INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

STUDIES IN THE EMULATION OF A COMPUTER SYSTEM

BY

SHING CHOY HUNG

THESIS SUBMITTED TO THE SCHOOL OF GRADUATE STUDIES,

UNIVERSITY OF OTTAWA, IN PARTIAL FULFILLMENT

OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

DEPARTMENT OF ELECTRICAL ENGINEERING

FACULTY OF SCIENCE AND ENGINEERING

UNIVERSITY OF OTTAWA

OTTAWA, ONTARIO

1977

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

ABSTRACT

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1 BASIC CONCEPTS 1

1.1 Introduction 1

1.2 Fundamentals 4

1.3 Emulation of a Target Machine 14

CHAPTER 2 EMULATION OF CENTRAL PROCESSOR INSTRUCTIONS 20

2.1 Introduction 20

2.2 Conventional Machine Language 20

2.2.1 Instruction Formats 20

2.2.2 Addressing Structure 22

2.2.3 Instruction Types 25

2.3 Central Processor Emulation 27

2.3.1 NOVA Series Computers 30

2.3.2 MICRODATA 1600 Computer 33

2.3.3 NOVA Emulator on the MICRODATA 1600 37

2.4 Conclusion 40

CHAPTER 3 EMULATION OF I/O SYSTEMS 49

3.1 Introduction 49

3.2 I/O Emulation Process 50

3.2.1 I/O System Structure 51

3.2.2 Interrupt Structures 54

3.2.3 Input/Output Instructions 56

3.2.4 Data Representation and Record Structure 56

3.2.5 Time Dependencies in I/O Processing 57

3.3 Emulation of NOVA I/O System on the MICRODATA 1600 57

3.3.1 MICRODATA 1600 Input/Output System 58

3.3.2 NOVA Input/Output System 72

3.3.3 NOVA I/O Emulation on the MICRODATA 1600 78

3.4 Summary 81

CHAPTER 4 Concluding Remarks 93

4.1 Introduction 93

4.2 Trade-offs in the NOVA Emulator Design 94

4.3 Outlook and General Discussion 94

APPENDIX A LIST OF NOVA INSTRUCTION SET 100

APPENDIX B ALPHABETICAL LIST OF MICRODATA 1600 MICRO-INSTRUCTIONS 102

APPENDIX C THE LISTING OF NOVA EMULATOR 104

REFERENCES 126

LIST OF FIGURES

Figure	Title	Page
1.1	Functional Structure of a Computer System	6
1.2	Primitive Operations Diagram	7
1.3	Five-level Nested Virtual Machine Structure	10
1.4	Two-level Nested Virtual Machine Structure	13
1.5	Main Memory and Control Store Map of IBM 360/65 for 7090 Emulation	18
2.1	MICRODATA 1600 CPU Block Diagram	31
2.2	The Basic Flow-chart of NOVA Emulator	41
2.3	The Memory Mapping Diagram of NOVA Emulator	42
2.4	Flow-chart for the MICRODATA Emulation of NOVA LDA and STA Instruction	43
3.1	Single Bus Structure	52
3.2	Multiple Bus Structure	53
3.3	Mapping of ESA to the External Interrupt Service Routine	70
3.4	Mapping of ESA to CA and EA in Concurrent Interrupt	73
3.5	Flow-chart of INT Routine	82
3.6	Flow-chart of NOVA TTI Instruction Executive Routine	86
4.1	Basic Modular Structure of NOVA Emulator	95

LIST OF TABLES

Table	Title	
2.1	File Register Assignments	38
2.2	Execution Times for the NOVA Emulator and the NOVA 1200	45
2.3	Execution Times for Similar Instructions on NOVA Emulator and MICRODATA 1600	47
3.1	I/O Control Microinstructions and Control Modes	59
3.2	File Register 0 Flags	61
3.3	Device Orders	63
3.4	Status Bytes Definition	64
3.5	Internal Status Byte	67
3.6	The Configuration of the Host and Target Machines	89

ACKNOWLEDGEMENTS

The author would like to thank Dr. D.K. Banerji for his guidance, helpful suggestions and encouragement during the course of this work.

The author is indebted to Dr. G. White for his assistance and encouragement during the testing of the NOVA emulator and to Dr. J. Raymond for his help in getting the emulator operational on the MICRODATA 1600.

Thanks are also due to Mrs. Claudette Henderson for her efficient typing of this manuscript.

The financial assistance of the University of Ottawa and the National Research Council under grant no. A-8306 is gratefully acknowledged.

Abstract

Various methods have been used in the past to implement the control structures of digital computers. Microprogramming is regarded as the most versatile approach to this end. It reduces the complexity and increases the flexibility of the control unit. Furthermore, dynamic microprogramming allows modification or reconstruction of the system architecture as viewed at the machine language level; such a system can be based on the "emulation" concept which utilizes microprogramming techniques to interpret various system architectures at the machine language level.

A study of the process of emulation is the subject of this thesis. The thesis describes the main features of both the target and the host systems that affect the emulation process. The principles underlying the design of an emulator are also discussed. For this purpose, a Data General NOVA system has been emulated on the MICRODATA 1600 and evaluated. The thesis concludes with some recommendations for further work.

CHAPTER 1

BASIC CONCEPTS

1.1 Introduction:

Changing from an existing computer system to another one with a different architecture is not a matter of simply removing the older system and plugging in the new one. The process is invariably full of problems and often requires a considerable amount of effort. In most cases, it is not economically feasible to discard the existing software, specially if a significant software library has been built up for the old system. Software development costs are much higher compared to the cost of hardware, and it may be prohibitively expensive to rewrite all the existing software to fit the new machine. Some relief is provided with the use of high level languages; if the user programs are all written in high level languages such as FORTRAN or COBOL, basically they can be run on the new system if a compiler is provided for these languages. But the problem is that high level languages have not been universally standardized. For example, programs written in ANSI FORTRAN or COBOL will not run on a machine which has a different version of FORTRAN or COBOL. Furthermore, the high level language programs may be mixed with machine or assembly language coded routines, so a degree of reprogramming is still required. However, despite these shortcomings, high level language coding can be a significant aid in conversion. In order to obtain a smooth transition from one machine to another, there are several approaches that can be followed. These techniques cover the range from reprogramming,

code to code translation, simulation and emulation [1]. A brief description of these approaches follows.

Reprogramming: While reprogramming the old system's workload for the new system, one can take advantage of any favorable features of the new machine. But this process is very time-consuming, error-prone, expensive, and usually the effort required is prohibitive. This is the least desirable solution to the problem of conversion.

Code to Code Translation: Prior to execution in the new machine, each instruction in the source program for the old system is replaced by one or more instructions of the new machine, which would execute the same operation intended by the old instruction. The resulting source program for the new machine is then assembled and executed. But it is difficult to automate the translation of an instruction that is modified according to the data or machine state before its actual execution. For instance, suppose a user program contains a segment in which one of the instructions is replaced by another instruction (which is specified as data in the program), if a certain machine state is met prior to the execution of that program segment. In this situation, any automated code to code translator will fail to do proper translation. Because the replacing instruction in the user program is specified as data, during translation no code conversion is required. Therefore, the code produced by the automated translator will not perform the same operation as intended by the old program. Solving a problem of this kind requires programmer assistance in defining the problem and reprogramming it by hand. In the preceding example, the pro-

grammer may need to create two non modifying program segments for the new machine to the modifiable segments of the old system. This approach can be successful only for systems with very similar structure; as machine differences increase (difference in bytes, word or file formats), so do translation problems.

Simulation: An image of the entire state of the simulated (target) machine is maintained on the simulating (host) machine i.e., the simulated machine is mapped on a bit-for-bit or character-for-character basis on the host system. Each instruction (of the simulated machine) is interpreted in sequence at execution time to alter the machine image in the same manner as the instruction would have done in the simulated machine. The simulator must interpret an instruction each time it is executed. Due to this interpretive overhead, programs may run more slowly on the host machine than on the original machine. One reason for longer execution time could be the failure of the simulation program to effectively preserve any parallelism that might exist in the target machine, and to interpret the parallel operations in an efficient sequential manner.

Emulation: Emulation may be defined as hardware, microprograms (firmware), and software* added to one computer system to enable it to execute programs written for another system. Any of these

---

* Rosin [2] defines emulation as implementing a target machine entirely by firmware (microprogramming) in the host machine. However, efficient emulation may require hardware/software/firmware tradeoffs and for this reason, we will not use Rosin's definition.

components may be absent in a specific case, but as a minimum either added hardware or added microprograms must be present. Without one of these, we say we have a simulator [2]. The system which supports emulation is known as the host machine while the other system is known as the target or virtual machine.

A study of the process of emulation is the subject of this thesis. For this purpose, a Data General NOVA system emulator has been implemented and evaluated; the host machine is a MICRODATA 1600 . In the rest of this chapter, fundamental concepts of microprogramming and emulation are described.

Chapter 2 deals with the details of implementation and evaluation of the emulator for the CPU instruction set (Arithmetic and logic instructions, and memory reference instructions) of the NOVA system. Chapter 3 deals with the emulation of the input/output (I/O) subsystem and Chapter 4 describes the integrated emulation for the NOVA system.

1.2 Fundamentals:

A conventional digital computer is functionally organized into five basic units, namely:

(1) Input

(2) Output

(3) Memory

(4) Arithmetic and logic unit (ALU)

(5) Control section

These five units communicate with each other through electronic

signals that represent data, instructions and control information as shown in Fig. 1.1.

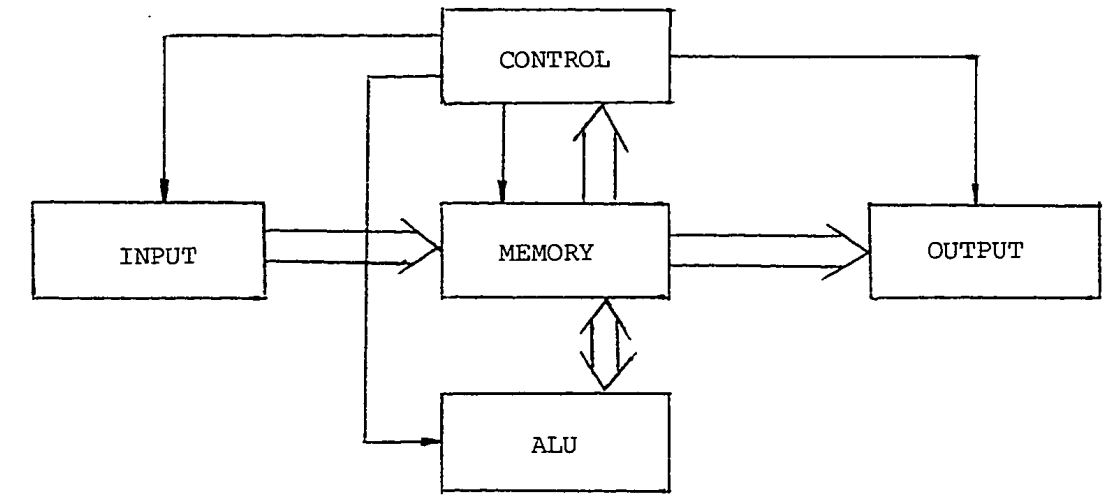
The control unit of the computer governs the operations of the various sections and directs the flow of information by emanating control information (signals) in a predetermined time sequence to open or close a subset of control gates throughout the system. The general control functions consist of fetching and decoding the machine instructions from main memory, gating data paths to perform the desired operations on data fields, and changing the state of the computer to allow the next required operations to be performed.

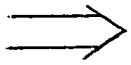
Practically, computers are required to perform complex operations as defined in the repertoire of their machine language i.e., addition, subtraction, shifting, etc. These operations can be synthesized by performing sequences of primitive operations. These primitive operations are governed by the control signals from the control unit, and generally completed in one or two basic clock cycles.

The basic primitive operations can be any of the following items: (1) Move Operation, (2) Unary Operation, (3) Binary Operation, (4) Convert Operation. They are constituted by connecting some basic building blocks in a variety of combinations as illustrated in Fig. 1.2. The basic building blocks are:

1. Transformation unit, which performs a specific transformation or operation on data, e.g., a hardware adder unit.

2. Storage element, which is capable of retaining information over a period of time, e.g., a register or a word in main memory.



