) + a_{20}A_1(K-1) - b_{20}B_1(K-1))/b_{21}$$

$$O(K) = B_1(K) + C(K)$$

$$A_1(K) = K_A - Y_e(K)$$

The resulting algorithm is fully implemented in the SACLZ program. The SACLZI program is scaled in order to use integer variables only.

To Compare Scaled Integer Results Against Real Results:

	t	$A_1(t)$	$B_1(t)$	O(t)	YI(t)	YM(t)
r	0.25	32.0	3.94×10^{-4}	25.6	25.6	0.25
SI	1	32	4	25681	25681	5632
conversion	1/4 = 0.25	32/1 = 32	$4/10{,}000_{-4}$ = 4×10^{-4}	25681/1,000 = 25.7	25681/1,000 = 25.7	5632/22528 = 0.25
r	20.0	32.0	6.27×10^{-2}	0.463	0.463	2.05×10^{-2}
SI	80	32	636	468	468	468
conversion	80/4 = 20	32/1 = 32	$636/10{,}000_{-2}$ = 6.36×10^{-2}	468/1,000 = 0.468	468/1,000 = 0.468	468/22528 = 2.08×10^{-2}

(r = real, SI = scaled integer)

For the scaled integer representation we now have:

i) tint x 4

the time interval, Δt, is scaled from 0.25 sec. to 1 sec., hence by a factor of 4. However the difference equations still use tint = 0.25 sec. in their implementation

ii) $B_1(t)$ x 10,000

the integrated output is scaled by a factor of 10,000

iii) $O(t)$ x 1,000

the PID output is scaled by a factor of 1,000

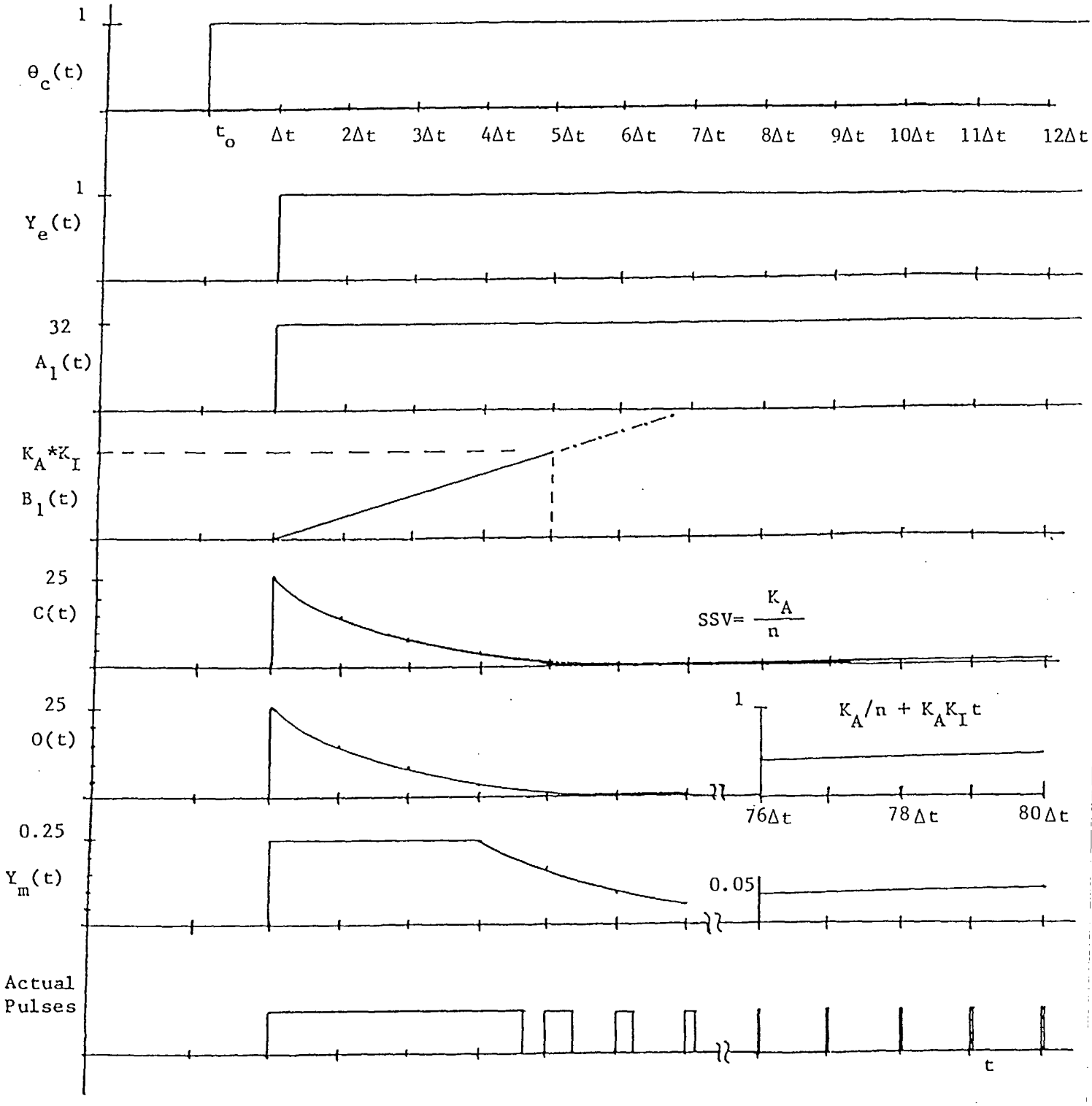
iv) $Y_I(t)$ x 1,000

the input to the PWM is scaled by a factor of 1,000

v) Ym(t) x 5632x4

the output of the PWM is scaled by a factor of 22528, this scale factor is the combination of 1,000 from $Y_I(t)$, 5.632 KP, and 4 from tint.