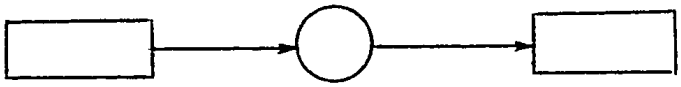

INFORMATION (DATA OR INSTRUCTION)

CONTROL

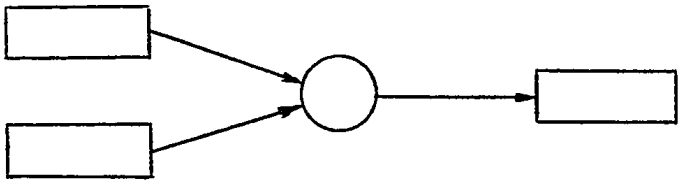
Fig. 1.1 Functional Structure of a Computer System

(a) MOVE OPERATION

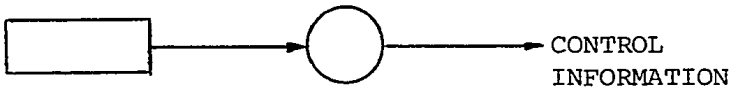
(b) UNARY OPERATION

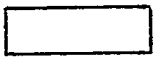


(c) BINARY OPERATION



(d) CONVERT OPERATION





STORAGE ELEMENT

LINK

TRANSFORMATION UNIT

Fig. 1.2 Primitive Operations Diagram

3. Links, which are paths (simply the wires) used to transfer information among storage and transformation units.

The hardware structure of a computer system can be described in terms of these elements.

One method of implementing a complex operation is to design combinational and sequential logic networks (control unit) which generate sequences of control signals to activate the primitive operations which constitute the desired complex operation; this method is known as hard-wired or conventional control implementation. But this design method is somewhat ad hoc, and as the machine functions become more complicated, it results in a very complex control unit. Also computers whose control units are designed in this fashion can perform only those operations which have been built (wired) into the control unit.

An alternative method of implementation that can reduce the complexity and increase the flexibility of the control unit is the use of microprogramming. The word "microprogramming" was coined by Professor Maurice Wilkes in his 1951 paper "The Best Way to Design an Automatic Calculating Machine" [6].

Microprogrammed implementation of control functions is achieved by using microinstructions to represent unique primitive operations. The microinstructions are stored in a control storage from which they are fetched, decoded and executed to generate the control signals required to activate the primitive hardware resources to achieve the defined primitive operations. In other words, the hard-wired control is replaced by a microprogrammed control section. Control signals (or control information) are stored in a fixed or

dynamically changeable control memory (ROM or RAM), which is traditionally separated from main memory. A complex operation can thus be represented by a sequence of stored microinstructions (microprogram) in control store. Executing a complex operation (machine instruction) reduces to executing the appropriate microinstructions in sequence.

The role of microprogramming would become more evident if a computer system is visualized as a hierarchical structure of nested virtual machines as illustrated in Fig. 1.3 [4], [5].

Problem oriented language level consists of a virtual machine that accepts languages designed to be used by applications programmers with problems to solve, e.g., COBOL, FORTRAN, GPSS, etc. The assembly language level is a symbolic form for one of the underlying languages (level 2 or level 1). An assembly language program is first translated to level 1 or 2 language and then interpreted by the appropriate virtual machine. The operating system machine level supports or manages all the other virtual machine levels. The conventional machine level represents machine language instructions carried out interpretively by the microprograms (e.g., 4180 000A, 1BCC represent LA 8, 10 and SR 12, 12 instructions, respectively, in IBM 360 assembly language). The microprogramming level represents the level at which the computer electronic circuits are directly activated to affect data flow within the hardware.

A physical machine is functionally defined by a set of external attributes: an instruction set, storage media (main memory, disks, tapes, etc.), interrupt system, channel configura-

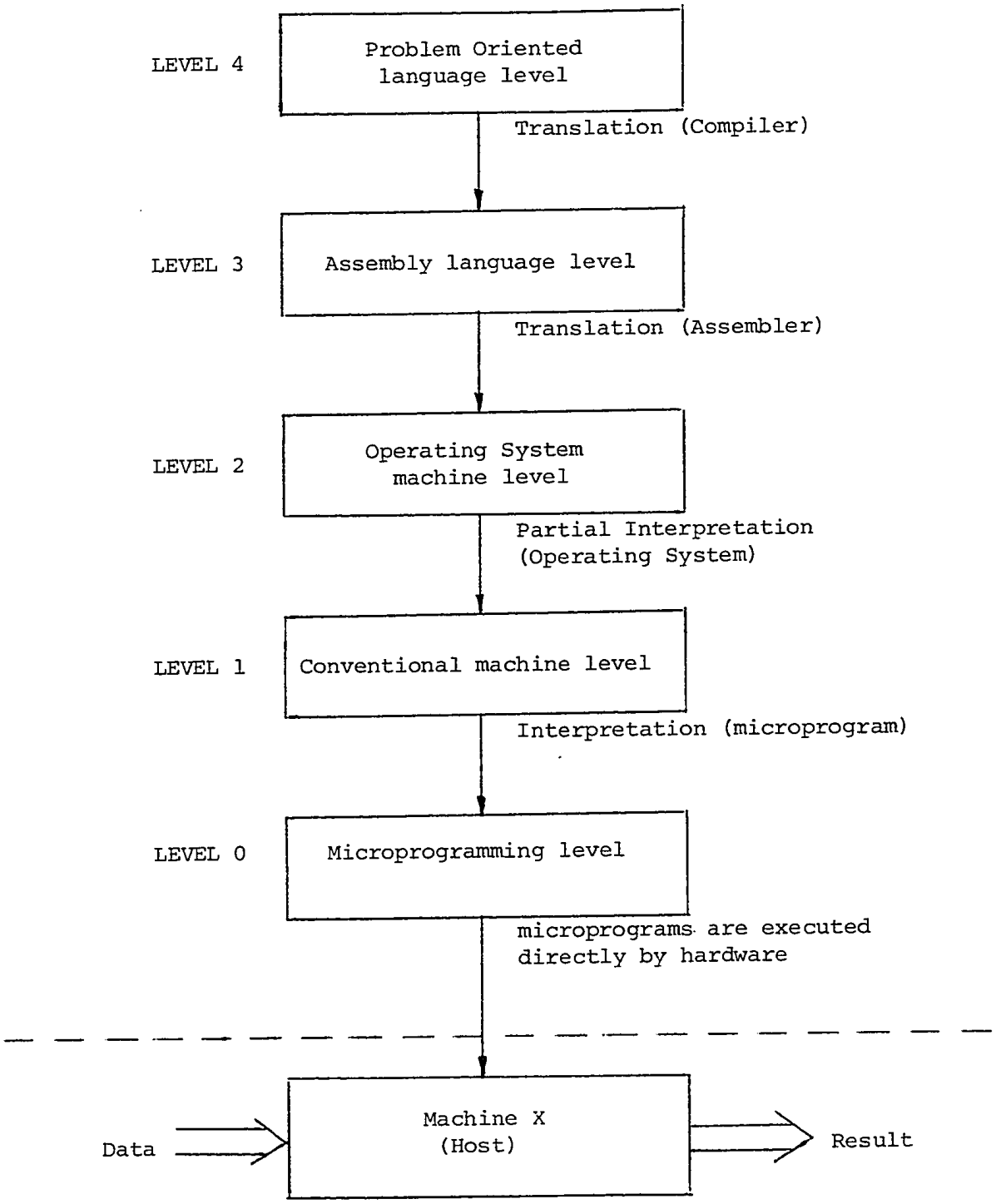


Fig. 1.3 Five-level Nested Virtual Machine Structure

tion and so on. Such a system is called a host system. Programs written in this system's language (in its defined machine code) can be run directly in this machine.

Based on the host system, a virtual machine or a number of nested virtual machines can be created through the agency of programming that transform the host's attributes to give the user the illusion of existence of a hypothetical computer system or virtual machine. Generally, the hypothetical system can use the power of the computer system more effectively and often reduces the total problem solving time in special areas. For instance, on a time-sharing interactive information retrieval system (e.g., an air-line reservation system) the operator can use an application oriented language to access the information from the system and to communicate with it. The user gets the feeling that the computer accepts commands in this language. In this case, what the user sees is really a hypothetical or virtual machine. To the user, it does not matter whether the commands are directly executed by the hardware or whether they are translated to an intermediate level language and then executed.

There is an important relation between a language and a virtual machine; each virtual machine has an associated machine language and all programs written in that language can be executed by the virtual machine. Conversely, a language can also define a virtual machine.

A machine with N levels can be regarded as consisting of N different virtual machines, each with a different machine language. Only programs written in the base level (level 0) can be directly

executed by the electronic circuits without the need for intervening translation or interpretation.* But the programs written in a higher level language (i.e. level 1, level 2, ... level N) must either be interpreted by an interpreter running on a lower level, or translated to the language corresponding to a lower level. For instance, in a conventional 2 level machine as shown in Fig. 1.4, programs written in assembly language (level 1) have to be translated by the assembler into the machine instruction code (level 0). Then the machine instructions are fetched and decoded, and control signals are generated to activate the hardware resources to perform the desired operations.

The programmer writing programs for the level N virtual machine need not be aware of the underlying interpreters or translators. He only knows that his programs, written for the N^{th} level virtual machine, can be executed.

As shown in Fig. 1.3, a microprogrammed machine (a machine whose control unit has been implemented using microprogramming techniques) can be visualized as a hierarchy of virtual machines. As mentioned earlier, the machine instructions (level 1) which are

---

* The difference between translation and interpretation is as follows: in translation, an instruction in the level N language is translated into an equivalent sequence of instructions in the level N-1 language. The translated program (in level N-1) language) is executed instead of the program in level N. In interpretation, an instruction in the level N language is examined and decoded, then a sequence of equivalent level N-1 instructions is executed immediately; no translated program is generated.

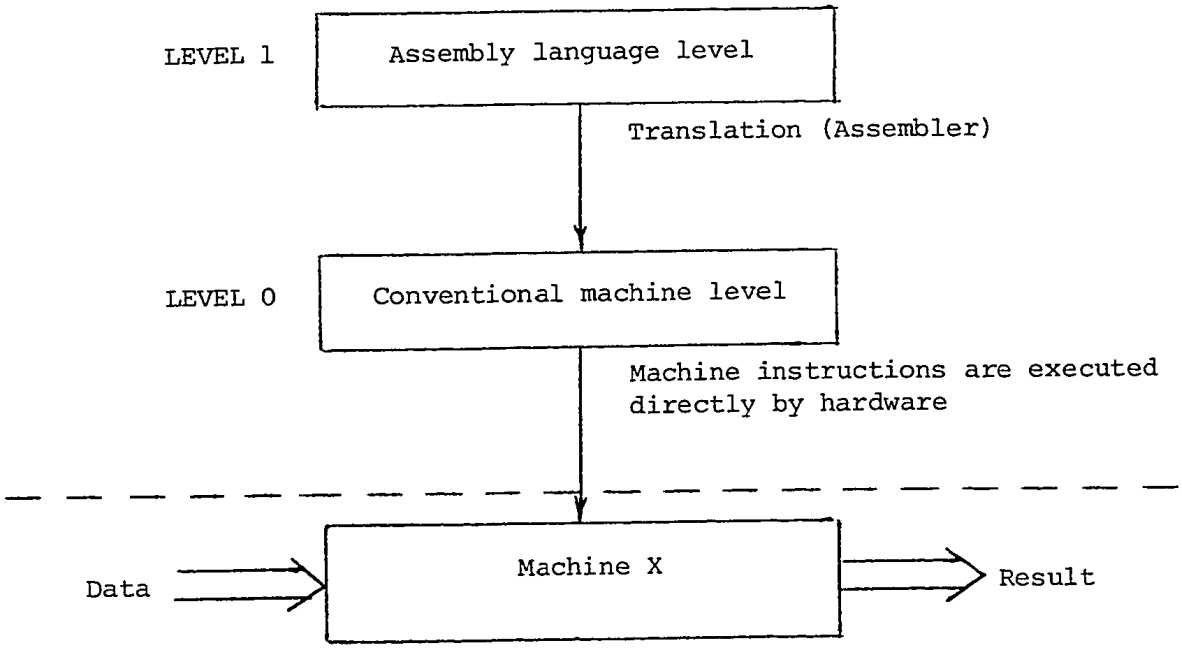


Fig. 1.4 Two-level Nested Virtual Machine Structure

stored in the main memory, are fetched and interpreted by the microprograms in the control store. In this structure, a new level 1 virtual machine can be defined by changing the contents of the control store.

A new term "nanoprogramming" has been coined recently, but it is simply a matter of creating another lower level virtual machine, and raising the microprogram level to an upper level. The only thing one has to bear in mind is that the programs written in the base level (the lowest level, level 0) can directly activate the primitive hardware resources and carry out the operations without either translation or interpretation.

1.3 Emulation of a Target Machine:

Before discussing the details of emulation, let us recall some basic definitions for microprogrammed processors.

A microcycle is the quantum of time required to execute one microinstruction.

A microcommand causes the hardware to generate the electronic signals required to perform a single primitive micro-operation during one microcycle.

A microinstruction is a collection of one or more microcommands, which performs a single micro-operation or several micro-operations during one microcycle. Such a microinstruction set is less powerful than the machine instructions of a conventional general purpose processor. Implementing a conventional machine instruction would require a sequence of microinstructions (microprogram). Sometimes,

a microinstruction is more complex than a conventional machine language instruction (e.g., microinstructions of IBM 360 system) and a considerably detailed knowledge of the computer architecture is needed to microprogram these systems.

A control store contains microinstructions, and usually has a faster access time compared to the main memory access time. It may consist of a Read Only Memory (ROM) or/and writable memory (RAM). In order to execute a sequence of instructions (microprogram), the system must fetch the instructions from the control store in the desired sequence.

A microprogram is a sequence of microinstructions which performs a useful task. A machine instruction is executed by initiating the associated microprogram execution. A static microprogramming system implements a machine instruction repertoire by means of fixed programs in a Read Only Memory. Usually, the processor is closely tailored (in word size, arithmetic conventions, etc.) to the machine vocabulary which it is to implement. A dynamic microprogramming system uses a read/write control store for storing the microinstructions. Such a system allows modification of the system architecture as observed at the machine language level. The same hardware can be made to support a variety of system architecture.* In a larger sense, the term microprogramming really means interpretive programming. The source language, the lowest level language into which user programs are

---

* Architecture of a system is defined as the attributes of the system as seen by the programmer.

translated for the interpreter can be called a directly executable language, DEL. The object language consists of the microinstructions. The DELs may be extensible and changeable with the notions of emulation (which will be discussed in the following sections).

It is important to have a thorough understanding of the following items in order to properly define the image of the target machine on the host. These consist of:

(1) The structure, size and addressing of main memory, registers, stacks, etc.

(2) Machine language formats, including effective operand address calculation.

(3) Operation codes and their functional meanings.

The process of designing an emulator involves the mapping of the emulated machine onto the host machine. This consists of:

1. Main storage mapping.
2. I/O system mapping.
3. Mapping of registers, triggers, counters, accumulators, and all other resources in the target machine that are addressable by the programmer.

Generally, two approaches have been used for emulation, namely, software-controlled emulation and firmware-controlled emulation.

Software-controlled emulation: In this approach the designer first develops and analyzes a pure software simulator of the target system. Certain host machine instruction sequences will stand out as being very frequently used, or as being awkward ways to perform simple functions, or both. These sequences become prime candidates for hardware or firmware support. This support is provided

by defining new instructions, each replacing such a sequence. These new instructions are in addition to the standard instruction set of the host machine and they aid the emulation process. This hybrid approach to emulation reduces the size of control store needed to provide emulation. Each instruction of the target machine is interpreted by a subroutine contained in the main memory of the host machine, and the subroutine uses special instructions as well as the conventional instructions of the host system to initiate the operations, as in the IBM 360/65 emulator for the IBM 7090 machine (see Fig. 1.5). One of the most notable of the special instructions devised for this purpose is the DO Interpretive Loop (DIL) instruction which is added to the conventional machine language of IBM 360/65 machine. The DIL instruction, implemented in ROM, is a single instruction replacing the subroutine that a pure software simulator would use to:

1. Fetch the simulated Instruction Counter (IC).
2. Convert the IC value to the address of the storage cell that contains the next instruction to be interpreted.
3. Fetch this next target instruction.
4. Update and store the simulated IC.
5. Perform any indexing required for the target instruction.
6. Convert the effective address obtained to the address of the storage cell containing the target operand.
7. Interpret the target instruction operation code and branch to the appropriate simulator routine that will

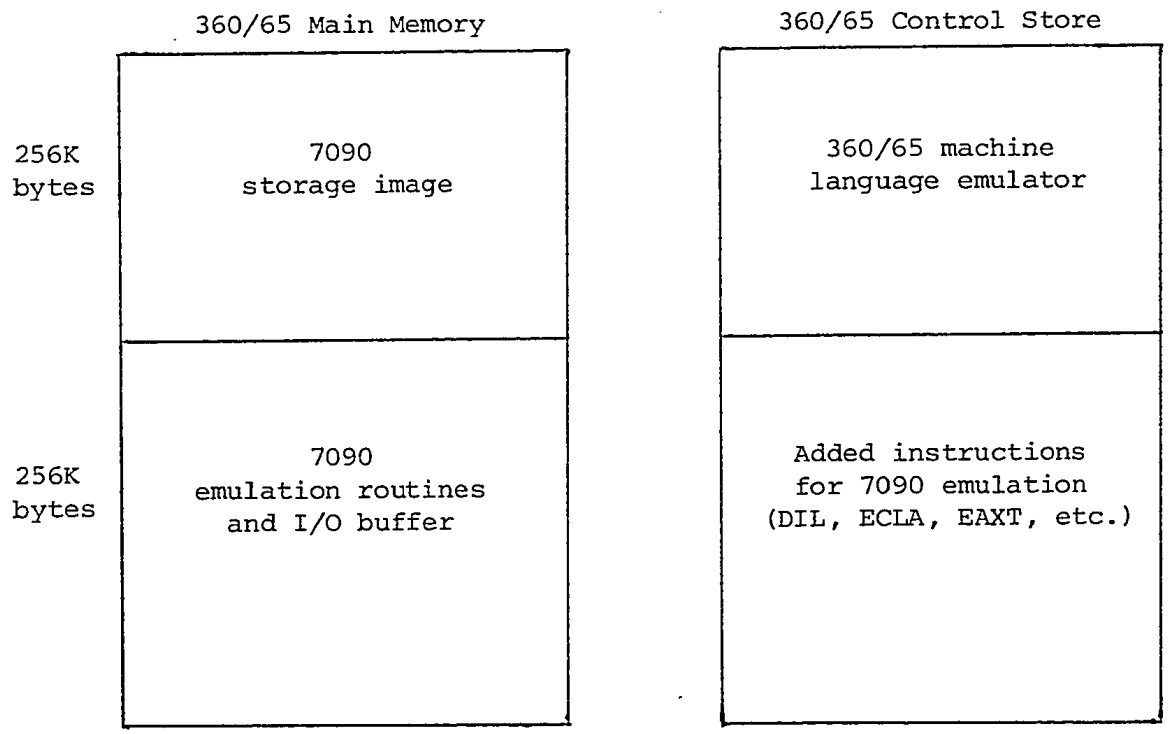


Fig. 1.5 Main Memory and Control Store Map of IBM 360/65 for 7090 Emulation

simulate the instruction.

The DIL instruction is used once for each instruction emulated.

Example 1.1: 360/65 emulation of 7090 instructions.

7090 Instruction	360 Emulation Routine
AXT (address to index true)	EAXT Microroutine that does address to index true. DIL Microroutine that does fetch and interpretation of next instruction.
CLA (clear and add)	ECLA Microroutine that does clear and add. DIL Microroutine (see above).
AXC (address to index complement)	LCR 360 instruction that complements that address. EAXT Microroutine (see above). DIL Microroutine (see above).

Firmware-controlled Emulation: For a dynamic microprogramming system, the computer can be restructured to emulate any computer instruction repertoire by loading the associated microprograms in the control store, so that programs of the target machine can be run without changing their object code. The emulator can take advantage of any favourable features of the host machine to achieve efficient performance. The actual performance achieved also depends on the degree of added hardware and software in the host machine. Software still has an important part to play in the emulation process, since in some cases it becomes uneconomical to implement a function by microprogramming, such as error recovery, code translation by table look-up, etc.

CHAPTER 2

EMULATION OF CENTRAL PROCESSOR INSTRUCTIONS

2.1 Introduction:

The central processor instruction set of most computers is functionally similar. It involves data manipulation, transfer of data among addressable registers and memory cells, changing instruction execution sequence, and so on. The machine instruction format is tailored according to the machine architecture and usually it varies from one machine to another. The length of operands, data representation and memory addressing mechanisms are usually in a variety of formats. All these components affect the emulation process.

Before discussing the central processor emulation in detail, a general survey of the conventional machine language of a typical system is provided to give the reader some idea about the various instruction formats and their functional meanings.

2.2 Conventional Machine Language:

2.2.1 Instruction Formats:

The information stored in the memory locations of a machine consists of two types: i) The instruction word which is used to determine the sequence of primitive operations which the machine would perform, and ii) the data word which is operated on by the machine in the process of executing an instruction. The job of

the control section is to fetch the next instruction word from the memory, decode the instruction and perform the desired operations. In order to fulfil these tasks, the instruction words must contain the following basic information:

1. The operation code: To specify what action (operation) is to be performed.
2. The addresses of the operands: To specify the location of data which is operated on by the instruction.
3. The address of the next instruction to be executed: This is not always necessary since the instructions are usually sequential; it is needed only in case of branch operations.

The design of the instruction repertoire reflects the architecture of the computer system and the availability of the hardware resources. In some systems, all instructions may have the same length (e.g. NOVA instructions are 16 bits long), while in others they may be of variable length, as in MICRODATA 1600 or IBM System/360. The operation-code part of the instructions may be of a fixed or variable length depending on the system. In addition, the number of operands in an instruction may vary according to the design of the machine instructions. Generally, a machine can be classified according to the number of addresses that are used in its instructions:

1. Three-address machine

Instructions specify the location of the two operands for a binary operation and the location where the result is to be stored.

2. Two-address machine

Instructions specify two operands and the result is stored

in one of the predetermined operand locations.

3. One-address machine

Instructions only specify one operand; a special register (accumulator) provides the other operand, and the result is usually stored in the accumulator.

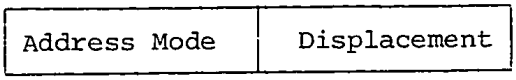
4. Zero-address machine

Generally, the instructions contain only the operation code; the operands are taken from a stack. Such an instruction set is typically used in a stack oriented system.

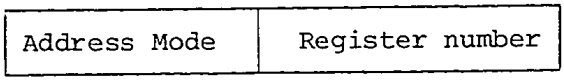
2.2.2 Addressing Structure:

Data stored in the main memory have to be fetched and manipulated to execute the desired operations as specified by the operation code. The addressing information for this purpose is specified in the operand field(s) of the instruction. Two typical memory reference formats are:

(a) Memory Word Addressing



(b) Register Addressing



In the former one, the effective address is calculated according to the address mode and the displacement value specified in the instruction. In register addressing, the address field contains the register designator and the operand address is determined by the address mode and the contents of the selected register.

Displacement field is a signed integer and gives the displacement from a specified memory location, which is determined by the address mode. Address mode field is used to specify how the displacement field is to be used in computing the effective address or how the contents of the selected register are to be used to locate the operand. There are typically five addressing modes used in most computer systems. In the following discussion "the contents of the selected register" may substitute the word "displacement" where register addressing is concerned.

1. Direct Addressing:

The displacement field specifies the absolute address of the operand. Some mini-computers divide memory into fixed pages of 256 or 512 words and allow direct addressing of the first page only.

2. Indirect Addressing:

The displacement field or the result of some address computation gives the memory location that contains the address of the operand. Usually indirect addressing may be used as a means of obtaining a full length memory address.

3. Indexed Addressing:

The displacement field is added to or subtracted from the contents of an index register. In some computers the contents of the index register may be automatically incremented or decremented after each operation. This scheme allows for efficient address calculation when operating with sequential locations of memory.

4. Relative Addressing:

The effective address is obtained by adding the displacement

field to one of the registers in the computer. Typical registers that are used are:

(a) Program Counter: This allows the programmer to have a "floating" page which represents a fixed number of memory locations preceding or following the address of the current instruction.