Theoretical Results from the SACL Algorithm(based on a continuous system)



```
      PROGRAM SACLZ
C
C THIS PROGRAM CONSISTS OF A SIMPLIFIED ATTITUDE
C CONTROL LOOP ALGORITHM BASED ON A PID CONTROLLER
C COMBINED WITH A PWM.
C
C WRITTEN BY   M. SAVOIE  -  JUNE, 1984.
C
C The attitude control algorithm computes a new PWM
C output pulse duration at every 250 msec. based on
C the measured attitude error.  For this simplified
C case, since there is no feedback, the attitude
C error is assigned the value of the commanded angle.
C The PID controller output is added to the wheel
C bias before being processed by the PWM.  The output
C pulse duration of the PWM represents a fraction of
C the time interval between samples and is calculated
C based on the magnitude and sign of the signal applied
C to it.
C
C DEFINITION OF VARIABLES
C   KA      -   PID input gain (---)
C   KI      -   PID integral gain (1/sec.)
C   N       -   PID frequency ratio (---)
C   P       -   PID lead frequency (rad/sec.)
C   KP      -   PWM gain (1/deg.)
C   TINT    -   integration step size (sec.)
C   TAU     -   PID time constant (sec.)
C   YEK     -   attitude error angle (deg.)
C   AK      -   PID proportional output (deg.)
C   BK      -   PID integral output (deg.)
C   CK      -   PID lag output (deg.)
C   OK      -   PID controller output (deg.)
C   TBK     -   wheel bias (deg.)
C   YIK     -   PWM input (deg.)
C   YMK     -   PWM output (duration in sec.)
C   THETAC  -   commanded angle (deg.)
C
C INITIALIZATION
      REAL KA,KI,N,P,KP,TINT,F,
     1     A10,A11,B10,B11,
     1     A20,A21,B20,B21,
     1     TBK,ERR,YEK,AK,AKM1,BK,BKM1,CK,CKM1,OK,
     1     YIK,YMK,T,THETAC
      OPEN(UNIT=1,TYPE='NEW',NAME='SACLZ.DAT')
C CONSTANTS
      KA=32.0
      KI=9.86E-5
```

```
        N=80.0
        P=0.0253
        KP=1.0/5.632
        TINT=0.25
        F=2.0/TINT
        TBK=0.0
C LEAD/LAG FILTER
        A10=P-F
        A11=F+P
        B10=N*P-F
        B11=F+N*P
C INTEGRAL TERM
        A20=KI
        A21=KI
        B20=-F
        B21=F
C INITIAL CONDITIONS
        ERR=0.0
        YEK=0.0
        AK=0.0
        AKM1=0.0
        BK=0.0
        BKM1=0.0
        CK=0.0
        CKM1=0.0
C MAIN LOOP
        T=0.0
        TYPE *,'Enter the commanded angle: '
        ACCEPT *,THETAC
        DO WHILE (T .LE. 20.0)
           YEK=ERR
           AKM1=AK
           AK=KA*YEK
           BKM1=BK
           BK=(A21*AK+A20*AKM1-B20*BKM1)/B21
           CKM1=CK
           CK=(A11*AK+A10*AKM1-B10*CKM1)/B11
           OK=BK+CK
           YIK=OK+TBK
           YMK=KP*YIK
           IF (ABS(YMK) .LE. 1.0) THEN
              YMK=YMK*TINT
           ELSE
              YMK=SIGN(1.0,YMK)*TINT
           ENDIF
           WRITE(1,10)T,AK,BK,OK,YIK,YMK
           ERR=THETAC
           T=T+TINT
```

```
      ENDDO
10    FORMAT(1X,F6.2,5(1X,E10.3))
      STOP
      END

$
```

```
  run saclz
Enter the commanded anslc:
1
FORTRAN STOP
$ cat saclz.dat
  0.00   0.000E+00   0.000E+00   0.000E+00   0.000E+00   0.000E+00
  0.25   0.320E+02   0.394E-03   0.256E+02   0.256E+02   0.250E+00
  0.50   0.320E+02   0.118E-02   0.154E+02   0.154E+02   0.250E+00
  0.75   0.320E+02   0.197E-02   0.937E+01   0.937E+01   0.250E+00
  1.00   0.320E+02   0.276E-02   0.575E+01   0.575E+01   0.250E+00
  1.25   0.320E+02   0.355E-02   0.359E+01   0.359E+01   0.159E+00
  1.50   0.320E+02   0.434E-02   0.230E+01   0.230E+01   0.102E+00
  1.75   0.320E+02   0.513E-02   0.154E+01   0.154E+01   0.682E-01
  2.00   0.320E+02   0.592E-02   0.108E+01   0.108E+01   0.480E-01
  2.25   0.320E+02   0.670E-02   0.809E+00   0.809E+00   0.359E-01
  2.50   0.320E+02   0.749E-02   0.647E+00   0.647E+00   0.287E-01
  2.75   0.320E+02   0.828E-02   0.551E+00   0.551E+00   0.245E-01
  3.00   0.320E+02   0.907E-02   0.494E+00   0.494E+00   0.219E-01
  3.25   0.320E+02   0.986E-02   0.461E+00   0.461E+00   0.204E-01
  3.50   0.320E+02   0.106E-01   0.441E+00   0.441E+00   0.196E-01
  3.75   0.320E+02   0.114E-01   0.430E+00   0.430E+00   0.191E-01
  4.00   0.320E+02   0.122E-01   0.423E+00   0.423E+00   0.188E-01
  4.25   0.320E+02   0.130E-01   0.419E+00   0.419E+00   0.186E-01
  4.50   0.320E+02   0.138E-01   0.418E+00   0.418E+00   0.185E-01
  4.75   0.320E+02   0.146E-01   0.417E+00   0.417E+00   0.185E-01
  5.00   0.320E+02   0.154E-01   0.417E+00   0.417E+00   0.185E-01
  5.25   0.320E+02   0.162E-01   0.417E+00   0.417E+00   0.185E-01
  5.50   0.320E+02   0.170E-01   0.417E+00   0.417E+00   0.185E-01
  5.75   0.320E+02   0.177E-01   0.418E+00   0.418E+00   0.186E-01
  6.00   0.320E+02   0.185E-01   0.419E+00   0.419E+00   0.186E-01
  6.25   0.320E+02   0.193E-01   0.419E+00   0.419E+00   0.186E-01
  6.50   0.320E+02   0.201E-01   0.420E+00   0.420E+00   0.187E-01
  6.75   0.320E+02   0.209E-01   0.421E+00   0.421E+00   0.187E-01
  7.00   0.320E+02   0.217E-01   0.422E+00   0.422E+00   0.187E-01
  7.25   0.320E+02   0.225E-01   0.422E+00   0.422E+00   0.188E-01
  7.50   0.320E+02   0.233E-01   0.423E+00   0.423E+00   0.188E-01
  7.75   0.320E+02   0.241E-01   0.424E+00   0.424E+00   0.188E-01
  8.00   0.320E+02   0.248E-01   0.425E+00   0.425E+00   0.189E-01
  8.25   0.320E+02   0.256E-01   0.426E+00   0.426E+00   0.189E-01
  8.50   0.320E+02   0.264E-01   0.426E+00   0.426E+00   0.189E-01
  8.75   0.320E+02   0.272E-01   0.427E+00   0.427E+00   0.190E-01
  9.00   0.320E+02   0.280E-01   0.428E+00   0.428E+00   0.190E-01
  9.25   0.320E+02   0.288E-01   0.429E+00   0.429E+00   0.190E-01
  9.50   0.320E+02   0.296E-01   0.430E+00   0.430E+00   0.191E-01
  9.75   0.320E+02   0.304E-01   0.430E+00   0.430E+00   0.191E-01
 10.00   0.320E+02   0.312E-01   0.431E+00   0.431E+00   0.191E-01
 10.25   0.320E+02   0.319E-01   0.432E+00   0.432E+00   0.192E-01
 10.50   0.320E+02   0.327E-01   0.433E+00   0.433E+00   0.192E-01
 10.75   0.320E+02   0.335E-01   0.434E+00   0.434E+00   0.192E-01
 11.00   0.320E+02   0.343E-01   0.434E+00   0.434E+00   0.193E-01
 11.25   0.320E+02   0.351E-01   0.435E+00   0.435E+00   0.193E-01
 11.50   0.320E+02   0.359E-01   0.436E+00   0.436E+00   0.193E-01
 11.75   0.320E+02   0.367E-01   0.437E+00   0.437E+00   0.194E-01
 12.00   0.320E+02   0.375E-01   0.437E+00   0.437E+00   0.194E-01
 12.25   0.320E+02   0.383E-01   0.438E+00   0.438E+00   0.195E-01
 12.50   0.320E+02   0.390E-01   0.439E+00   0.439E+00   0.195E-01
 12.75   0.320E+02   0.398E-01   0.440E+00   0.440E+00   0.195E-01
 13.00   0.320E+02   0.406E-01   0.441E+00   0.441E+00   0.196E-01
 13.25   0.320E+02   0.414E-01   0.441E+00   0.441E+00   0.196E-01
 13.50   0.320E+02   0.422E-01   0.442E+00   0.442E+00   0.196E-01
 13.75   0.320E+02   0.430E-01   0.443E+00   0.443E+00   0.197E-01
 14.00   0.320E+02   0.438E-01   0.444E+00   0.444E+00   0.197E-01
 14.25   0.320E+02   0.446E-01   0.445E+00   0.445E+00   0.197E-01
 14.50   0.320E+02   0.454E-01   0.445E+00   0.445E+00   0.198E-01
 14.75   0.320E+02   0.461E-01   0.446E+00   0.446E+00   0.198E-01
 15.00   0.320E+02   0.469E-01   0.447E+00   0.447E+00   0.198E-01
 15.25   0.320E+02   0.477E-01   0.448E+00   0.448E+00   0.199E-01
 15.50   0.320E+02   0.485E-01   0.449E+00   0.449E+00   0.199E-01
 15.75   0.320E+02   0.493E-01   0.449E+00   0.449E+00   0.199E-01
 16.00   0.320E+02   0.501E-01   0.450E+00   0.450E+00   0.200E-01
 16.25   0.320E+02   0.509E-01   0.451E+00   0.451E+00   0.200E-01
 16.50   0.320E+02   0.517E-01   0.452E+00   0.452E+00   0.200E-01
 16.75   0.320E+02   0.525E-01   0.452E+00   0.452E+00   0.201E-01
 17.00   0.320E+02   0.532E-01   0.453E+00   0.453E+00   0.201E-01
 17.25   0.320E+02   0.540E-01   0.454E+00   0.454E+00   0.202E-01
 17.50   0.320E+02   0.548E-01   0.455E+00   0.455E+00   0.202E-01
 17.75   0.320E+02   0.556E-01   0.456E+00   0.456E+00   0.202E-01
 18.00   0.320E+02   0.564E-01   0.456E+00   0.456E+00   0.203E-01
 18.25   0.320E+02   0.572E-01   0.457E+00   0.457E+00   0.203E-01
 18.50   0.320E+02   0.580E-01   0.458E+00   0.458E+00   0.203E-01
 18.75   0.320E+02   0.588E-01   0.459E+00   0.459E+00   0.204E-01
 19.00   0.320E+02   0.596E-01   0.460E+00   0.460E+00   0.204E-01
 19.25   0.320E+02   0.603E-01   0.460E+00   0.460E+00   0.204E-01
 19.50   0.320E+02   0.611E-01   0.461E+00   0.461E+00   0.205E-01
 19.75   0.320E+02   0.619E-01   0.462E+00   0.462E+00   0.205E-01
 20.00   0.320E+02   0.627E-01   0.463E+00   0.463E+00   0.205E-01
$
```

```
      PROGRAM SACLZI
C
C THIS PROGRAM CONSISTS OF A SIMPLIFIED ATTITUDE
C CONTROL LOOP ALGORITHM BASED ON A PID CONTROLLER
C COMBINED WITH A PWM.
C
C WRITTEN BY   M. SAVOIE  -  JUNE, 1984.
C
C The attitude control algorithm computes a new PWM
C output pulse duration at every 250 msec. based on
C the measured attitude error.  For this simplified
C case, since there is no feedback, the attitude
C error is assigned the value of the commanded angle.
C The PID controller output is added to the wheel
C bias before being processed by the PWM.  The output
C pulse duration of the PWM represents a fraction of
C the time interval between samples and is calculated
C based on the magnitude and sign of the signal applied
C to it.
C
C DEFINITION OF VARIABLES
C   KA      -   PID input gain (----)
C   KI      -   PID integral gain (1/sec.)
C   N       -   PID frequency ratio (----)
C   P       -   PID lead frequency (rad/sec.)
C   KP      -   PWM gain (1/deg.)
C   TINT    -   integration step size (sec.)
C   TAU     -   PID time constant (sec.)
C   YEK     -   attitude error angle (deg.)
C   AK      -   PID proportional output (deg.)
C   BK      -   PID integral output (deg.)
C   CK      -   PID lag output (deg.)
C   OK      -   PID controller output (deg.)
C   TBK     -   wheel bias (deg.)
C   YIK     -   PWM input (deg.)
C   YMK     -   PWM output (duration in sec.)
C   THETAC  -   commanded angle (deg.)
C
C INITIALIZATION
      INTEGER*4 KA,KI,N,P,KP,TINT,F,
     1          A10,A11,B10,B11,
     1          A20,A21,B20,B21,
     1          AKM1,BK,BKM1,CK,CKM1,OK,
     1          YIK,YMK,T,THETAC
      OPEN(UNIT=1,TYPE='NEW',NAME='SACLZI.DAT')
C CONSTANTS
      KA=32
C     KI=9.86E-5
```

```
C       N=80.0
C       P=0.0253
        KP=1            !(1.0/5.632)*5.632
        TINT=1          !F IS COMPUTED WITH TINT=0.25
C       F=2.0/TINT
        TBK=0
C LEAD/LAG FILTER
        A10=-79747      !(P-F)*10000
        A11=80253       !(F+P)*10000
        B10=-6          !N*P-F
        B11=10          !F+N*P
C INTEGRAL TERM
        A20=1           !KI*10000
        A21=1           !KI*10000
        B20=-8          !-F
        B21=8           !F
C INITIAL CONDITIONS
        ERR=0
        YEK=0
        AK=0
        AKM1=0
        BK=0
        BKM1=0
        CK=0
        CKM1=0
C MAIN LOOP
        T=0
        TYPE *,'Enter the commanded angle: '
        ACCEPT *,THETAC
        DO WHILE (T .LE. 80)
           YEK=ERR
           AKM1=AK
           AK=KA*YEK
           BKM1=BK
           BK=(A21*AK+A20*AKM1-B20*BKM1)/B21
           CKM1=CK
           CK=(A11*AK+A10*AKM1-B10*CKM1)/B11
           OK=(BK+CK)/10    !scale down by 10
           YIK=OK+TBK
           YMK=KP*YIK
           IF (JIABS(YMK) .LE. 5632) THEN
              YMK=YMK*TINT
           ELSE
              YMK=JISIGN(5632,YMK)*TINT
           ENDIF
           WRITE(1,10)T,AK,BK,OK,YIK,YMK
           ERR=THETAC
           T=T+TINT
```

```
      ENDDO
10    FORMAT(1X,I4,5(1X,I6))
      STOP
      END

$
```

```
  run saclzi
Enter the commanded angle:
1
FORTRAN STOP
$ cat saclzi.dat
   0      0      0      0      0      0
   1     32      4  25681  25681   5632
   2     32     12  15571  15571   5632
   3     32     20   9506   9506   5632
   4     32     28   5867   5867   5632
   5     32     36   3684   3684   3684
   6     32     44   2374   2374   2374
   7     32     52   1589   1589   1589
   8     32     60   1118   1118   1118
   9     32     68    836    836    836
  10     32     76    667    667    667
  11     32     84    566    566    566
  12     32     92    505    505    505
  13     32    100    469    469    469
  14     32    108    448    448    448
  15     32    116    436    436    436
  16     32    124    429    429    429
  17     32    132    425    425    425
  18     32    140    423    423    423
  19     32    148    422    422    422
  20     32    156    421    421    421
  21     32    164    422    422    422
  22     32    172    422    422    422
  23     32    180    423    423    423
  24     32    188    423    423    423
  25     32    196    424    424    424
  26     32    204    425    425    425
  27     32    212    426    426    426
  28     32    220    426    426    426
  29     32    228    427    427    427
  30     32    236    428    428    428
  31     32    244    429    429    429
  32     32    252    430    430    430
  33     32    260    430    430    430
  34     32    268    431    431    431
  35     32    276    432    432    432
  36     32    284    433    433    433
  37     32    292    434    434    434
  38     32    300    434    434    434
  39     32    308    435    435    435
  40     32    316    436    436    436
  41     32    324    437    437    437
  42     32    332    438    438    438
  43     32    340    438    438    438
  44     32    348    439    439    439
  45     32    356    440    440    440
  46     32    364    441    441    441
  47     32    372    442    442    442
  48     32    380    442    442    442
  49     32    388    443    443    443
  50     32    396    444    444    444
  51     32    404    445    445    445
  52     32    412    446    446    446
  53     32    420    446    446    446
  54     32    428    447    447    447
  55     32    436    448    448    448
  56     32    444    449    449    449
  57     32    452    450    450    450
  58     32    460    450    450    450
  59     32    468    451    451    451
  60     32    476    452    452    452
  61     32    484    453    453    453
  62     32    492    454    454    454
  63     32    500    454    454    454
  64     32    508    455    455    455
  65     32    516    456    456    456
  66     32    524    457    457    457
  67     32    532    458    458    458
  68     32    540    458    458    458
  69     32    548    459    459    459
  70     32    556    460    460    460
  71     32    564    461    461    461
  72     32    572    462    462    462
  73     32    580    462    462    462
  74     32    588    463    463    463
  75     32    596    464    464    464
  76     32    604    465    465    465
  77     32    612    466    466    466
  78     32    620    466    466    466
  79     32    628    467    467    467
  80     32    636    468    468    468
$
```