(b) Base Register: Where the base register contains the starting address of the memory segment assigned to the data or instruction being referenced.

5. Immediate Addressing:

The address part of the instruction actually contains the operand itself rather than an address, thereby eliminating the need for effective address calculation and fetching of data.

Most machines allow a combination of two or more of the stated techniques in their addressing mechanism to enhance their addressing flexibility. In addition to the addressing techniques that have been discussed, stack addressing capability is provided in some computers. They have the facility to push/pop the contents of memory locations or registers into/from the stack and update the stack pointer automatically. Except for the instructions to fetch from memory or store information into memory the instructions in these machines specify no addresses at all. A diadic operation operates on top two levels of the stack by popping these two operands from the stack; after the specified manipulation the result is pushed on top of the stack.

2.2.3 Instruction Types:

In a conventional machine, every machine instruction represents an algorithm for changing the status vector of the processor; some instructions change the value of one or more memory words or registers, others may merely change the program counter etc. However, the functions of most instructions among different machines are basically similar. In this section the major types of general purpose instructions will be discussed in detail:

1. Data Movement Instructions:

These instructions require both the source and the destination of the data to be specified either implicitly or explicitly in the instructions, and also indicate the amount of data to be moved. The execution of these instructions results in data being moved from one place (a particular memory word, register, stack or immediate data) to another. In some systems, certain "move" instructions not only move data but also set condition codes within the system depending on whether the data is zero, positive or negative.

2. Dyadic (Binary) Instructions:

Dyadic operations require two operands to produce a result. Most of the arithmetic and logic instructions (e.g., ADD, SUB, AND, OR, etc.) are classified into this category. Typically, condition codes are updated after each operation.

3. Monadic (Unary) Instructions:

Monadic instructions operate on only one operand to produce a result. Generally, these instructions provide the capability of shifting or rotating the contents of a register word or byte,

and typically, condition code updating is also involved.

4. Conditional Branch Instructions:

These instructions are used to test a specified condition and alter the instruction flow sequence if the condition is satisfied. A common method is to test some condition set by the previous operations and jump to a particular memory location if the condition is met; otherwise the following instruction in sequence is executed.

5. Subroutine Call Instructions:

A subroutine call instruction saves the return address, then causes a jump to the subroutine starting location. In some machines the return address is placed in a single fixed memory location or stored in the first word of the subroutine prior to the first executable instruction. Another method involves a stack of registers used for saving the return addresses.

6. Loop Control Instructions:

Some machines have instructions to facilitate the execution of a group of instructions a fixed number of times. This involves using a counter (a register or memory word) that is incremented or decremented by some constant value and the counter is tested each time through the loop. If a certain condition holds the loop is terminated.

So far, conventional machine instructions have been discussed with the exception of Input/Output (I/O) instructions. The emulation of non I/O instructions of the virtual system (central processor instructions) is comparatively straight forward, since this does not involve any timing, data conversion or machine dependent

features. In the following sections, the hardware resources of the host system that affect the emulation process and the mechanism for CPU instruction set emulation will be discussed in detail.

2.3 Central Processor Emulation:

The hardware resources of the host machine play a significant role in the process of emulation. In essence the emulation process involves mapping the addressable resources of the target machine and the links between those resources on to the host. The better the mapping of the target resources onto the most appropriate host resources and also, the more similar the host and target machine structures, the better the performance of the emulator.

In order to properly transfer the target machine attributes onto the host, the primary characteristics of the hardware features of the host system (microprogrammable computer) must be carefully examined. These are:

1. Data Path Structure:

The data paths are used to connect the various functional components in a computer. Data may be transferred over them in one direction or in both directions. If the host and the target machines have different data path widths, then a virtual match through proper manipulation of the available facilities must be created. If the data path width of the host system is wider than the target's then it has to be reduced to the desired number of bits, usually by masking out the unwanted bits. On the contrary,

if the data path width of the host is smaller than the target's, iterative multiples of the host data path width are required to achieve a virtual wider path.

2. Functional Processing Unit (FPU):

This unit generally provides basic functional operations using full binary operands, and often a decimal arithmetic mode (4 bit BCD mode) is also included. Shift/rotate capabilities may either be provided in the airthmetic and logic unit (ALU) or in a separate unit. The simplest form of such a unit is the two input, one output ALU that performs simple arithmetic and logic functions. In some systems, in addition to shifters, the FPU may contain masking units which can selectively mask certain bits from the input operands or the result.

In addition to the fundamental (primary) functions (e.g. ADD, SUBTRACT, AND, OR, etc.) that should be implemented in a functional processing unit, some special functions (or non-primary functions) such as hardware multiply/divide facility may be provided in some systems. If the standard instruction set of the target machine includes hardware multiply/divide instructions, for example, and if this facility is not implemented in the host system, then they can be realized by a sequence of primary operations in the host control storage.

In a downward emulation, the FPU in the host machine is more versatile than that in the target system or the processing functions of the target machine FPU constitute a subset of the host's processing functions. If the hardware structure of both systems is similar, then not only the emulator can be implemented conven-

iently, but also the performance of the virtual system may be improved. On the contrary, in an upward emulation, not only is the implementation elaborate, also the performance of the virtual system may be highly degraded.

3. Local Storage:

The local storage is a set of registers used to hold information to be routed to other functional units and, generally, has very fast read/write times. Typically, the number of addressable registers is less than the total number of registers in a local storage. During emulation, in addition to the mapping of addressable registers, the non-addressable registers (e.g., instruction register, status register, etc.) also have to be mapped onto the host system. If the host machine has insufficient local storage for this purpose, then the slower main memory has to be used along with the available local storage, with a resulting loss of efficiency in terms of speed.

4. Main Memory:

The number of available words of main memory is usually limited by the addressability of the system (the size of the memory address register). Mapping of target machine main memory to host memory has a significant effect on the efficiency of the emulator. If a direct one-to-one mapping is not possible, a virtual match between the two different memory width systems, by using iterative multiple or masking method, is required to obtain the proper data and instruction width. The relatively slow access time of main memory degrades the efficiency of an emulator if the iterative multiple method rather than the masking method has to be used.

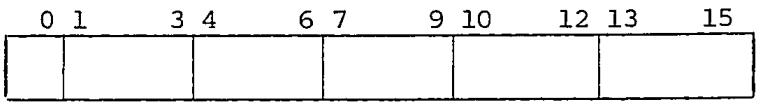
At this point, it would be appropriate to discuss an example of emulation of an actual system (Data General NOVA) on a MICRO-DATA 1600 microprogramming system. However, before giving any details of the emulator, a general introduction to these systems is necessary.

2.3.1 NOVA Series Computers

The NOVA line computers are general purpose computers with a 16-bit word length. All machines are organized around four accumulators which are used for temporary data storage, data manipulation in all arithmetic and logic class instructions, and also as part of the I/O system. The 15-bit address of the next instruction to be fetched is held in the PC (Program Counter). These five registers are accessible to programmers. Even though the internal organizations of various machines within the NOVA family are different from each other, the same instruction set is used among them and the programming for all machines is completely compatible.

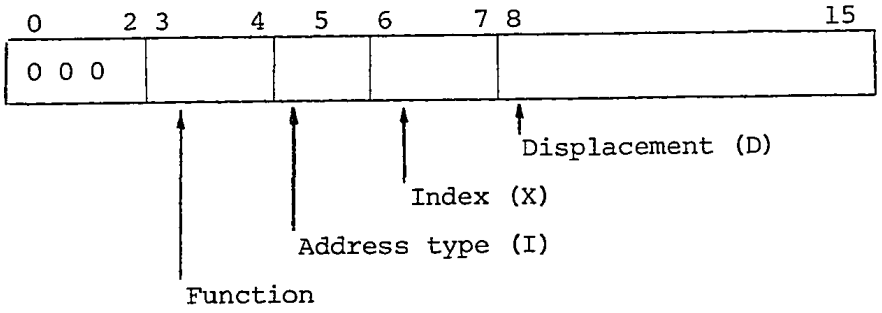
Instruction Formats

In the NOVA series machines the bits of a word are numbered from left to right as 0 to 15; the same numbering scheme is used in registers. By convention, all numbers representing instruction words, register contents, codes, addresses, and any data appearing in programs are always octal unless otherwise specified. The whole word is partitioned as follows:

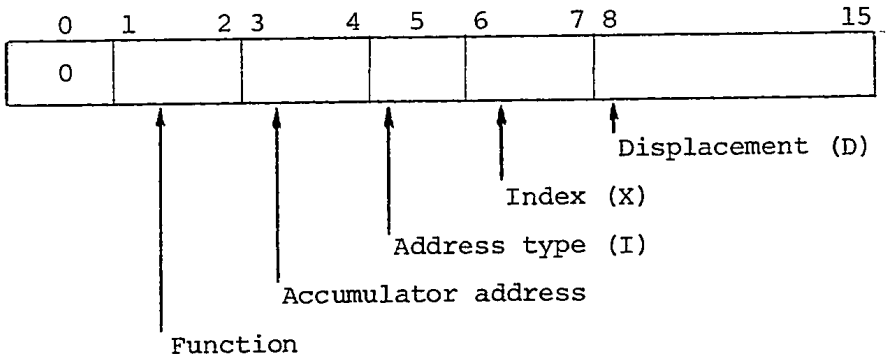


There are four basic types of instruction formats:

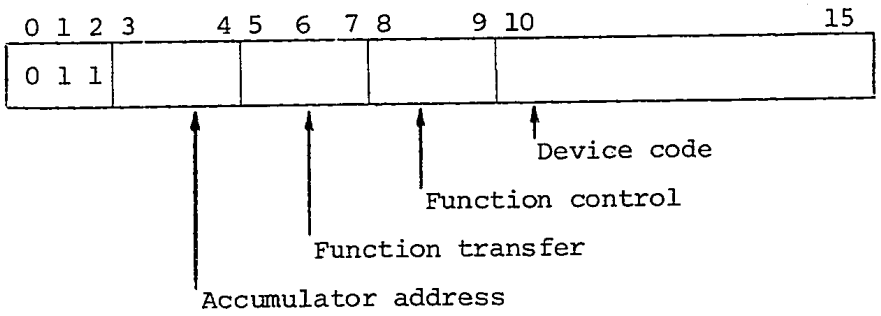
(a) Jump and modify memory commands



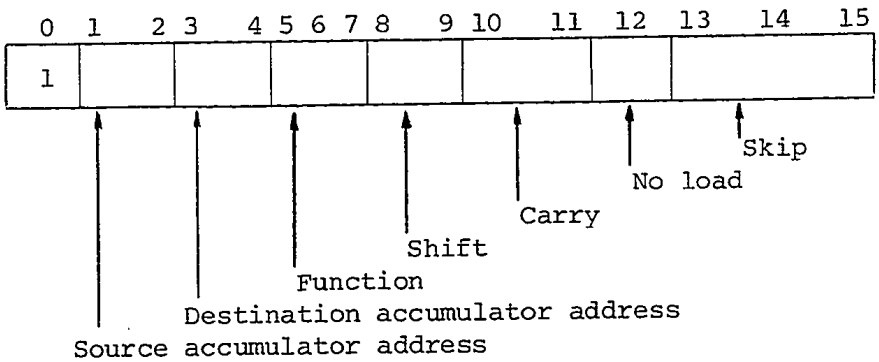
(b) Move Data commands



(c) Input-Output Commands

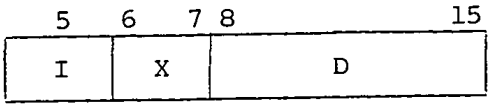


(d) Arithmetic and Logic commands



Addressing Mechanism

The effective address in the memory reference instructions depends on the values of I, X, and D:



where I: indirect bit

X: index bits

D: displacement

X	Derivation of address
00	Page zero addressing D is an address in the range 00000_8 to 00377_8
01	Relative addressing D is a signed displacement (-200_8 to $+177_8$) that is added to the address in PC
10	Base Register addressing D is a signed displacement (-200_8 to $+177_8$) that is added to the address in AC2
11	Base Register addressing D is a signed displacement (-200_8 to $+177_8$) that is added to the address in AC3

The calculation of operand address can be classified into four types.

1. Direct Addressing:

If I=0, the effective address of the operand would be:

Effective address = the contents of index register

± Displacement

or = Displacement D if X=00

2. Indirect Addressing:

If I=1, the processor retrieves another address from the location specified by the address already determined according to

the information in the instruction. The 0 bit of the retrieved new word is the indirect address bit. If this bit is zero, bits 1 to 15 are the effective address, otherwise they specify a location for another level of address to be retrieved. The process continues until some referenced location is found with a zero in the 0-bit position. Then bits 1 to 15 of the location are the effective address.

3. Auto incrementing/decrementing addressing:

If any memory location in the range 20_8-37_8 is accessed by indirect addressing, the processor will retrieve the contents of that specified memory location and increment or decrement the word retrieved, then write the altered word back into the same location and use the altered word as the new direct or indirect address.

- If the word is taken from locations 20_8-27_8, it is incremented by one.
- If the word is taken from locations 30_8-37_8, it is decremented by one.

4. Subroutine Jump

Execution of a JSR (Jump subroutine) instruction will load the return address in AC3 and cause a jump to the effective address and continue processing from there.

2.3.2 MICRODATA 1600 Computer

The MICRODATA 1600 is a microprogrammable digital computer with 8-bit word length in core memory and up to 64K byte memory capacity in the basic enclosure. The standard CPU has capacity

for up to 4K words × 16 bits of control memory and, optionally can have an alterable control memory of up to 16K words. Each 16-bit word in the control memory contains a single microinstruction.

The architecture of the MICRODATA 1600 CPU is illustrated in Fig. 2.1. The major processing facility of the processor is the arithmetic logic unit (ALU). It performs addition, subtraction, data transfer, logical AND, OR, exclusive OR and shifting. Inputs to the ALU are from the B bus and/or from the selected file register, and output is via the A bus to the selected destination register.

A status register consisting of file register 0, common to both the primary and secondary files, is used for various internal flags like overflow, negative or zero condition etc., generated by the ALU.

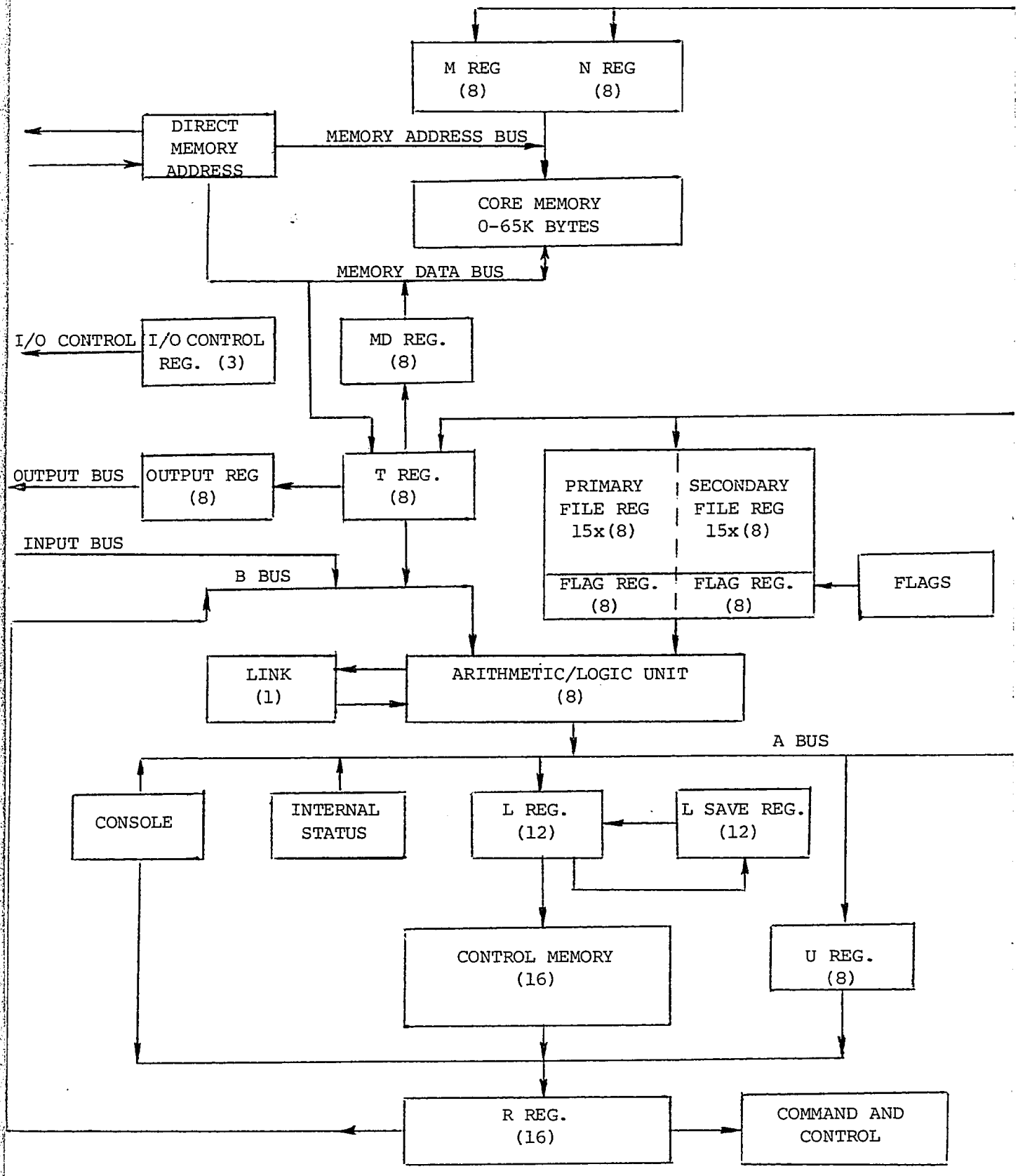
All registers except the file registers have specific functions in the machine. A brief description of the processor registers follows:

T Register:

The 8-bit T register serves as the operand register for most operate-type commands, and as a buffer for data being written into or read from core memory, and for output on the byte I/O bus.

M and N Registers:

The 8-bit M and N registers hold the current 16-bit address of the location being accessed in core memory. The M register holds the 8 high order bits of the address, and the N register holds the eight low order bits.



MICRODATA 1600 CPU BLOCK DIAGRAM

Fig. 2.1

U Register:

The 8-bit U register is used to modify the 8 high order bits of the control memory output.

File Registers:

Two files (Primary and Secondary) of 16 registers each provide storage for internal flags and user's data. Register 0 (in both files) is available to the user and is used for internal flags.

L Register:

The 12-bit L register holds the address of the next microinstruction to be read from control memory.

L-Save Register:

The 12-bit L-Save register saves the incremented contents of the L register when a Jump Extended microinstruction (JE) is executed, unless storage was inhibited by a previous Inhibit L-Save microinstruction (ILS).

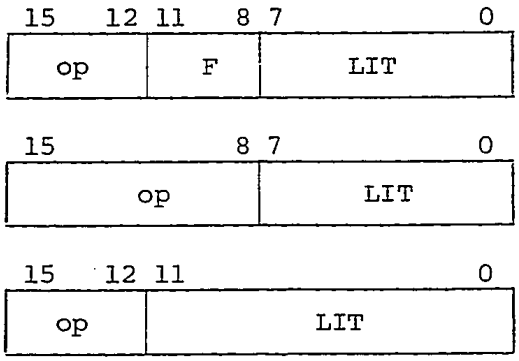
Link Register:

The 2-bit Link register contains storage for an arithmetic link bit and a memory link bit.

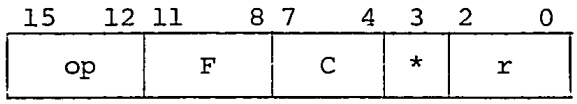
Microinstruction Formats:

There are five basic formats used for the microinstructions.

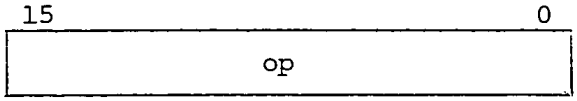
(a) Literal Commands



(b) Operate Commands



(c) Generic Commands



where OP : Operation Code

F : File Register Designator

C : Control Field

LIT: Literal

* : File Inhibit

r : Destination Register Designator

Lists of the NOVA instruction set and the MICRODATA 1600 microinstruction set are provided in Appendix A and B respectively. For more detailed information about both systems, the reader is referred to the manufacturer supplied documents [7] [8].

2.3.3 NOVA Emulator on the MICRODATA 1600

In order to map the NOVA machine onto the MICRODATA system, the following items should be considered:

1. Mapping of NOVA registers onto MICRODATA registers: This step is shown in Table 2.1.

In the NOVA computer, the PC register is automatically incremented by one, after the execution of the current instruction or the effective address is loaded into the PC if a JMP or JSR instruction has been performed. MICRODATA primary file registers 4 and 5

Table 2.1

MICRODATA File Register Assignments for NOVA System

File bank	File Register	Corresponding NOVA Register
Primary	0	MICRODATA condition flags
	1 (bit 1)	Carry bit
	2	Upper byte of Instruction Reg. (IU)
	3	Lower byte of Instruction Reg. (IL)
	4	Lower byte of Program Counter (LL)
	5	Upper byte of Program Counter (LU)
	6	Lower byte of AC0 (A0L)
	7	Upper byte of AC0 (A0U)
	8	Lower byte of AC1 (A1L)
	9	Upper byte of AC1 (A1U)
	A	Lower byte of AC2 (A2L)
	B	Upper byte of AC2 (A2U)
	C	Lower byte of AC3 (A3L)
	D	Upper byte of AC3 (A3U)
	E	Lower byte of Operand Register (OL)
	F	Upper byte of Operand Register (OU)
Secondary	0	MICRODATA Condition flags (common to both files)
	1	Temporary Storage
	2	Temporary Storage
	3	Temporary Storage
	4	RTC Counter
	5	Selected Clock Frequency (Initial RTC Counter Value)
	6	TTO Buffer
	7	TTI Buffer
	8	PTR Buffer
	9	Temporary Storage
	A	Low Byte of Mask Register
	B	PTR Busy/Done Byte *
	C	ID Byte **
	D	RTC Busy/Done Byte *
	E	TTI Busy/Done Byte *
	F	TTO Busy/Done Byte *

NOTE: ** Bit 7 assigned as interrupt flag (IF bit)
Bit 7=1 interrupt flag on
Bit 7=0 interrupt flag off
Bit 6 assigned as NI indicator
Bits 0, 1, 2 and 4 are used to reflect the Done flag Conditions of TTO, TTI, RTC and PTR respectively (1 indicates Done flag on, 0 indicates Done flag off)

* Bit 7 indicates the Busy flag condition
Bit 7=1 Busy flag set
Bit 7=0 Busy flag clear
Bit 0 indicates the Done flag condition
Bit 0=1 Done flag set
Bit 0=0 Done flag clear

are assigned as the image of the NOVA program counter, and updated by a microprogram called LUP after execution of the current NOVA instruction.

Bit 1 of primary file register 1 is used to store the emulated carry bit to reflect the resulting state of the carry after execution of any arithmetic and logic instruction of the emulated system.

2. Memory Mapping:

NOVA family machines are 16-bit machines whereas the MICRODATA 1600 is a 8-bit byte oriented machine. Therefore, in order to represent a NOVA word, two bytes of MICRODATA memory are used. During instruction fetch, two consecutive bytes are retrieved from the MICRODATA memory and stored in a pair of file registers, corresponding to the NOVA instruction register.

After the operand address calculation for a NOVA instruction, the address is loaded in the operand register (MICRODATA file registers 14 and 15). The contents of the operand register are multiplied by two and the address is then transferred to M and N registers if memory read/write is required; otherwise the contents of the operand register are transferred to PC register if a JMP or JSR instruction has been executed.

Before the emulator is activated, the virtual NOVA machine must be initialized by placing initial values into the emulated PC and other registers. At the same time, I/O device interfaces requiring setups - e.g., record format control, baud rate control, and so on must also be initialized. Once emulation mode is entered, the MICRODATA central processor behaves as if it were the emulated

(NOVA) central processor.

The FET microsubroutine fetches the simulated PC and converts the PC value to the address of the storage cell that contains the instruction to be interpreted; it then retrieves the NOVA instruction from memory and places the subject instruction in the emulated Instruction Register (primary file registers 2 and 3). The NOVA instruction is then interpreted by the emulator. Some of the microroutines used for this purpose are MRE, IOA, IDA etc. After the NOVA instruction is executed and the PC is updated, control passes to the INT microsubroutine which handles internal housekeeping e.g., I/O data transfer, and interrupt handling. If no I/O interrupt is detected then the next NOVA instruction is fetched and the whole cycle repeated.