APPENDIX B

2) The Validation Testprogram Running on the Actual Intel SBC Hardware (CMD.C, = CMVAL.C86)

```
/*********************************************************/
/* C version of a Simplified Attitude Control Algorithm */
/*********************************************************/
/* This validation testprogram implements the "SACL" al-*/
/* gorithm using scaled integer variables.The "ABS"      */
/* function,not available in the link library of the     */
/* Lantech C 8086 cross software development tools, was */
/* replaced with an "ABS" subroutine.The resulting code */
/* contains no floating point instructions and can be    */
/* executed by an 8086 CPU.The C functions "scanf" and   */
/* "printf" are used for I/O operations as "CMD.C" is to*/
/* be run on an actual Intel SBC hardware(8086 CPU).This*/
/* is the only difference to the validation testprogram */
/* to be run on the simulated Intel SBC hardware         */
/* (VALCMD.C).                                           */
/*********************************************************/
/*    Max Streit, Intellitech Canada Ltd, Sept.84        */
/*********************************************************/

/*   #include math   ("exp" and "fabs" replaced)    */

main()
     {
       long int   ka,thetac;
       long int   a10,a11,b10,b11,a20,a21,b20,b21;
       long int   err,yek,ak,akm1,bk,bkm1,ck,ckm1;
       int        kp,t,tint,tbk,ok,yik,ymk;

/*     Initialization                              */

       ka = 32;
/*     ki = 9.86E-5;    */
/*     n  = 80;    */
/*     p  = 0.253; */
       kp = 1;      /*  (1.0/5.632)*5.632   */
       tint = 1;    /*  f is computed with tint=0.25   */
/*     f  = 2.0/tint  */
       tbk = 0;

/*  lead/lag filter  */
       a10 = -79747;    /*  (p-f)*10000   */
       a11 = 80253;     /*  (f+p)*10000   */
       b10 = -6;        /*  n*p-f   */
       b11 = 10;        /*  f+n*p   */

/*  integral term    */
       a20 = 1;         /*  ki*10000   */
       a21 = 1;         /*  ki*10000   */
```

```
        b20 = -8;        /*  -f  */
        b21 = 8;         /*  f  */

/*      initial conditions                      */

        err = 0;
        yek = 0;
        ak = 0;
        akm1 = 0;
        bk = 0;
        bkm1 = 0;
        ck = 0;
        ckm1 = 0;

/*      main loop                               */

        t  = 0;
        printf("Enter the commanded angle :\n");
        scanf("%D", &thetac);

        while (t <= 80)
            {
             yek = err;
             akm1 = ak;
             ak = ka * yek;
             bkm1 = bk;
             bk = (a21*ak+a20*akm1-b20*bkm1)/b21;
             ckm1 = ck;
             ck = (a11*ak+a10*akm1-b10*ckm1)/b11;
             ok = (bk + ck)/10;  /*  scaled down by 10  */
             yik = tbk + ok;
             ymk = kp * yik;

             if (abs(ymk) <= 5632)
                ymk = ymk * tint;
             else
                {
                 if (ymk > 0)  ymk =  5632 * tint;
                 else          ymk = -5632 * tint;
                }

          printf(" %d %D %D %d %d %d\n",t,ak,bk,ok,yik,ymk);

             err = thetac;
             t = t + tint;
            }
          }
```

```
 /* absolute value of floating point number */

abs(x)
int x;
{
    int z;

    if (x >= 0)
     z = x;
    else
     z = -x;

    return(z);
}
$
```

APPENDIX B

3) The Validation Testprogram Running on the Simulated Intel SBC Hardware ("VALCMD.C")

```
/*************************************************************/
/* C version of a Simplified Attitude Control Algorithm */
/*************************************************************/
/* This validation testprogram implements the "SACL" al-*/
/* gorithm using scaled integer variables.The "ABS"      */
/* function,not available in the link library of the     */
/* Lantech C 8086 cross software development tools, was */
/* replaced with an "ABS" subroutine.The resulting code */
/* contains no floating point instructions and can be    */
/* executed by an 8086 CPU.Normally the C functions      */
/* "scanf" and "printf" are used for I/O operations.As   */
/* "VALCMD.C" is to be run on a simulated 8086 CPU two   */
/* "C" callable procedures("PRINT","IN") were added to   */
/* the link library to allow for I/O operations which    */
/* use N.mPc's "raw memory" feature for I/O.             */
/*************************************************************/
/*    Max Streit, Intellitech Canada Ltd, Sept.84        */
/*************************************************************/

/*   #include math   ("exp" and "fabs" replaced)   */


main()
     {
      long int   ka,thetac;
      long int   a10,a11,b10,b11,a20,a21,b20,b21;
      long int   err,yek,ak,akm1,bk,bkm1,ck,ckm1;
      int        kp,t,tint,tbk,ok,yik,ymk;

/*    Initialization                              */

      ka = 32;
/*    ki = 9.86E-5;    */
/*    n  = 80;    */
/*    p  = 0.253; */
      kp = 1;      /*  (1.0/5.632)*5.632  */
      tint = 1;    /*  f is computed with tint=0.25  */
/*    f  = 2.0/tint  */
      tbk = 0;

/*  lead/lag filter  */
      a10 = -79747;    /*  (p-f)*10000  */
      a11 = 80253;     /*  (f+p)*10000  */
      b10 = -6;        /*  n*p-f  */
      b11 = 10;        /*  f+n*p  */

/*  integral term    */
      a20 = 1;         /*  ki*10000  */
```

```
        a21 = 1;          /*  ki*10000  */
        b20 = -8;         /*  -f   */
        b21 = 8;          /*  f  */

/*      initial conditions                          */

        err = 0;
        yek = 0;
        akm1 = 0;
        bkm1 = 0;
        ckm1 = 0;

/*      main loop                                   */

        t   = 0;
        ak  = 0;
        bk  = 0;
        ok  = 0;
       yik  = 0;
       ymk  = 0;
        ck  = 0;

/*        printf("Enter the commanded angle :\n");     */
/*        scanf("%D", &thetac);                        */
        thetac = in();  /* IO procedure for N.mPc */

        while (t <= 80)
            {
             yek = err;
             akm1 = ak;
             ak = ka * yek;
             bkm1 = bk;
             bk = (a21*ak+a20*akm1-b20*bkm1)/b21;
             ckm1 = ck;
             ck = (a11*ak+a10*akm1-b10*ckm1)/b11;
             ok = (bk + ck)/10;   /*  scaled down by 10  */
             yik = tbk + ok;
             ymk = kp * yik;

             if (abs(ymk) <= 5632)
                ymk = ymk * tint;
             else
                {
                 if (ymk > 0)  ymk =  5632 * tint;
                 else          ymk = -5632 * tint;
                }

          print(t,ak,bk,ok,yik,ymk);    /* IO procedure for N.mPc */
```

```
            err = thetac;
            t = t + tint;
          }
        }

 /* absolute value of floating point number */
abs(x)
int x;
{
    int z;

    if (x >= 0)
     z = x;
    else
     z = -x;

    return(z);
}

$
```

APPENDIX B

4) The Benchmark Program Used for Performance Tests

```
/* Sieve test program */
#define true 1
#define false 0
#define size 8190

char flags[size + 1];

main() {
        int i,prime,k,count,iter;

        printf("10 iterations\n");
        for (iter = 1; iter <= 10; iter++)
          {
           count = 0;
           for (i = 0; i <= size;i++)
           flags[i] = true;
           for (i = 0; i <= size;i++)
             {
              if (flags[i])
               {
                prime = i + i + 3;
                for (k = i + prime; k <= size; k += prime)
                  flags[k] = false;

                count++;
               }
             }
        }
      printf("\n%d primes.",count);
     }
$
```

```
  set def 86sieve
$ type vaxsieve.c
/*****************************************/
/*          Sieve test program           */
/* This version does more loops(100 000) in  */
/* a smaller number range(1...15) than the   */
/* orisinal sieve program in order to set a  */
/* C program that can be run (for a varyins  */
/* number of iterations)on the VAX, the(real)*/
/* 86/12 and the 86/12 on the N.mPc system.  */
/*****************************************/
/* Max Streit, Intellitech Canada Ltd, Aus.84*/
/*****************************************/
#define true 1
#define false 0
#define size 15


main() {
        long int flags[size + 1];
        long int i,prime,k,count,iter;

        printf("100000 iterations\n");
        for (iter = 1; iter <= 100000; iter++)
          {
           count = 0;
           for (i = 0; i <= size;i++)
           flags[i] = true;
           for (i = 0; i <= size;i++)
             {
              if (flags[i])
               {
                prime = i + i + 3;
                for (k = i + prime; k <= size; k += prime)
                  flags[k] = false;

                count++;
              }
            }
        }
      printf("\n%d primes.",count);
     }
$
```

```
/*********************************************/
/* "NMPCSIEVE.C": Sieve test program         */
/* (version to be run on N.mPc)              */
/*********************************************/
/* This version uses "long integer" variables*/
/* and does just one iteration in the 1...15 */
/* range of numbers.This is factor 100000    */
/* times less work than what the "vaxsieve"  */
/* program does.In this way a direct perfor- */
/* mance comparison between VAX and a 86/12  */
/* run on N.mPc with a reasonable simulation */
/* time on N.mPc is achieved.                */
/*********************************************/
/* INITIALIZATION of the corresponding simu- */
/* N.mPc simulation                          */
/*                                           */
/* Information from loader:                  */
/*       - end of reserved memory area:700hex */
/*       - starting address of data segment  */
/*         is 610hex;                        */
/*       - code segment starting address is  */
/*         400hex(=1024)                     */
/*       - a possible initialization:        */
/*            - "deposit 1024 :ip"           */
/*            - "deposit 0x61 :ds            */
/*               deposit 0x61 :ss"           */
/*              (0x61 = 61hex = 610/16)      */
/*            - "deposit 0xf0 :sp"           */
/*              (700hex chosen as top of     */
/*               stack;700hex - 610hex=F0hex)*/
/*********************************************/
/* Max Streit, Intellitech Canada Ltd, Aug.84*/
/*********************************************/
#define true 1
#define false 0
#define size 15


main() {
        long int flags[size + 1];
        long int i,prime,k,count,iter;

/*      printf("1 iteration\n");    */

         for (iter = 1; iter <= 1; iter++)
          {
           count = 0;
           for (i = 0; i <= size;i++)
```

```
            flags[i] = true;
            for (i = 0; i <= size;i++)
              {
               if (flags[i])
                {
                 prime = i + i + 3;
                 for (k = i + prime; k <= size; k += prime)
                   flags[k] = false;

                 count++;
                }
              }
          }

/*     printf("\n%d primes.",count);        */

      }
$
```

APPENDIX C

Test Software Development and Execution Procedure for the actual Intel SBC Hardware

Steps in generating and executing code on actual Intel 86/30 target HW running iRMX86