The FET and INT microsubroutines are the most significant routines in the emulator, and are used once for each NOVA instruction. The basic flow-chart and the memory mapping diagram for the overall emulation process are shown in Figs. 2.2 and 2.3 respectively. A detailed flow-chart for the MICRODATA emulation of NOVA LDA and STA instructions is shown in Fig. 2.4.

Table 2.2 lists the emulator execution times for the NOVA instruction set, along with the corresponding timings on a real NOVA.

2.4 Conclusion:

It is clear from Table 2.2 that there is a performance penalty to be paid for using the MICRODATA 1600 to emulate the NOVA. This

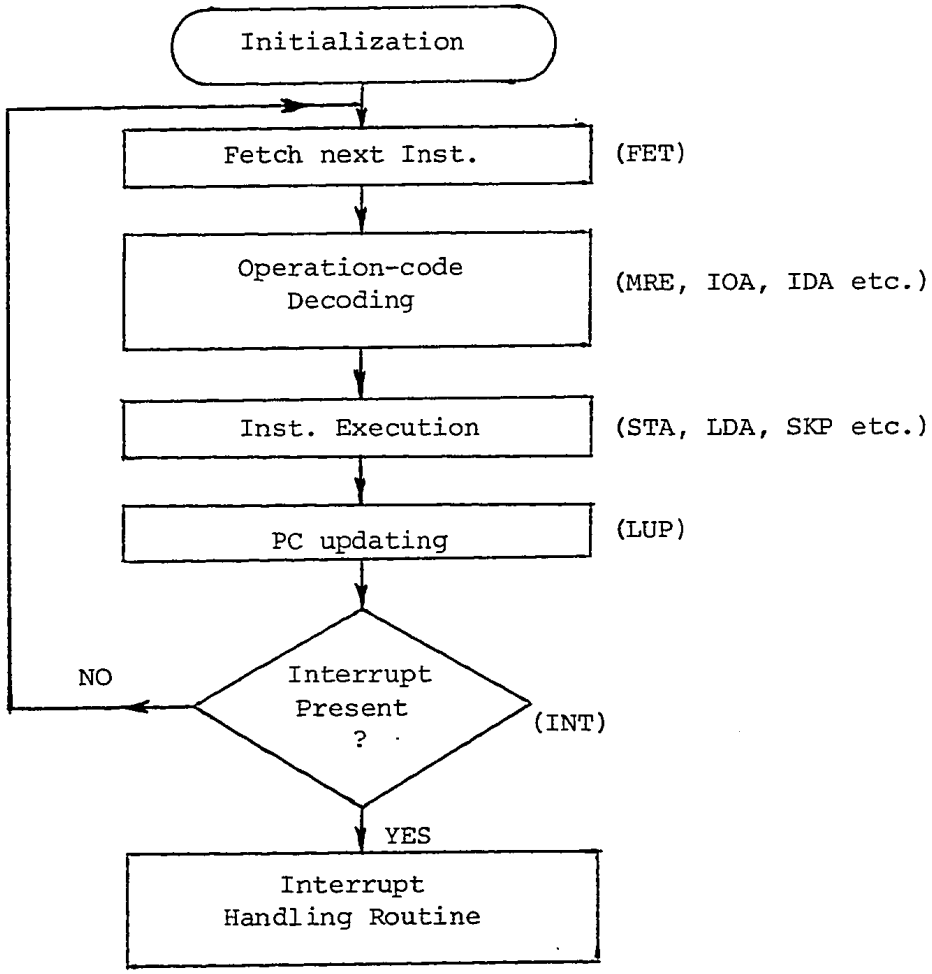
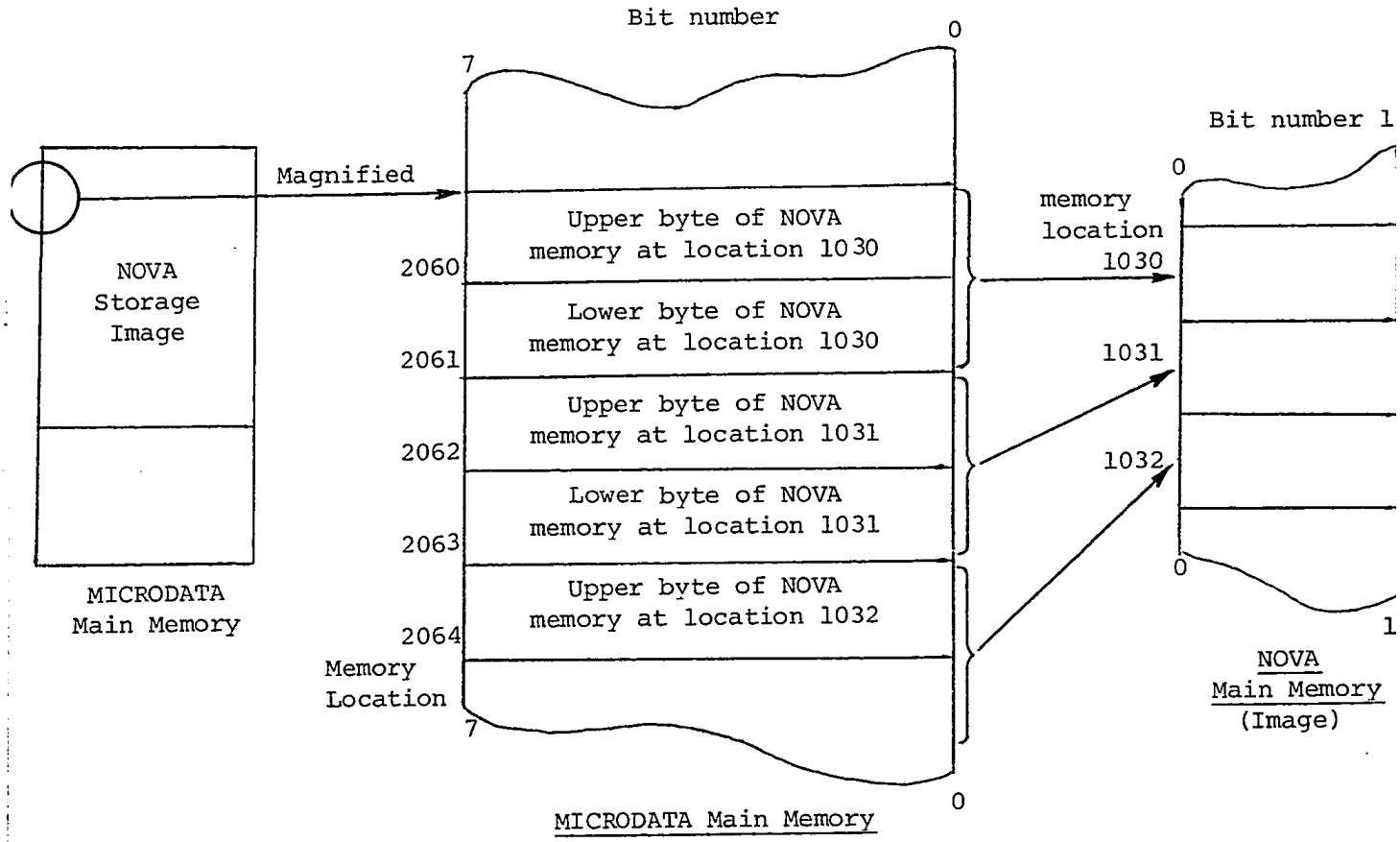
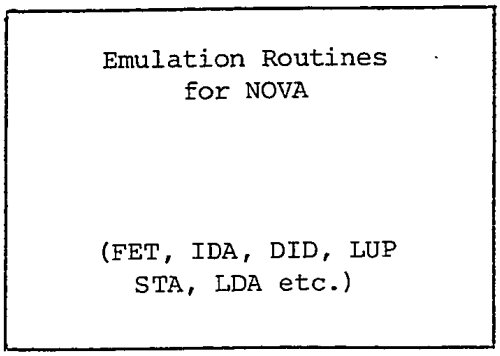


Fig. 2.2 The Basic Flow-Chart of NOVA Emulator





MICRODATA Control Memory

Fig. 2.3 The Memory Mapping Diagram of NOVA Emulator

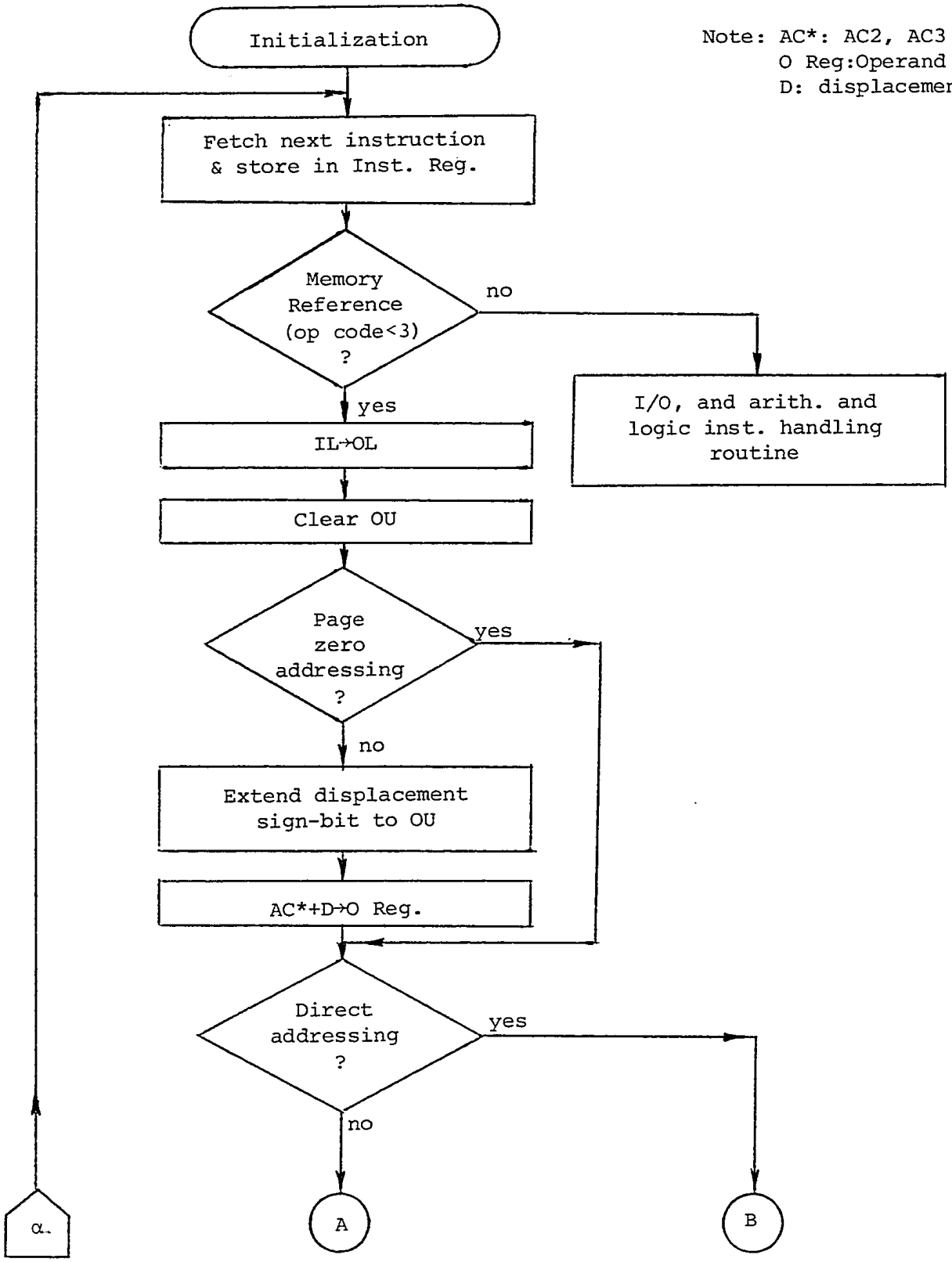


Fig. 2.4 Flow-Chart for the MICRODATA Emulation of NOVA LDA and STA Instructions

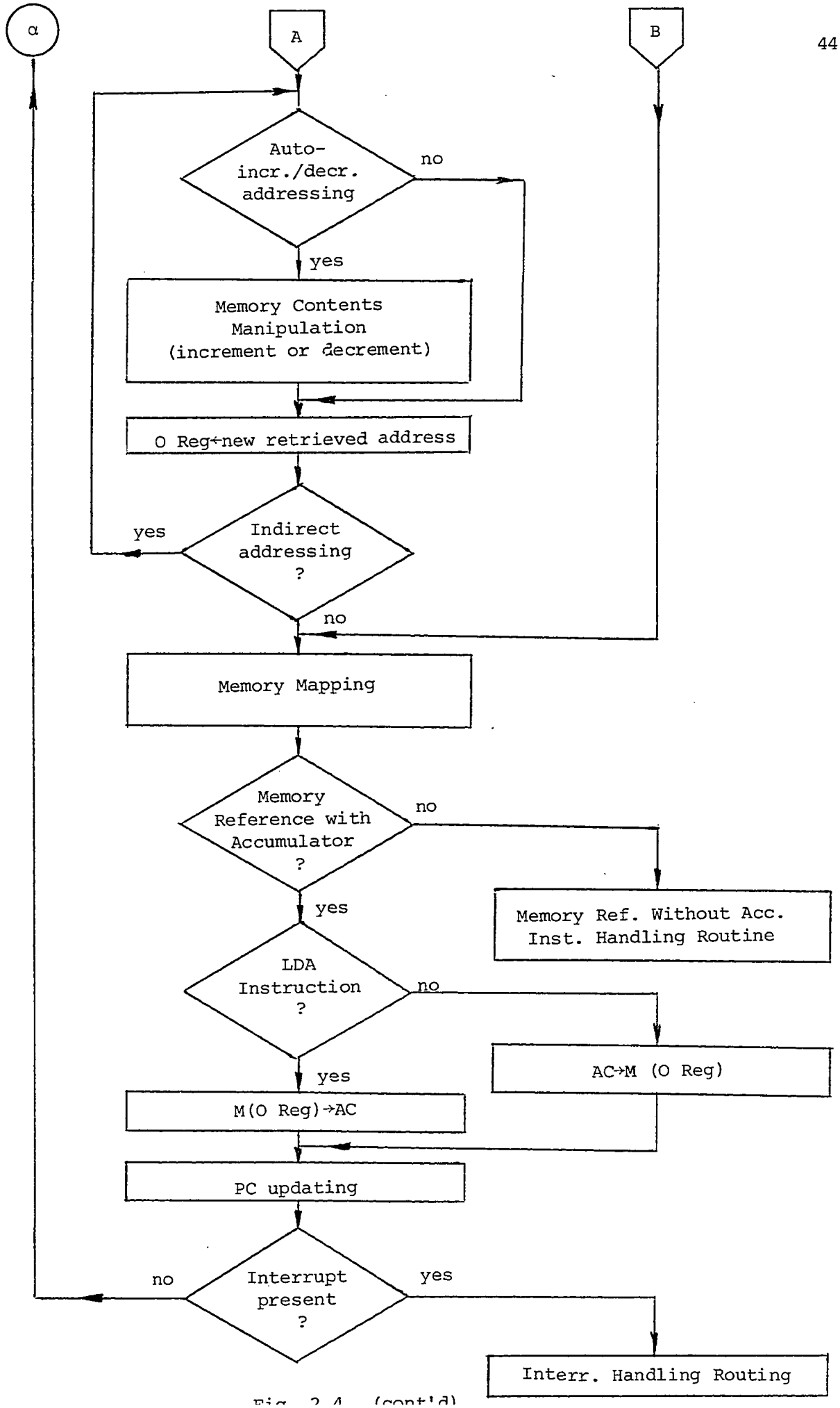

α
A
B
Auto-incr./decr. addressing
no
yes
Memory Contents Manipulation (increment or decrement)
O Reg←new retrieved address
Indirect addressing ?
yes
no
Memory Mapping
Memory Reference with Accumulator ?
no
Memory Ref. Without Acc. Inst. Handling Routine
yes
LDA Instruction ?
no
AC→M (O Reg)
yes
M(O Reg)→AC
PC updating
Interrupt present ?
no
yes
Interr. Handling Routing

Fig. 2.4 (cont'd)

Table 2.2

Execution Times for the NOVA Emulator

and the NOVA 1200 (in microseconds)

Instruction		Emulator	NOVA 1200
1. Memory Reference Instructions (Direct Page Zero Addressing)			
	JMP	5.0	1.35
	JSR	6.0	1.35
	ISZ	12.0	3.15
	DSZ	12.0	3.15
	LDA	9.0	2.55
	STA	9.4	2.55
2. Arithmetic and Logic Instructions			
	COM	16.8	1.35
	NEG	18.0	1.35
	MOV	16.8	1.35
	INC	18.0	1.35
	ADC	18.0	1.35
	SUB	19.4	1.35
	ADD	18.6	1.35
	AND	18.4	1.35
3. Input-Output Instructions			
	SKP__ TTI	8.4	2.55
	NIOS TTI	9.0	3.15
	NIOC TTI	9.6	3.15
	DIAS AC,TTI	10.8	2.5
	DIAC AC,TTI	11.4	2.5
	SKP__ TTO	8.6	2.55
	NIOS TTO	15.2	3.15
	NIOS TTO	11.4	3.15
	DOAS AC,TTO	16.6	3.15
	DOAC AC,TTO	12.8	3.15
	SKP__ PTR	8.6	2.55
	NIOS PTR	27.2	3.15
	NIOC PTR	11.4	3.15
	DIAS AC,PTR	29.4	2.55
	DIAC AC,PTR	13.6	2.55
	SKP__ RTC	8.6	2.55
	NIOS RTC	9.0	3.15
	NIOC RTC	9.8	3.15
	DOAS AC,RTC	13.8	3.15
	DOAC AC,RTC	14.6	3.15

is somewhat inevitable, since the two systems employ different design philosophies. For every NOVA instruction and operand, the MICRODATA must go through two cycles of memory addressing. In addition, the NOVA instructions are implemented for optimum utilization of NOVA hardware. But on the MICRODATA 1600 , the highly condensed information contained in the NOVA instructions has to be interpreted by a significant piece of microcode. Furthermore, the MICRODATA 1600 is not a high speed microprogrammable computer (one microinstruction is executed every 200 nsecs.), so the decoding and execution of target instructions takes a significant amount of time compared with the hardwired NOVA 1200 system.

Several instructions were timed and are listed in Table 2.3 in order to compare the speed of the NOVA emulator with the MICRODATA instructions which perform similar operations. The average ratio of execution times for these instructions between the MICRODATA native mode and NOVA emulation mode (except JMP and JSR inst.) is approximately 1:1.5. If the NOVA system is simulated at software level on the MICRODATA the execution times for the simulator will be much longer compared to the execution times for the firmware emulator.

Apart from the difference in data path widths, the processor functions of both the target and the host systems are very similar. Some improvement in emulation speed would result if the MICRODATA 1600 were a 16-bit machine, i.e., if its memory word size and data paths were 16-bit wide. For example, target instruction fetch would then require only one memory cycle instead of the present two. However, the process of interpreting the target machine

Table 2.3

Execution Times for Similar Instructions on NOVA Emulator and MICRODATA 1600/30 (in microseconds)

Instruction	NOVA Emulator	MICRODATA 1600/30 [9]
JMP	5.0	5.2
JSR	6.0	6.8
ISR	12.0	7.8
DSR	12.0	7.8
LDA	9.0	5.8
STA	9.4	6.6

instructions would more or less remain the same. Therefore, the overall speed of the emulator would not be twice the speed of the present emulator; however, there will definitely be some improvement in speed.

CHAPTER 3

EMULATION OF I/O SYSTEMS

3.1 Introduction

By means of the input/output (I/O) system and peripheral devices, the computer can communicate with the outside world, and the user can access the resources within the computer system. Peripheral devices are usually asynchronous to the processor and may be widely different in their speeds of operation, so some form of "hand shaking" or protocol is necessary for processor-device communication.

In order to identify an individual device, each device attached to a computer is assigned a device address which can be unique or shared by more than one device. The device address field usually consists of two parts: a device controller (interface) address and a subaddress indicating the selected device among the identical units connected to the controller. A device is selected when the device controller recognizes its address; then the device is able to transfer data as specified by the computer program.

Data transfer between a peripheral device and the processor may be affected either under program control or under the control of the external device. If the data transfer is affected under program control, transfer occurs whenever an I/O instruction is issued by the program. Usually the program examines the status of the external device until it is ready to receive or transmit data, then the I/O instruction is sent.

Interrupt technique is another means to handle data transfer

that is independent of the computer program and the data are transferred at the clock rate determined by the external device. In fact, the interrupt may be regarded as a mechanism for synchronizing the central processor with its external environment.

Just as it is necessary to create a mapping of target CPU facilities in the host CPU resources, it is also necessary to create a similar mapping of target I/O system in the host I/O facilities. The overall effect of the execution of target machine commands on the host machine should be to produce the desired end result. In the emulation of central processor instructions, the processor state transitions are well-defined and resources are localized to a set of local storage or memory locations; furthermore, there are no timing dependencies in the process. Unfortunately, none of these factors apply to the emulation of I/O operations. The order of state transitions during I/O operations depends on the timing relationships between the particular processor and its I/O devices, as well as the final state of the machine is not determined until the completion of the I/O operations. This, along with the differences in the interrupt structures of the host and the target machines, makes I/O emulation much more difficult than the emulation of central processor instructions.

3.2 I/O Emulation Process

Before discussing the emulation of I/O instructions in detail, it is appropriate to examine the factors that influence the mapping of target I/O resources onto those of the host machine.

3.2.1 I/O System Structure

There are two basic I/O system structures used in most computer systems.

A. Single Bus Structure:

In this structure a single bus interconnects all system elements as shown in Fig. 3.1.

The data transmission protocol between any two elements of the system is identical, and data can be transferred between any pair of elements. This makes it possible for two input/output devices to communicate with each other or for input/output devices to send or receive data directly to or from memory without processor intervention. This scheme offers a single type of interface for all system modules, and the processor can address a peripheral device in the same way that memory locations are addressed, so no special I/O instructions are needed. The PDP-11 organization is an example of this approach.

B. Multiple Bus Structure:

This is the most common computer structure, and a typical example of this scheme is shown in Fig. 3.2. A separate I/O bus is used for each input/output method employed. A program I/O bus is used for devices that use interrupt based or programmed I/O. A DMA (Direct Memory Access) bus is used for devices that transfer blocks of data to or from memory at high speed. The protocol signals for each bus are designed according to its I/O transfer features.