Create the program:

```
-TX CMVALID.C86
 I
 (key in program)
 cntrl Z
 Q
 E
```

(note: - is the iRMX86 prompt, TX invokes a screen text editor, I stands for insert, Q stands for quit, E stands for exit)

Compiled the program:

```
-CC86 CMVALID.C86 LARGE
```

(note: CC86 invokes the Intel C-86 compiler, LARGE specifies the large case (small model can have up to 64KB of code and 64KB of data, with all pointers occupying two bytes, large model can have access to the full addressing space of the 8086, each source file generates a distinct pair of code and data segments of up to 64KB in length, all pointers are four bytes long))

Link the program:

```
-SUBMIT TKB.CMD (CMVALID.OBJ,CMVALID)
```

(note: SUBMIT invokes a command file, TKB.CMD, CMVALID.OBJ is the object file generated from the compilation process, CMVALID is the name of the executable task)

Listing of the TKB.CMD command file:

```
link86 %0, &
:libr:lamain.obj, &
:libr:lclib.lib, &
:libr:large.lib, &
:libr:87null.lib &
to %1 fastload bind &
objectcotrols(purge) &
segsize(stack(+1000H)) &
mempool(+1000H,+0D000H)
```

(note: link86 invokes the 8086 linker, %0 and %1 are associated with the first and second parameters beins passed in the submit command line, & is a continuation character, :libr: is a logical name for a directory, lamain.obj, lclib.lib, large.lib, and 87null.lib are modules required to resolve symbols generated by the compiler)

Running the program:

```
-CMVALID
```

$

APPENDIX D

1) Test Software Development and Execution Procedure for Simulated Intel SBC Hardware

```
**************************************************
* N.MPC VALIDATION : C ON A 86/12 SIMULATION     *
**************************************************
* This simulation is described in detail in Intel- *
* litech's technical report "Validation of N.mPc   *
* Microprocessor Simulation"                       *
**************************************************
* This directory not only contains the files for   *
* the validation simulation and the testprogram to *
* be run for the validation("VALCMD.C") but also   *
* some files produced when developing the valida-  *
* tion testprogram. "SACLZ.FOR" is the original    *
* Fortran version of the validation algorithm; the *
* "SACLZI.FOR" file is a scaled version of         *
* "SACLZ.FOR" and uses integer variables. "CMD.C"  *
* is a translation of "SACLZ.FOR" into C. "CMD.C"  *
* also uses scaled integer variables in order *
* to produce code that, after cross compilation,can*
* be executed on an 8086 CPU."CMD.C" can be run on *
* any real machine that has a "C" compiler as it   *
* uses "C" standard functions for I/O("printf",    *
* "scanf") whereas the otherwise identical valida- *
* tion program run on the 86/12 simulation("VAL-   *
* CMD.C") uses N.mPc's "Raw Memory" feature for    *
* I/O. The procedures "PRINT.S" and "IN.S" have    *
* been written and added to the link library so    *
* that C programs can call them for convenient IO  *
* in this N.mPc simulation. The following text     *
* shows how to build the simulation of the Intel   *
* 86/12 SBC and execute  the testprogram "VALCMD.C"*
* on the simulated hardware.                       *
**************************************************

Steps in buildins a simulation of an iSBC86/12
hardware and running the validation testprogram
"VALCMD.C" on the simulated hardware(quotation marks
mark the actual commands)
--------------------------------------------------------

1) SIMULATION BUILDING:

a) Compile hardware modules:
                  -"set def 86val"(to go into validation
                                   directory)
                   "ic *.isp"(do it one file by one)
b) Software development:
                  -"set def c86"
                   "@cc8086 valcmd"(done in the "c86"
```

```
                                    directory)
                    "copy valcmd.s as86"
                    "set def as86"
                    "86asmotol valcmd"(done in "as86"
                                       directory)
                    "copy valcmd.out 86val"
                    "set def 86val"
                    "copy valcmd.out romcore.;"
c) Integration of hardware and software:
                   -"ec val"(ec prompt:Hardware, Software
                             or Both?)
                    "b"(for integrating both new hardware
                        and software;this is equivalent
                        to:-"ecologist val"
                           -"smp val"
                       )

2) RUNNING THE SIMULATION:

a) Clear simulated memories:
                   -"smp val"
b) Put simulation into runtime mode:
                   -"run val"
c) Initialization of the 8086 CPU:
                   -"deposit 1024 :ip"(the code block
                                       starts at 400h=1024)
                   -"deposit 0xab :ds"
                    "deposit 0xab :ss"(the data block
                                       starts at ab0h)
                   -"deposit 264 :sp"(sets the Top of the
                                      stack to 3000=16*ss + sp)

d) Simulation observation;I/O: The program displays its
   results automatically as mentioned above.Compare
   the results to the ones obtained by running "CMD.C"
   or "HEXCMD.C" on the VAX or on the real Intel 86/12.
   They are identical if the same angle was chosen as
   an input. The angle has to be input as a number of
   two decimal digits. The "IN" function picks the first
   two decimal numbers if a string of ASCII characters
   is given to it as an input.The accepted input will
   be echoed in hex.The "PRINT" function also prints
   the results of the simulation in hex.

e) Simulation control: -"run" to start or continue
                       -"Control C" to stop
                       -"e" to exit
```

THE DEVELOPMENT OF THE "SIMPLE SPACE ATTITUDE CONTROL"(SACL)
ALGORITHM

1) Run "SACLZ.FOR"(real variables):
   -"run saclz"
   -check results in file "SACLZ.DAT"
2) Run "SACLZI.FOR"(inteser varibles,scaled):
   -"run saclzi"
   -compare results in "SACLZI.DAT" to
    those in "SACLZ.DAT";they are
    equivalent if scaling factors and
    roundins errors from the inteser
    representation are taken into account
3) Run "CMD.C"(translation of "SACLZI.FOR" into C):
   -"run CMD"
   -compare results to those of
    the validation simulation
   -"run HEXCMD"
   -same as "CMD" but outrut
    printed in hex numbers
   -"CMD.C" was also run on the actual
    Intel SBC hardware
   -replacing the standard C I/O functions
    ("SCANF","PRINTF") with calls to special
    routines("IN","PRINT") using N.mFc's
    "Raw Memory" feature for I/O makes "CMD.C"
    the validation test program "VALCMD.C"

$

APPENDIX D

2) Performance Test Execution Procedure for Simulated Intel SBC Hardware

```
   In order to set a measurable simulation
   time on the VAX the "vaxsieve" version executes
   100'000 times in the 1..15 range.The same version
   may also be run on a real 86/12 machine("86sieve").
3) Simulation Use: -"smp val"
                   -"run val"
4) Initialization: -"deposit 1024 :ip"
                   -"deposit 0x61 :ds"
                   -"deposit 0x61 :ss"
                   -"deposit 0xf0 :sp"
5) Simulation Observation: Time start and stop of
   the simulation run by using the "Control t" fa-
   cility before and after the run.The difference in
   CPU time is the simulation execution time for one
   iteration of the "sieve" program in the 1..15
   number  range.A breakpoint on the last instruction
   executed("bkpt :ir eql 0xcb00") halts the simulation
   after program execution.
6) Simulation Control: -"run" to start or continue
                       -"Control C" to stop
                       -"q" to exit
7) Results: The simulated 86/12 takes about 400 seconds
            to execute the "sieve" benchmark once.
=============================================================

*************************************************************
* RESULTS OF THE PERFORMANCE COMPARISON BETWEEN VAX 11-780;*
* iSBC8612 AND A SIMULATION OF THE iSBC8612(running on VAX *
* 11-780) USING A "C" PROGRAM AS A BENCHMARK("SIEVE")      *
*************************************************************
*                                                           *
* NB: "P" means performance;the performance is inversely    *
*     proportional to the time needed to execute the bench-*
*     mark program a certain number of times on a certain   *
*     processor.                                            *
*************************************************************
*                                                           *
*         P(VAX) : P(8612) : P(8612 simulation)             *
*                                                           *
*             corresponds approximately to                  *
*                                                           *
*              2'364'000 : 190'000 : 1                      *
*                                                           *
*************************************************************
$
```

****
* PERFORMANCE COMPARISON BETWEEN A VAX 11-780, AN*
* iSBC86/12 AND A 86/12 SIMULATION USING THE *
* 'SIEVE' BENCHMARK PROGRAM *
****

A) Running 'sieve' 100;000 times on the VAX 11-780
++

- use 'control t' to set present CPU time

- 'run vaxsieve'

- check CPU time again;difference to first measurement is time spent by VAX 11-780 to execute the 'sieve' benchmark 100'000 times

- the CPU time needed by the VAX 11-780 for 100,000 sieve executions is roughly 17 seconds

++

B) Running 'sieve' 100;000 times on an iSBC86/12

For this test an iSBC86/12 equipped with a 'C' Cross Compiler is necessary.The testrun was therefore done at Communications Research Centre, Ottawa, in cooperation with Michel Savoie.The following result was obtained when running 'sieve' 100'000 times (using 'long integer' type variables in 'sieve'):

- 211 seconds

C) Running 'sieve' once on a Simulated iSBC86/12
==========

1) Simulated Hardware: Intel 86/12(described in detail in this report).
2) User Program: 'sieve' is a well known benchmark program.The version to be run on the 86/12 simulation('nmpcsieve') does just one execution of 'sieve' in the number range from 1..15 using 'long integer' variables in order to get a reasonable simulation time when running 'sieve' using N.mPc. The same program can be run on the VAX.

APPENDIX E

Printouts from Running the Performance Testprogram on Simulated Intel SBC Hardware

```
smp val
$ run val
        Welcome to N.mPc/VMS
N.mPc: val
# dep 1024 :ip
# dep 0x61 :ss
# dep 0x61 :ds
# dep 0xf0 :sp
# bkpt :ir eql 0xcb00
breakpoint number 1
# run
MAX 14:05:32 VAL        CPU=00:22:49.13 PF=9716 IO=15881 MEM=243
run
simulation halted by bkpt 1
(bkpt :ir eql 0xcb00)
#
MAX 14:24:28 VAL        CPU=00:29:17.95 PF=9806 IO=15884 MEM=317
a
$
$
$
$
$
```

```
$
$
$
$
$
$ smp val
$ run val
        Welcome to N.mPc/VMS
N.mPc: val
# dep 1024 :ip
# dep 0x61 :ss
# dep 0x61 :ds
# dep  0xf0 :sp
# bkpt :ir eql 0xcb00
breakpoint number 1
# run
MAX 14:26:10 VAL        CPU=00:29:20.02 PF=10178 IO=16253 MEM=314
run
simulation halted by bkpt 1
(bkpt :ir eql 0xcb00)
#
#
MAX 16:04:23 VAL        CPU=00:35:53.72 PF=10208 IO=16256 MEM=372
@
$

   smp val
 $ run val
         Welcome to N.mPc/VMS
 N.mPc: val
 # dep 1024 :ip
 # dep 0x61 :ss
 # dep 0x61 :ds
 # dep 0xf0 :sp
 # bkpt :ir eql 0xcb00
 breakpoint number 1
 # run
 MAX 14:25:16 VAL        CPU=00:54:42.01 PF=6035 IO=24147 MEM=308
 run
 simulation halted by bkpt 1
 (bkpt :ir eql 0xcb00)
 #
 MAX 14:36:18 VAL        CPU=01:01:26.93 PF=6063 IO=24150 MEM=373
```

```
$ link vaxsieve
$ run vaxsieve
MAX 10:17:13   (DCL)   CPU=00:14:16.66 PF=6792 IO=10200 MEM=65
$ run vaxsieve
100000 iterations

10 primes.
$
MAX 10:17:34   (DCL)   CPU=00:14:33.22 PF=6900 IO=10211 MEM=82
$
$
$
$ run vaxsieve
MAX 10:18:32   (DCL)   CPU=00:14:33.23 PF=6903 IO=10214 MEM=82
$ run vaxsieve
100000 iterations

10 primes.
$
MAX 10:18:51   (DCL)   CPU=00:14:49.77 PF=7013 IO=10225 MEM=82
$
$
$
$ run vaxsieve
MAX 10:19:07   (DCL)   CPU=00:14:49.81 PF=7016 IO=10228 MEM=82
$ run vaxsieve
100000 iterations

10 primes.
$
MAX 10:19:29   (DCL)   CPU=00:15:06.40 PF=7126 IO=10239 MEM=82
$
```

APPENDIX F

Listing of the Topology File for the Simulation of the 86/12 SBC

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                                                                        !
!   Topology file for the simulation of an Intel                         !
!   86/12 board featuring an external memory, a                          !
!   multibus interface(arbiter) a dualport RAM,                          !
!   a Programmable Interrupt Controller(PIC) and a                       !
!   global memory.These are all the elements                             !
!   planned to be implemented for a validation of                        !
!   N.mPc by this 86/12 simulation.                                      !
!                                                                        !
!   Max Streit, Intellitech Canada Ltd, May 84                           !
!                                                                        !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!                                                                        !
!   Memory map for the 86/12 :                                           !
!                                                                        !
!   - RAM(seen by processor):0...1023                                    !
!                                                                        !
!   - (private) ROM:1024...14335                                         !
!                                                                        !
!   - Multibus:14336...16383 is shared by:                               !
!                                                                        !
!         -Global Memory:14336...15359                                   !
!         -RAM(seen via Multibus):15360...16383                          !
!                                                                        !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

        signal
                M_rd,
                Mbus(16),
                M_hnible(4),
                Ale,
                Ready,
                Nmi,
                Int,
                Inta,
                L_den,
                L_bhe,
                Dt_r,
                Reset,
                Test,
                Lock,
                Status(3),
                Iorc,       !used for reading/writing the PIC
                Iowc,       !registers

                Ioads7,     !demultiplexed I/O addresses A0,A1
                Ioads1,     !used in PIC
```

```
! PIC signals

                Irq0,
                Irq1,
                Irq2,
                Irq3,
                Irq4,
                Irq5,
                Irq6,
                Irq7,

! Multibus signals

                Mad(20),
                Mda(16),
                Mrdc,
                Mwtc,
                Male,
                Xack,
                Busy,
                Bprn,
                Bpro,
                Bhen;

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        processor       cpu = "max86cpu.sim" ;
        time delay 250ns;

        connections
                        ! I/O signals used
                        ! to access the PIC
                        ! registers:
                          ioads7   =       Ioads7,
                          ioads1   =       Ioads1,

                           iorc    =       Iorc,
                           iowc    =       Iowc,

                          ale      =       Ale,
                          rd_      =       M_rd,
                          mbus     =       Mbus,
                          m_hnible=        M_hnible,
                          ready    =       Ready,
                          NMI      =       Nmi,
                          int      =       Int,
```

```
                        inta    =       Inta,
                        den_    =       L_den,
                        bhe_    =       L_bhe,
                        dt_r    =       Dt_r,
                        reset   =       Reset,
                        test    =       Test,
                        lock    =       Lock,
                        status  =       Status;


!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        processor       rom = "max86mem.sim" ;
        time delay 250ns ;

        connections     ale     =       Ale,
                        rd_     =       M_rd,

                        mbus    =       Mbus,
                        m_hnible=       M_hnible,
                        ready   =       Ready,
                        den_    =       L_den,
                        bhe_    =       L_bhe,
                        dt_r    =       Dt_r,
                        status  =       Status;

      initial           mem86   = romcore ;

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        processor       term    = "terminal.obj" ;
        time delay 250ns ;

        connections     ale     =       Ale,
                        rd_     =       M_rd,

                        mbus    =       Mbus,
                        m_hnible=       M_hnible,
                        ready   =       Ready,
                        den_    =       L_den,
                        bhe_    =       L_bhe,
                        dt_r    =       Dt_r,
                        status  =       Status;

      initial           tty     = ( wtty ) ;
                      ! the VMS designation of the
                      ! working terminal is "tt:",
                      ! a name not accepted by the
                      ! ecologist;a logical name ac-
                      ! ceptable to the ecol. has to
```

```
                        ! be defined in the login file;
                        ! (def wtty tt: in logsys.com)

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      processor          int = "multint86.obj";

      time delay         250ns ;

      connections
                         ! Internal bus connections

                          mbus      =        Mbus,
                          rd_       =        M_rd,
                          m_hnible=          M_hnible,
                          status =           Status,
                          ale       =        Ale,
                          ready     =        Ready,
                          bhe_      =        L_bhe,
                          den_      =        L_den,
                          dt_r      =        Dt_r,

                         ! Multibus connections

                          mad       =        Mad,
                          mda       =        Mda,
                          mrdc      =        Mrdc,
                          mwtc      =        Mwtc,
                          iorc      =        Iorc,
                          iowc      =        Iowc,
                          male      =        Male,
                          xack      =        Xack,
                          busy      =        Busy,
                          bprn      =        Bprn,
                          bpro      =        Bpro,
                          bhen      =        Bhen;


!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        processor        gmem = "globalmem.obj";

        time delay       250ns ;

        connections
                          bhen      =        Bhen,
                          mad       =        Mad,
                          mda       =        Mda,
```

```
                        male      =        Male,
                        xack      =        Xack,
                        mrdc      =        Mrdc,
                        mwtc      =        Mwtc;


        initial         me       = sblcore;
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        processor       ram = 'dpram.obj' ;

        time delay      250ns ;

        connections     !CPU ports
                        ale        =       Ale,
                        rd_        =       M_rd,
                        mbus       =       Mbus,
                        m_hnible=          M_hnible,
                        ready      =       Ready,
                        den_       =       L_den,
                        bhe_       =       L_bhe,
                        dt_r       =       Dt_r,
                        status     =       Status,
                        lock       =       Lock,
                        !Multibus ports
                        bhen      =        Bhen,
                        mad       =        Mad,
                        mda       =        Mda,
                        male      =        Male,
                        xack      =        Xack,
                        mrdc      =        Mrdc,
                        mwtc      =        Mwtc;
      initial           me       = ramcore;

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

processor               pic    =   'pic.obj';

time delay              250ns;

connections             IRQ0      =       Irq0,
                        IRQ1      =       Irq1,
                        IRQ2      =       Irq2,
                        IRQ3      =       Irq3,
                        IRQ4      =       Irq4,
                        IRQ5      =       Irq5,
```

```
                        IRQ6        =         Irq6,
                        IRQ7        =         Irq7,

                     Databus        =         Mbus,

                          RD        =         Iorc,
                          WR        =         Iowc,
                          CS        =         Ioads7,
                          A0        =         Ioads1,
                         INT        =         Int,
                        INTA        =         Inta;
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

processor             inter         = 'interrupt.obj';

time delay            250ns;

connections           IRQ0          = Irq0,
                      IRQ1          = Irq1,
                      IRQ2          = Irq2,
                      IRQ3          = Irq3,
                      IRQ4          = Irq4,
                      IRQ5          = Irq5,
                      IRQ6          = Irq6,
                      IRQ7          = Irq7;
$
```

APPENDIX G

The "OTOL" Program

NAME: otol -Object File to l.out format transformation

SYNOPSIS:

otol -machine[{d|dx}] input_file [output_file]
[-ainitrecord_1] [-ainitrecord_2] ..

DESCRIPTION:

This program transforms the object file of several machines to l.out format. The user has to specify a minimum of two things.

1) machine name.
2) Input object file.

The machine should be specified with one of the following characters.

m - for the MOTOROLA data file
i - for the INTEL INTELLEC 8/MDS data file.
t - for the TEKTRONIX HEXADECIMAL data file.
r - for the RCA COSMAC data file.
o - for the MOS Technology data file.
s - for the SIGNETICS ABSOLUTE OBJECT data file.
f - for the FAIRCHILD FAIRBUG data file.

The option can be specified in the following manner.

d - Put the debug option on. The program will output the data record address range to the standard output in decimal value.
x - This is a modifier to 'd' option. 'dx' will output the address range in hexadecimal value.

The input file is the file containing the object data file of the machine. This is essential for the program to work.

The output file is the output file which will have the l.out format transformation. By default, a filename 'l.out' will be used.

Init Record

The program will fill up the specified range of memory with the initial value and will give a warning message if this range overlaps any data records. The User can specify more than one initrecord and in more than one way. The init record can be specified as follows.

1) -aloweraddress-higheraddress$initvalue

2) -a%-higheraddress$initvalue
In this case the lower limit of the range is implicit. The program will fill up location from the highest address of the user program to the specified higheraddress limit with the initvalue.

3) -aloweraddress-%$initvalue
In this case the higher limit of the range is implicit. The program will fill up location from the specified loweraddress to the lowest address of the user program with the initvalue.

The value can be specified in octal, decimal or hex format.
For octal - precede the number with 0
decimal - number is written as such
hex - precede the number with 0x

The program scans the input file according to the machine specifications and produces the l.out or the specified output file. The check sum is checked after every line or data record of the object file gives warning message if its not correct. The l.out file has only those location filled up which are specified in the object file. So the gaps can occur in the memory and Runtime will give an error message if the User program tries to access any of those locations. The User can use the initrecord facility to overcome this problem.

AUTHOR

Kaushik Sheth
Case Western Reserve University, Cleveland. Oh 44106.

EXAMPLE

```
otol -m test.l
```

The program will take the input file test.l and scan through it assuming the machine is MOTOROLA. The output file is l.out.

```
otol -idx test.l test.core -a%-0x3000$25 -a0-%$0 -a400-500$0
```

The program will take the input file test.l. scan through it assuming the machine is INTEL, output file is test.core. It will fill up the memory location as follows.

1) highest location of program to 0x3000 with 25 value.
2) 0 to lowest location of program with 0 value.
3) 400 to 500 with 0 value. It will overwrite the values in those locations if program has written some thing to it.

The program will also output the memory map in the hex-range value.

SEE ALSO

mdump -- program to see the memory locations and symbols
sme -- program to edit the memory location.

BUGS

Only INTEL and MOTOROLA program are checked at this moment. The User might have to type '\$' in the case of init records if the operating system does not allow to pass '$' as it is to the program. If you find any thing else let us know !!

APPENDIX H

1) The Program Used To Test the Simulated 8086 CPU

```
cat test.86
```

```
;*************************************************
;*                                               *
;*    Test program for single step testing of    *
;*    all the Intel 8086's instructions in va-   *
;*    rious addressing modes.The program is      *
;*    written in the as8086 assembler used by    *
;*    the Lantech Cross software tools.          *
;*    "TEST.86" is transferred into "l.out"      *
;*    format using the Lantech assembler,        *
;*    linker, loader and the "otol" program.     *
;*                                               *
;*************************************************
;*                                               *
;*    Max Streit, Intellitech Canada Ltd         *
;*    June 1984                                  *
;*                                               *
;*************************************************

; constants

data8:   .equ     238
data16:  .equ     11ddH
disp8:   .equ     11H
disp16:  .equ     0100H
fill:    .ds      20
addr:    .dw      0
addr16:  .dw      0
port:    .equ     00c2H
type:    .equ     7
;***************************************
seg16:   .segment         code

seg16:   .ends
;***************************************

; two datasegments for string instruction tests

datseg:  .segment         data
         .org 3072
a:       .db      1
b:       .db      2
c:       .db      3
d:       .db      4
e:       .db      5
f:       .db      6
g:       .db      7
```

```
h:      .db     8
i:      .db     9
j:      .db     10
k:      .db     11
l:      .db     12
m:      .db     13
n:      .db     14
o:      .db     15
p:      .db     16

datseg: .ends
*************************************
destseg: .segment       data

         .org 3328

q:       .dw    0001h
r:       .dw    0302h
s:       .dw    0504h
t:       .dw    0706h
u:       .dw    0a09h
v:       .dw    257
w:       .dw    257
x:       .dw    257

destseg:  .ends
*************************************
* This segment contains all the 8086 *
* instructions                       *
*************************************
tseg:   .segment        code

* the ROM portion of the 86/12 memory
* starts at 1024dec
        .org 1024

* every possible way of data shuffling
* is tested by MOV instructions;this
* tests the correctness of all register
* and memory transfers

        mov     [bx+si],di
        mov     [bx+di],di
        mov     [bp+si],di
        mov     [bp+di],di
        mov     [si],di
        mov     [di],di
        mov     [bp],di
```

```
        mov     [bx],di
        mov     disp8[bx,si],di
        mov     disp8[bx,di],di
        mov     disp8[bp,si],di
        mov     disp8[bp,di],di
        mov     disp8[si],di
        mov     disp8[di],di
        mov     disp8[bp],di

        mov     disp8[bx],di
        mov     disp16[bx,si],di
        mov     disp16[bx,di],di
        mov     disp16[bp,si],di
        mov     disp16[bp,di],di
        mov     disp16[si],di
        mov     disp16[di],di
        mov     disp16[bp],di
        mov     disp16[bx],di
        mov     addr,di
*******
        mov     di,[bx,si]
        mov     di,[bx,di]
        mov     di,[bp,si]
        mov     di,[bp,di]
        mov     di,[si]
        mov     di,[di]
        mov     di,[bp]
        mov     di,[bx]
        mov     di,disp8[bx,si]
        mov     di,disp8[bx,di]
        mov     di,disp8[bp,si]
        mov     di,disp8[bp,di]
        mov     di,disp8[si]
        mov     di,disp8[di]
        mov     di,disp8[bp]
        mov     di,disp8[bx]
        mov     di,disp16[bx,si]
        mov     di,disp16[bx,di]
        mov     di,disp16[bp,si]
        mov     di,disp16[bp,di]
        mov     di,disp16[si]
        mov     di,disp16[di]
        mov     di,disp16[bp]
        mov     di,disp16[bx]
        mov     di,addr
**************************************
        mov     [bx,si],#data8
        mov     [bx,di],#data8
```

```
        mov     [bp,si],#data8
        mov     [bp,di],#data8
        mov     [si],#data8
        mov     [di],#data8
        mov     [bp],#data8
        mov     [bx],#data8
        mov     disp8[bx,si],#data8
        mov     disp8[bx,di],#data8

        mov     disp8[bp,si],#data8
        mov     disp8[bp,di],#data8
        mov     disp8[si],#data8
        mov     disp8[di],#data8
        mov     disp8[bp],#data8
        mov     disp8[bx],#data8
        mov     disp16[bx,si],#data8
        mov     disp16[bx,di],#data8
        mov     disp16[bp,si],#data8
        mov     disp16[bp,di],#data8
        mov     disp16[si],#data8
        mov     disp16[di],#data8
        mov     disp16[bp],#data8
        mov     disp16[bx],#data8
        mov     addr,#data8
**********************************
        mov     [bx,si],#data16
        mov     [bx,di],#data16
        mov     [bp,si],#data16
        mov     [bp,di],#data16
        mov     [si],#data16
        mov     [di],#data16
        mov     [bp],#data16
        mov     [bx],#data16
        mov     disp8[bx,si],#data16
        mov     disp8[bx,di],#data16
        mov     disp8[bp,si],#data16
        mov     disp8[bp,di],#data16
        mov     disp8[si],#data16
        mov     disp8[di],#data16
        mov     disp8[bp],#data16
        mov     disp8[bx],#data16
        mov     disp16[bx,si],#data16
        mov     disp16[bx,di],#data16
        mov     disp16[bp,si],#data16
        mov     disp16[bp,di],#data16
        mov     disp16[si],#data16
        mov     disp16[di],#data16
        mov     disp16[bp],#data16
```

```
        mov     disp16[bx],#data16
        mov     addr,#data16
*******
        mov     b'[bx,si],#data8
        mov     b'[bx,di],#data8
        mov     b'[bp,si],#data8
        mov     b'[bp,di],#data8
        mov     b'[si],#data8

        mov     b'[di],#data8
        mov     b'[bp],#data8
        mov     b'[bx],#data8
        mov     b'disp8[bx,si],#data8
        mov     b'disp8[bx,di],#data8
        mov     b'disp8[bp,si],#data8
        mov     b'disp8[bp,di],#data8
        mov     b'disp8[si],#data8
        mov     b'disp8[di],#data8
        mov     b'disp8[bp],#data8
        mov     b'disp8[bx],#data8
        mov     b'disp16[bx,si],#data8
        mov     b'disp16[bx,di],#data8
        mov     b'disp16[bp,si],#data8
        mov     b'disp16[bp,di],#data8
        mov     b'disp16[si],#data8
        mov     b'disp16[di],#data8
        mov     b'disp16[bp],#data8
        mov     b'disp16[bx],#data8
        mov     b'addr,#data8
**********************************
arops:  jmp jmp3
        aaa
        aad
        aam
        aas
        adc     [bx,si],#data16
        adc     [bx,si],#data8
        adc     [bx,si],#data8
        adc     [bx,si],#data8
        adc     [bx,si],di
        adc     al,#data8
        adc     ax,#data16
        adc     b'[bx,si],#data8
        adc     b'[bx,si],bh
        adc     bh,[bx,si]
        adc     di,[bx,si]
        add     [bx,si],#data16
        add     [bx,si],#data8
```

```
        add     [bx,si],#data8
        add     [bx,si],#data8
        add     [bx,si],di
        add     al,#data8
        add     ax,#data16
        add     b'[bx,si],#data8
        add     b'[bx,si],bh
        add     bh,[bx,si]

        and     [bx,si],#data16
        and     [bx,si],#data8
        and     [bx,si],di
        and     al,#data8
        and     ax,#data16
        and     b'[bx,si],#data8
        and     b'[bx,si],bh
        and     bh,[bx,si]
        and     di,[bx,si]
**********************************
* test of procedure calls
        call    proc1
        calli   [bx,si]
        calll   proc3,tses
        callli  [bx,si]
**********************************
        cbw
        clc
        cld
        cli
        cmc
**********************************
* target procedures for calls
proc1:  mov     ax,#1
        ret
proc2:  mov     ax,#2
        ret     #data16
proc3:  mov     ax,#3
        retl
proc4:  mov     ax,#4
        retl    #data16
**********************************
        cmp     [bx,si],#data16
        cmp     [bx,si],#data8
        cmp     [bx,si],#data8
        cmp     [bx,si],#data8
        cmp     [bx,si],di
        cmp     al,#data8
        cmp     ax,#data16
```

```
        cmp     b'[bx,si],#data8
        cmp     b'[bx,si],bh
        cmp     bh,[bx,si]
        cmp     di,[bx,si]
        repe
        cmpsb
        repe
        cmpsw
        repne
        cmpsb
        repne
        cmpsw
        cwd
        daa
        das
        dec     [bx,si]
        dec     ax
        dec     b'[bx,si]
        dec     bp
        dec     bx
        dec     cx
        dec     di
        dec     dx
        dec     si
        dec     sp
        div     [bx,si]
        div     b'[bx,si]
*       esc     [bx,si]         ;not implemented
        hlt

        idiv    [bx,si]
        idiv    b'[bx,si]
        imul    [bx,si]
        imul    b'[bx,si]
*       in      al,dx           ; (use inb)
        in      al,#port
*       in      ax,dx           ; (use inw)
        in      ax,#port
        inb
        inc     [bx,si]
        inc     ax
        inc     b'[bx,si]
        inc     bp
        inc     bx
        inc     cx
        inc     di
        inc     dx
        inc     si
```

```
        inc     sp
**************************************
* software interrupts
        int     3                   ;use int3
        int     type
        int3
        into
        inw
        iret
*****************************************************
* test for conditional control transfer instructions

11:     cmp     ax,bx
        ja      12
13:     cmp     ax,bx
        jae     14
15:     cmp     ax,bx
        jb      16
17:     cmp     ax,bx
        jbe     18
19:     cmp     ax,bx
        jc      110
111:     cmp     ax,bx
        jcxz    112
113:    cmp     ax,bx
        je      114
115:    cmp     ax,bx
        js      116
117:    cmp     ax,bx
        jge     118
119:    cmp     ax,bx
        jl      120
121:    cmp     ax,bx
        jle     122
133:    cmp     ax,bx
        jna     124


12:     cmp     ax,bx
        jnae    13
14:     cmp     ax,bx
        jnb     15
16:     cmp     ax,bx
        jnbe    17
18:     cmp     ax,bx
        jnc     19
110:    cmp     ax,bx
        jne     111
```

```
112:    cmp     ax,bx
        jns     113
114:    cmp     ax,bx
        jnge    115
116:    cmp     ax,bx
        jnl     117
118:    cmp     ax,bx
        jnle    119
120:    cmp     ax,bx
        jno     121
122:    cmp     ax,bx
        jnp     133
124:    cmp     ax,bx
        jns     124
126:    cmp     ax,bx
        jnz     126
128:    cmp     ax,bx
        jo      128
130:    cmp     ax,bx
        jp      130
132:    cmp     ax,bx
        jpe     132
134:    cmp     ax,bx
        jpo     136
136:    cmp     ax,bx
        js      136
138:    cmp     ax,bx
        jz      138
*****************************************
* unconditional control transfer instructions

jmp0:   jmp     jmp2


jmp1:
        jmp     jmp0
jmp2:
        jmp     arops
jmp3:
        jmpi    [bx,si]
jmp4:
        jmpl    data16,data16
jmp5:
        jmpli   [bx,si]
*************************************

        lahf
        lds     di,[bx,si]
```

```
        lea     di,[bx,si]
        les     di,[bx,si]
        lock
        lodsb
        lodsw
************************************************
* loop instructions

lo1:    loop    lo1

lo2:    cmp ax,bx
        loope   lo2
lo3:    cmp ax,bx
        loopne  lo3
lo4:    cmp ax,bx
        loopnz  lo4
lo5:    cmp ax,bx
        loopz   lo5

****************************************
        mov     [bx,si],#data16
        mov     [bx,si],#data8
        mov     [bx,si],di
        mov     b'addr16,al
        mov     addr16,ax
        mov     ah,#data8
        mov     al,#data8
        mov     al,addr16
        mov     ax,#data16
        mov     ax,addr16
        mov     b'[bx,si],#data8
        mov     b'[bx,si],bh
        mov     bh,#data8
        mov     bh,[bx,si]
        mov     bl,#data8
        mov     bp,#data16
        mov     bx,#data16
        mov     ch,#data8
        mov     cl,#data8
        mov     cx,#data16

        mov     di,#data16
        mov     di,[bx,si]
        mov     dl,#data8
        mov     dx,#data16
        mov     si,#data16
        mov     sp,#data16
        movsb
```

```
        movsd   [bx,si],ss
        movsd   ss,[bx,si]
        movsw
        mul     [bx,si]
        mul     b'[bx,si]
        neg     [bx,si]
        neg     b'[bx,si]
        nop
        not     [bx,si]
        not     b'[bx,si]
        or      [bx,si],#data16
        or      [bx,si],#data8
        or      [bx,si],di
        or      al,#data8
        or      ax,#data16
        or      b'[bx,si],#data8
        or      b'[bx,si],bh
        or      bh,[bx,si]
        or      di,[bx,si]
*       out     dx,al               ;use outb
*       out     dx,ax               ;use outw
        out     al,#port
        out     ax,#port
        outb
        outw
        pop     [bx,si]
        pop     ax
        pop     bp
        pop     bx
        pop     cx
        pop     di
        pop     dx
        pop     si
        pop     sp
        popf
        popsd   ds
        popsd   es
        popsd   ss
        push    [bx,si]
        push    ax

        push    bp
        push    bx
        push    cx
        push    di
        push    dx
        push    si
        push    sp
```

```
pushf
pushss  cs
pushss  ds
pushss  es
pushss  ss
rcl     [bx,si]
rcl     b'[bx,si]
rclv    [bx,si]
rclv    b'[bx,si]
rcr     [bx,si]
rcr     b'[bx,si]
rcrv    [bx,si]
rcrv    b'[bx,si]
rep
repe
repne
repnz
repz
ret
ret     #data16
retl
retl    #data16
rol     [bx,si]
rol     b'[bx,si]
rolv    [bx,si]
rolv    b'[bx,si]
ror     [bx,si]
ror     b'[bx,si]
rorv    [bx,si]
rorv    b'[bx,si]
sahf
sar     [bx,si]
sar     b'[bx,si]
sarv    [bx,si]
sarv    b'[bx,si]
sbb     [bx,si],#data16
sbb     [bx,si],#data8
sbb     [bx,si],#data8
sbb     [bx,si],#data8

sbb     al,#data8
sbb     ax,#data16
sbb     b'[bx,si],#data8
sbb     b'[bx,si],bh
sbb     bh,[bx,si]
sbb     di,[bx,si]
repe
scasb
```

```
repe
scasw
repne
scasb
repne
scasw
segcs
segds
seges
segss
shl     [bx,si]
shl     b'[bx,si]
shlv    [bx,si]
shlv    b'[bx,si]
shr     [bx,si]
shr     b'[bx,si]
shrv    [bx,si]
shrv    b'[bx,si]
stc
std
sti
repe
stosb
repe
stosw
repne
stosb
repne
stosw
sub     [bx,si],#data16
sub     [bx,si],#data8
sub     [bx,si],#data8
sub     [bx,si],#data8
sub     [bx,si],di
sub     al,#data8
sub     ax,#data16
sub     b'[bx,si],#data8
sub     b'[bx,si],bh
sub     bh,[bx,si]
sub     di,[bx,si]
test    [bx,si],#data16
test    [bx,si],di
test    al,#data8
test    ax,#data16
test    b'[bx,si],#data8
test    b'[bx,si],bh
wait
xchg    ax,ax
```

```
        xchg    ax,bp
        xchg    ax,bx
        xchg    ax,cx

        xchg    ax,di
        xchg    ax,dx
        xchg    ax,si
        xchg    ax,sp
        xchg    bh,[bx,si]
        xchg    di,[bx,si]
        xlat
        xor     [bx,si],#data16
        xor     [bx,si],#data8
        xor     [bx,si],di
        xor     al,#data8
        xor     ax,#data16
        xor     b'[bx,si],#data8
        xor     b'[bx,si],bh
        xor     bh,[bx,si]
        xor     di,[bx,si]
tses:   .ends

        .eject
        .end
$
```

APPENDIX H

2) List of "Bugs" fixed in the initial 8086 CPU Description

```
!*****************************************************!
!   List of "Buss" in the Trivedi version of the 8086 fixed  !
!   by M. Streit:                                     !
!              1) introduction of the word/byte case for  !
!                 the LDA and STA instructions            !
!              2) active high logic for the ready signal  !
!                 (wait if ready = high) because two mo-  !
!                 dules use this input signal(N.mPc "ORs" !
!                 all ports writing to a signal)          !
!              3) a delay of 2 units introduced between   !
!                 the two memory accesses for read/write  !
!                 of a word from/to an odd address to allow !
!                 private or global memory to finish its  !
!                 first read/write operation before the sub-!
!                 sequent one starts                      !
!              4) a missing "next;" after the IO = Output; !
!                 statement in the "do_output" procedure  !
!                 disabled all output operations and had to !
!                 be added                                !
!              5) offsets in Jumps were 1 unit too short be-!
!                 cause improper Jump execution(ip=ip+setQ) !
!                 where the last ip incrementation took pla-!
!                 ce in the setQ procedure but didn't affect!
!                 the Jump;step by step execution of the  !
!                 Jump(iptemp=setQ;next;ip=ip+iptemp;next;) !
!                 now perform a Jump using the proper offset!
!                 The linking loader program also had to be !
!                 adjusted(see i80861.i in llcf directory). !
!              6) the effective address calculation was   !
!                 wrongly encoded in the case of [bp,si],di; !
!                 bx was mixed up with bp(in case r_m = 02) !
!              7) the number Fhex=15dec was wrongly repre- !
!                 sented as 0x15=15hex(in ISP')=21dec in  !
!                 the DAA and AAA(2 times) instructions   !
!              8) in all add- or subtract carry instructions!
!                 the carry flag(cf) was added without ex- !
!                 sion to a word and was therefore misin-  !
!                 terpreted as -1;cf had to be replaced by !
!                 (cf ext 16) in the ADC,AIC,SBC,SAI,ADCI, !
!                 and SBB instructions                    !
!              9) improper use of setQ by the "setimd_sw" !
!                 procedure in the ADCI instruction fixed by!
!                 arranging the cases of ir<9:8> correctly !
!             10) decoding and procedure for the ADC      !
!                 reg/mem, reg instruction had to be added !
!                 (op1 = ir<15:9> = 010octal);same for ADD !
!                 reg/mem, reg (op1 = 0)                  !
!             11) similar changes as on ADD/ADC had to be  !
```

performed on SUB/SBB(subtract with borrow)

12) incorrect quotient range criterion(ax(16 bit) cannot be greater than FFFFh) replaced in the "div" procedure; divisor read corrected to reflect the word/byte possibilitie for unsigned division

13) bug in "idiv":reg32<31:16> = dx; (and not <31:15>

14) software interrupt type0 also implemented in "idiv" and "div" for attempts to divide by zero

15) the two instructions "mul" and "div" expect unsigned operands;as sign extension occurs when byte operands are acquired this problem had to be fixed by avoiding byte operand acquisition

16) byte- and word-operand case were not correctly treated in "mul" and "imul"

17) the mem/reg destination case had to be added in the AND, OR and XOR instructions

18) the implicite count register for all the shifts and rotates is cl(=cx<lobyte>) and not dx!

19) rotating through carry had to be fixed in all instructions using it

20) all shifts and rotates only took care of word operations;the byte case had to be introduced

21) inadequate size variable(reg3, 2 bit wide instead of 3) spoilt parameter passing in the PUSHS instructions;extension of reg3 to 3 fixed the problem

22) improper acquisition of the offset operand in the LES instruction fixed

23) the "storereg" procedure improperly stored signextended bytes instead of bytes only into the registers,thereby affecting also the one half of a 16 bit register that should remain unaffected by a byte transfer

24) WAIT changed from "if TEST inactive" to "while TEST inactive"; wait_ff is reset as soon as test becomes active(see 8086 Hardware Manual, pg.2-18)

25) the opcode for the "SCAS" instruction was not put into the decoding table and therefore not decoded

26) incrementing and decrementing of the data-
indexes in the string instructions was in-
as a one bit constant(wrd1) is interpreted
as 0 or -1 by ISP';extension to word size
solves the problem

27) end of repetition in case of cx = 0 had to
be added in the SCAS and CMPS instructions

28) for correct execution of repeated string
instructions the "zeroflag" bit of the re-
peat prefixes had to be stored in a global
variable(repzf); at the same this prevents
the repeat prefixes from affecting the
flags

29) only the lower byte of an alu result
should affect the parity flag(pf);the tre-
atment of word results in the alu procedu-
had to be changed to take parity only of
lobyte of the result

30) the parity flag was stuck at 1; the reason
was that the predefined N.mPc procedure
"parity" always produced zeroes as output;
the problem was fixed by introducing an
internal parity procedure called "par"

31) the overflow flag was not correctly set
for subtractions(Ex:-3 - (-3) = 0 produ-
ced an overflow); a case by case treatment
for over flows had to be introduced for
addition and subtraction

32) the overflow determination wrongly used
carry bit instead of the sign bit in the
byte case of the alu procedure

33) JMP(intrasegment direct,long and short)
Jumped one byte too far because the jump
in these instructions was executed before
the ip was increased by the offset acqui-
sition(set0, set0_2)

34) the fifo_empty flag was not set in the JMP
(short) instruction;so the instruction
queue didn't set cleared after a short
Jump and wrong instructions got executed

35) the indirect control transfer instructions
(CASI,CISI,JAI,JII) were omitted in the
decoding table; as their decoding would
have coincided wth the one of PUSH the
latter's decoding had to be changed; too

36) the physical address calculation("b_alu"
procedure) did not perform the 4 bit posi-
tion left shift(= %16) of the segment base

value before its addition to the offset

37) intra segment calls/returns wrongly pushed ip and cs(instead of only ip)

38) all CALL and RET instructions were incorrect because improper sequence of assignments led to faulty control transfers

39) the type constant for software interrupts was only defined as a 3 bit constant, thus limiting software interrupt types to 0-3; softint was redefined to a byte size allo- for the 255 types of the 8086 to be executed

40) "next" statements were added to the INTERTERRUPT procedure to ensure that the flags are being pushed on the stack before any flags are altered

41) if the bp register is used as a base for calculation of the effective address the stack segment is referenced unless an override prefix specifies another segment register;this setting was introduced into the "setEA" procedure

42) variables (besides strings,bp) using the EA as an offset use the data segment unless an override segment specifies another segment; this override possibility was also introduced into "setEA"

43) the "mem_read" and "mem_write" procedures were only using the first 64 kBytes of the address space because m_hnible(bits 19:16 of the 20 bit address) was set to 0 at the beginning of both procedures;m_hnible is now set to address<19:16>

44) the BIU queue filling fetch operations did interfere with the readings of the types of external interrupts; a flag(intair = interrupt acknowledge in progress) had to be set up to prevent the BIU from corrupting the data sent by the PIC as well as the corresponding "while(intair) delay" in the BIU procedures

45) the FILL_FIFO BIU-procedure used the cs register to fill two bytes into the instruction queue; so a change of cs would immediately affect the FILL_FIFO procedure causing reading from wrong addresses as as the corresponding ip(stored globally in b_ip) would not be affected by the EU

changes; storing the initial cs in a another global variable,b_cs,solves the the problem

46) the CPI(compare immediate) instruction used wrd3(=ir<9>) instead of wrd1(=ir<8>) to determine word or byte operation; this bug happened only after several executions of the instruction and caused the N.mPc system to blow up and pass control back to the operating system(VMS) setting an ACCESS VIOLATION error message; N.mPc also didn't blow up in the compare instruction itself so that this kind of bug is very difficult to trace; this bug was found by varying the input assembler program in order to determine that caused the problem

47) the segment argument in the "setmem" procedure was reduced to a two bit size(from unappropriate WORD size;as well the segment argument was zeroextended to three bit size before being assigned to the 3bit EB_seg variable to prevent automatic sign by the system(thereby assigning faulty segment values to EB_seg); this sign extension was also introduced in "storemem"

48) the memory read calls in the "INTERRUPT" procedure used the argument "type * 4" to read the interrupt pointers; but type is defined as a byte and therefore sign extended by the "INTERRUPT" procedure which expects a word sized argument;this only created a problem for interrupt types higher than 31 because only at type = 32 the product 4*32=128 starts having a 1 in the 8th bit, which is interpreted as the sign of a negative number by the "INTERRUPT" procedure which then destrys the the correct type value by signextending it to 16 bits.(non sign) extension of the interrupt type before it is used by "INTERRUPT" solves the problem

49) The "CMP" instruction only worked in the "CMP reg,reg/mem" direction;the necessary code to do "CMP reg/mem,reg" was added in the arithmetic instruction's field under cases 034 and 035.Like in the case of "CPI"(see 46) N.mPc aborted the runtime environment catastrophically.The last in-

```
!                 struction observed before the abort occur-!
!                 red was the one executed two instructions !
!                 before the faulty instruction, indicating !
!                 a delay between execution inside N.mPc and!
!                 display of the executed instruction.So if !
!                 a catastrophical abort occurs one always  !
!                 to suspect the cause in an instruction    !
!                 slightly BEFORE the last one executed.    !
!              50) The "INCR" and "DECR" instructions did not!
!                 update the flags when incrementing/decre- !
!                 menting registers.This was fixed by using !
!                 the "alu" procedure for incrementing/dec- !
!                 rementing rather than just adding/subtrac-!
!                 ting 1.                                   !
!              51) The carry in the case of subtractions(cal-!
!                 led "borrow) was improperly set.The borrow!
!                 is the inverted carry except in the case  !
!                 of a subtraction of zero from some number !
!                 where the borrow, as well as the carry,   !
!                 is equal to 0.The carry for subtraction is!
!                 determined in the same way as the carry   !
!                 for addition but one adds the 2's comple- !
!                 of the number to be subtracted.The carry  !
!                 is then inverted except in the case of the!
!                 subtrahend being zero,where carry=borrow=0!
!              52) The "SHL" and "SAL" instructions did not  !
!                 set the carry with the bits shifted out   !
!                 leftwards.The carry flag now contains the !
!                 last bit shifted leftwards out of a regis-!
!                 ter.                                      !
!              53) An unwanted sign extension in the "MUL"   !
!                 instruction corrupted results for negative!
!                 numbers.Zero extension of multiplicand and!
!                 multiplicator to result size prior to the !
!                 multiplication solved the problem.        !
!              54) The conditional Jump instructions "JG",   !
!                 "JLE","JL","JNL" did not take the overflow!
!                 flag into account for their Jump conditi- !
!                 ons.Conditions were corrected according to!
!                 hardware manual for the Intel 8086.       !
!*************************************************************!
```

APPENDIX I

Listings of the I/O Assembly Routines used in the Validation Simulation ("PRINT.S", "IN.S")

```
            .public         _in
_in:    .segment        code
            .assume         cs:_in,ds:$common

            push        bp
            push        bx
            push        cx
*stack:
*   (sp+8): oldcs
*   (sp+6): oldip
*   (sp+4): bp
*   (sp+2): bx
*      sp-> cx

          mov         bp,sp       ;have bp point to
                                     ;bottom of stack

*****************************************
* INPUT ROUTINE FOR THE VALIDATION SIMULATION *
*****************************************
          mov       ax,#10   ;writes "Enter angle
          out       ax,#0    ;(degrees,2 digits):"
          mov       ax,#13
          out       ax,#0
          mov       ax,#69
          out       ax,#0
          mov       ax,#110
          out       ax,#0
          mov       ax,#116
          out       ax,#0
          mov       ax,#101
          out       ax,#0
          mov       ax,#114
          out       ax,#0
          mov       ax,#32
          out       ax,#0


          mov       ax,#97
          out       ax,#0
          mov       ax,#110
          out       ax,#0
          mov       ax,#103
          out       ax,#0
          mov       ax,#108
          out       ax,#0
```

```
        mov     ax,$101
        out     ax,$0
        mov     ax,$40
        out     ax,$0
        mov     ax,$100
        out     ax,$0
        mov     ax,$101
        out     ax,$0
        mov     ax,$103
        out     ax,$0
        mov     ax,$114
        out     ax,$0
        mov     ax,$101
        out     ax,$0
        mov     ax,$101
        out     ax,$0
        mov     ax,$115
        out     ax,$0
        mov     ax,$44
        out     ax,$0
        mov     ax,$50
        out     ax,$0
        mov     ax,$32
        out     ax,$0
        mov     ax,$100
        out     ax,$0
        mov     ax,$105
        out     ax,$0
        mov     ax,$103
        out     ax,$0
        mov     ax,$105
        out     ax,$0
        mov     ax,$116
        out     ax,$0
        mov     ax,$115
        out     ax,$0
        mov     ax,$41
        out     ax,$0
        mov     ax,$58
        out     ax,$0
        sub     ax,ax

NONUM:  in      ax,$0       ;the first two numbers in a
                            ;strins of characters are ta-
                            ;ken to be the input "angle"

        and     ax,$7fh     ;strips parity bit off
        cmp     ax,$57      ;prevents input of anything
```

```
          js        NONUM     ;but numbers 0-9
          cmp       ax,#48
          jb        NONUM
          out       ax,#0

          sub       ax,#48
          mov       bx,ax
          sub       ax,ax
          mov       ax,#10
          mul       bx
          mov       bx,ax
          sub       ax,ax

NONU:     in        ax,#0
          and       ax,#7fh   ;strips parity bit off
          cmp       ax,#57    ;prevents input of anything
          js        NONU      ;but numbers 0-9
          cmp       ax,#48
          jb        NONU
          out       ax,#0

          sub       ax,#48
          add       ax,bx     ;ax now contains "angle" to be
                              ;input
          mov       dx,#0     ;the function return returns a
                              ;doubleword in the dx,ax registers
                              ;the higher word(in dx) is always zero

END:        pop       cx
            pop       bx
            pop       bp

            retl

*****************************************
* THIS SUBROUTINE PRINTS A WORD(THE VALUE IN *
* THE AX REGISTER) IN HEX ON THE SCREEN OF   *
* THE WORKING TERMINAL;IT IS INVOKED TWICE TO*
* HANDLE 32 BIT VARIABLES;4 BIT GROUPS OF THE*
* NUMBER TO BE DISPLAYED ARE CONVERTED TO THE*
* CORRESPONDING ASCII CODE                   *
*****************************************
asciprt:
          mov       cx,ax   ;cx and ax registers
          mov       ah,ch   ;are used

          shr       ah      ;transform highest nibble
          shr       ah      ;of ax to hex(in ASCII)
```

```
            shr         ah
            shr         ah
            and         ah,#0fh
            cmp         ah,#9
            js          hex
            add         ah,#30h
            jmp         next
hex:       add         ah,#37h
next:       mov         al,ah
            out         al,#0   ;output highest nibble on
                                ;on screen
            mov         ah,ch
            and         ah,#0fh
            cmp         ah,#9
            js          hex1
            add         ah,#30h
            jmp         next1
hex1:        add         ah,#37h
next1:      mov         al,ah
            out         al,#0    ;output second highest
                                 ;nibble on screen
            mov         al,cl
            shr         al
            shr         al
            shr         al
            shr         al
            and         al,#0fh
            cmp         al,#9
            js          hex2
            add         al,#30h
            jmp         next2
hex2:       add         al,#37h
next2:      out         al,#0    ;output third highest
                                 ;nibble on screen

            mov         al,cl
            and         al,#0fh
            cmp         al,#9
            js          hex3
            add         al,#30h
            jmp         next3
hex3:        add         al,#37h
next3:       out         al,#0   ;output least significant
                                 ;nibble on screen
            ret


_in:   .ends
               .end

$
```

```
            .public     _print
_print:     .segment    code
            .assume     cs:_print,ds:$common

            push        bp
            push        ax
            push        cx
*stack:
*    (sp+24): ymk lo
*    (sp+22): yik lo
*    (sp+20): ok lo
*    (sp+18): bk hi
*    (sp+16): bk lo
*    (sp+14): ak hi
*    (sp+12): ak lo
*    (sp+10): t lo
*    (sp+8): oldcs
*    (sp+6): oldip
*    (sp+4): bp
*    (sp+2): ax
*       sp--> cx

          mov         bp,sp      ;have bp point to
                                    ;bottom of stack

          mov       al,#13     ;carriage return
          out       al,#0
          mov       al,#10     ;line feed
          out       al,#0

          mov       ax,10[bp]  ;print "t"(word)
          call      asciprt
          mov       ax,#32
          out       ax,#0

          mov       ax,14[bp]  ;print "ak"(doubleword)
          call      asciprt
          mov       ax,12[bp]
          call      asciprt
          mov       ax,#32
          out       ax,#0

          mov       ax,18[bp]  ;print "bk"(doubleword)
          call      asciprt
          mov       ax,16[bp]
          call      asciprt
```

```
        mov     ax,#32
        out     ax,#0

        mov     ax,20[bp]   ;print "ok"(word)
        call    ascirt
        mov     ax,#32
        out     ax,#0

        mov     ax,22[bp]   ;print "wik"(word)
        call    ascirt
        mov     ax,#32
        out     ax,#0

        mov     ax,24[bp]   ;print"ymk"(word)
        call    ascirt
        mov     ax,#32
        out     ax,#0

        mov     al,#13    ;carriage return
        out     al,#0
        mov     al,#10    ;line feed
        out     al,#0


END:    pop     cx
        pop     ax
        pop     bp

        retl

*****************************************
* THIS SUBROUTINE PRINTS A WORD(THE VALUE IN *
* THE AX REGISTER) IN HEX ON THE SCREEN OF   *
* THE WORKING TERMINAL;IT IS INVOKED TWICE TO*
* HANDLE 32 BIT VARIABLES;4 BIT GROUPS OF THE*
* NUMBER TO BE DISPLAYED ARE CONVERTED TO THE*
* CORRESPONDING ASCII CODE                   *
*****************************************
ascirt:
        mov     cx,ax   ;cx and ax registers
        mov     ah,ch   ;are used

        shr     ah      ;transform highest nibble
        shr     ah      ;of ax to hex(in ASCII)
        shr     ah
        shr     ah
        and     ah,#0fh
```

```
            cmp     ah,#9
            js      hex
            add     ah,#30h
            jmp     next
hex:       add     ah,#37h
next:       mov     al,ah
            out     al,#0   ;output hishest nibble on
                            ;on screen
            mov     ah,ch
            and     ah,#0fh
            cmp     ah,#9
            js      hex1
            add     ah,#30h
            jmp     next1
hex1:        add     ah,#37h
next1:      mov     al,ah
            out     al,#0    ;output second highest
                             ;nibble on screen
            mov     al,cl
            shr     al
            shr     al
            shr     al
            shr     al
            and     al,#0fh
            cmp     al,#9
            js      hex2
            add     al,#30h
            jmp     next2
hex2:       add     al,#37h
next2:      out     al,#0    ;output third highest
                             ;nibble on screen

            mov     al,cl
            and     al,#0fh
            cmp     al,#9
            js      hex3
            add     al,#30h
            jmp     next3
hex3:        add     al,#37h
next3:       out     al,#0   ;output least significant
                             ;nibble on screen
            ret


_print:   .ends
             .end
$
```

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