Obviously, the I/O bus structures of the host and the target systems play an important role in I/O emulation. If the I/O struc-

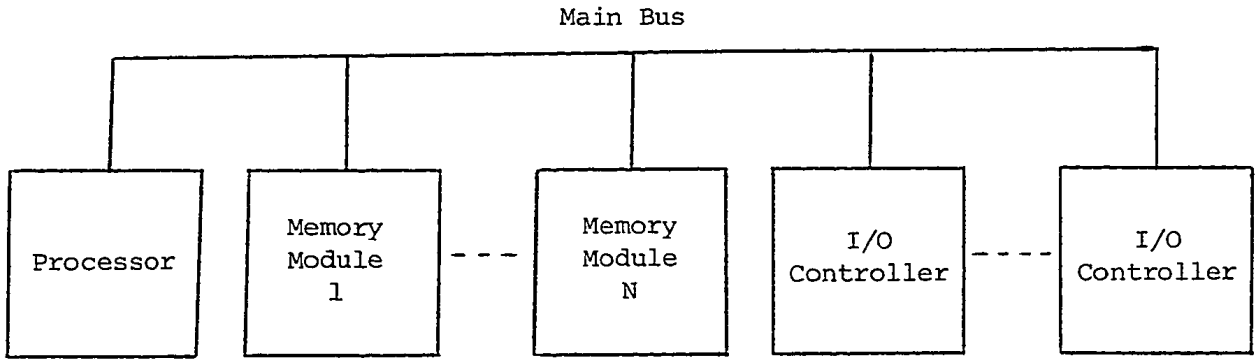


Fig. 3.1 Single Bus Structure

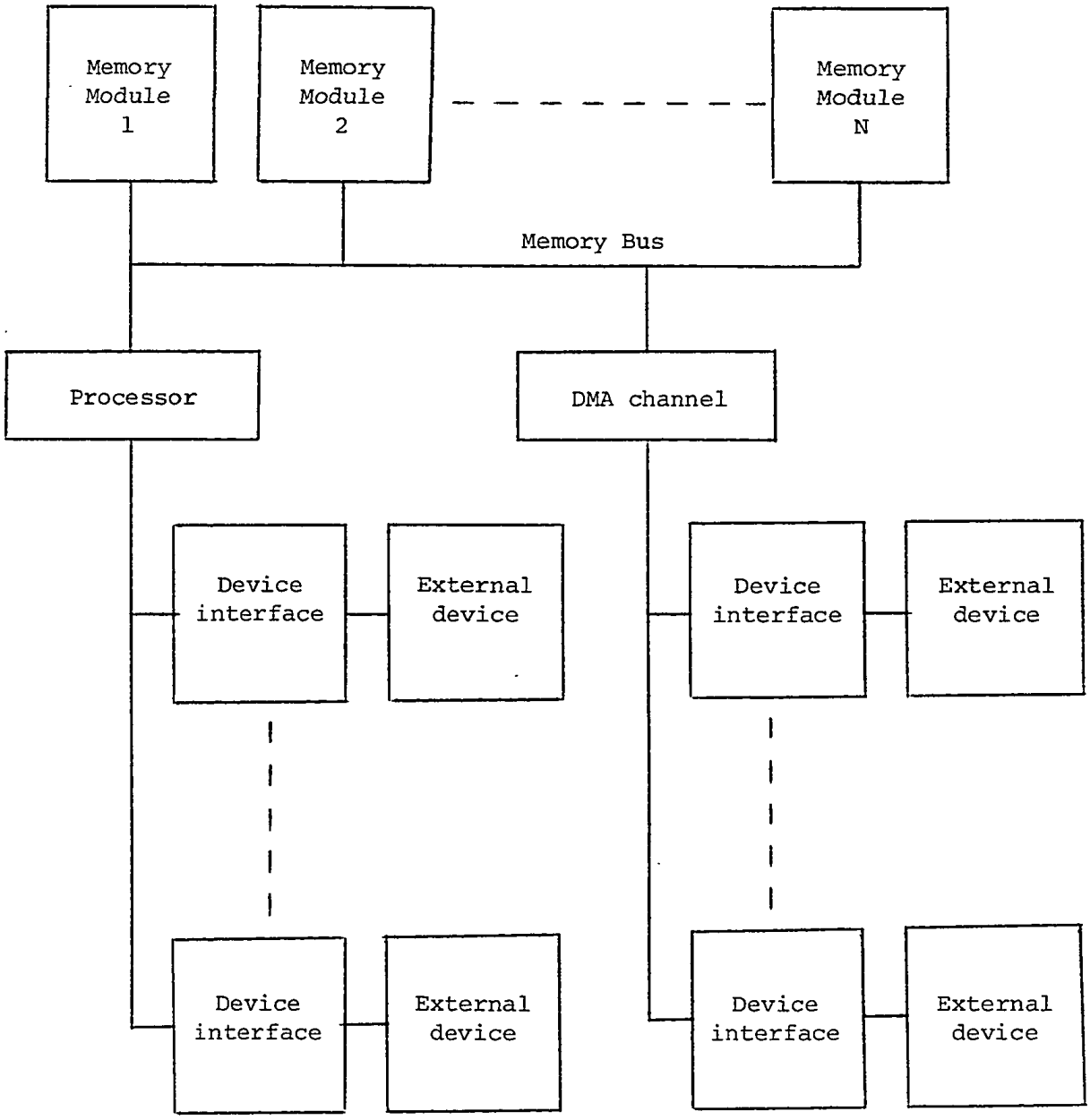


Fig. 3.2 Multiple Bus Structure

tures are different, then it is necessary to simulate the target bus structure on the host system and to interpret the target I/O protocol signals properly. For example, if a multiple bus system has to emulate a single bus target machine, then besides the firmware implementation, added hardware or front-end processor may be necessary to assist the simulation. On the contrary, if a single bus host system has to emulate a multiple bus target machine, the emulator I/O performance is highly degraded because of the nature of the target I/O bus structure.

3.2.2 Interrupt Structures

On some computers the interrupt facilities are very basic. The task of saving the contents of processor registers and/or memory cells, determining the cause of interruption, and establishing the service routine priority are left to the programmer. On more sophisticated computers, some or all of these tasks may be directly achieved by hardware, so the interrupt handling is more efficient from the programmer's viewpoint. Typically, in such an interrupt system, when an interrupt occurs, it interrupts the running program on completion of the current instruction and inhibits further interrupts. Processor hardware automatically saves the contents of the incremented program counter in a predeterminated memory location, then causes a jump to a specific trap location which contains the starting address of the main service routine. (Typically, location zero may be used to store the program counter and location one for the trap address.)

The main service routine must save the contents of processor

registers, identify the peripheral device which caused the interrupt, and branch to the appropriate service routine for that device. After the device is serviced, the main service routine must restore the registers, enable interrupts, and resume the interrupted program.

In a more sophisticated interrupt system, the device service routines may be interrupted by other interrupts of higher priority, and each service routine will independently preserve and restore processor registers and return links. Interrupt handling in these systems is usually nested in a priority sequence, so a stack is generally used for queueing the suspended lower priority activities. After the higher priority routine is completed, it must restore the processor registers, and access the return link from the stack in order to continue the interrupted process.

In a microprogrammable computer, interrupts are usually "soft" rather than "hard"; that is, when an interrupt occurs, it only sets a flag to signal the processor instead of executing any hardware functions like trapping the program into a specified memory location or saving the contents of program counter, etc. The interrupt is subsequently processed by a microprogram according to the defined interrupt structure of the system. Therefore, in I/O emulation the interrupt handling strategy defined by the target machine can be directly implemented by firmware. However, added hardware may be involved in order to perform certain functions strictly under hardware control instead of performing these by sequences of micro-instructions in order to reduce the firmware complexity or improve the system performance on the host system.

3.2.3 Input/Output Instructions

The analysis of target I/O commands is essential in order to properly interpret each meaningful command bit or field so that equivalent actions can be executed on the host system. A typical I/O instruction format is shown as follows:

Class Code (I/O Operation)	Operation Code	Operand	Device Select Code

The class code specifies an input/output class of instruction. The device select code identifies which external device is being addressed, the operation field specifies the action to be taken, while the operand field specifies the data or command that is transferred via a selected register or memory location from (to) the processor to (from) the external device. In some target systems, there may be an I/O status word associated with the I/O operations. In this case, it is essential to analyze the status word in order to properly emulate its function on the host system.

3.2.4 Data Representation and Record Structure:

Host and target machine incompatibility in data representation for I/O is an important factor to be considered in I/O system emulation. For example, the host system may require the I/O data to be in EBCDIC form, whereas the target system may require ASCII representation for the same purpose. Therefore, the emulator must convert the input/output data into the appropriate representation.

The most straightforward scheme for this conversion is table look-up. Some saving in the use of control store can be made by

storing the conversion table in main memory. Firmware/software tradeoff may also be achieved by writing a software subroutine which is executed by the host to provide the code conversion.

The structure or format of I/O records for various peripheral devices is another important factor that affects I/O emulation. For example, the formats of disk records for the host and the target system may be different; again here, the emulator must create a virtual compatibility between the two systems.

3.2.5 Time Dependencies in I/O Processing

Many programs written for older systems use programming techniques which depend on the timing relationships between the central processor and peripheral devices [2]. The most common type of time dependent programs assume that the processor will be able to execute a certain minimum number of instructions before an I/O operation reaches a given stage. It is almost certain that the host machine and its peripherals do not have the same timing relationships as do the target machine and its peripherals. Once again, the emulator must ensure the virtual preservation of these timing relationships so that time dependent target programs are executed properly.

3.3 Emulation of NOVA I/O System on the MICRODATA 1600

In this section, NOVA I/O system emulation on the MICRODATA 1600/30 is described in detail ; the purpose of this is to show the detailed mechanisms of I/O emulation. However, before discussing

this I/O emulation example any further, it is appropriate to introduce the I/O systems of both the NOVA and the MICRODATA 1600 in order to put the emulation details in the proper perspective.

3.3.1 MICRODATA 1600 Input/Output System

The MICRODATA 1600 system provides an extremely fast elementary input/output capability. The data paths and control functions are simple elements that can be sequenced from the control memory. Microprograms in the control memory can implement facilities with a high degree of versatility in timing, data paths and I/O capabilities.

The I/O facility consists of:

A. A three-bit I/O Control Register.

B. A byte input bus and input control lines.

C. A byte output bus and output control lines.

A brief description of these facilities follows:

A. Input/Output Control Register

The I/O Control Register can be loaded or cleared by the I/O control microinstructions; the contents of the I/O Control Register define an I/O bus mode. The byte I/O control instructions and control modes are shown in Table 3.1.

During the execution of I/O control microinstructions, the three low order bits of the C-field are placed in the I/O Control Register. The I/O Control Register output may be decoded to form individual control signals defining the type of transfer to be performed on the byte I/O bus (e.g. CIO, DIX, etc.) and the state of the serial interface output (e.g., SOX). Of the eight possible

Table 3.1

I/O Control Microinstructions and Control Modes

Command Format

15 14 13 12	11 10 9 8	7 6 5 4	3	2 1 0
Op. code=7	f	c	*	r

Mnemonic	Operand Field	C Field (in Hex.)	Mode or Contents of I/O Control Register	Control Activity
CIO*	f,(r)	8	0	Clear I/O Mode
COX*	f,(r)	9	1	Control Output
DOX*	f,(r)	A	2	Data Output
SOX*	f,(r)	B	3	Spare Output
CAK*	f,(r)	C	4	Concurrent Acknowledge
IAK*	f,(r)	D	5	Interrupt Acknowledge
DIX*	f,(r)	E	6	Data Input
SIX*	f,(r)	F	7	Stack Input

states of the I/O Control Register, one represents no activity on the I/O bus (CIO), three are output modes (COX, DOX, SOX) and four are input modes (CAK, IAK, DIX and SIX).

Whenever the value in the I/O Control Register is set to 4, 5, 6 or 7, the input bus is substituted for the T-register in any command (e.g., ADD, LOR, etc.) that selects the T-register or its complement.

B. Input Lines

The input data lines (byte input bus) are input to the B bus (ref. Fig. 2.1) to transfer data from a peripheral device to the CPU under firmware (microprogram) control. The input control lines are input to the bits of file register 0 (zero) to reflect the service request by an external device. The flag bits of file register 0 are defined as shown in Table 3.2.

C. Output Lines

The output data lines (byte output bus) originate with the output register (OD register, Fig. 2.1) to carry data from the CPU to an addressed peripheral, operated under firmware control. The output control lines originate with the I/O Control Register to generate the relevant signals (concurrent acknowledge, data output, etc.) for the external device.

Input/Output Microinstructions

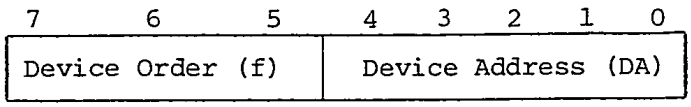
Before giving any detailed description of the I/O microinstructions, we examine the control byte that directs the basic function of the MICRODATA 1600 I/O system. The control byte provides a 3-bit device order and a 5-bit device address as follows:

Table 3.2

File Register 0 Flags

Bit Number	Description
0	Overflow Condition
1	Negative Condition
2	Zero Condition
3	Concurrent I/O Request (or Spare)
4	Internal Interrupt
5	I/O Reply * (or Spare)
6	Stack Overflow (or Serial TTY)
7	External Interrupt

* This flag is normally not used in MICRODATA 1600 I/O units.



Device Address

Each device on the byte I/O bus is assigned a unique 5-bit number or address. The numbers are assigned by means of selectively placed jumper wires on the printed circuit board of the device controller. The assigned device address is used by the device controller to compare against the device address of the control byte to determine if it is being addressed, and for identifying the device to the processor when requesting an interrupt or concurrent I/O transfer.

Device Order

The 3-bit device order specifies the type of I/O operation to be performed. Standard device orders designate the operations shown in Table 3.3. Order codes are shown with their standard assignments, but they may be changed depending on the individual interface requirements. The device order accompanies the device address, and is sent prior to each data transfer.

Status Byte

The 8-bit status byte input as the result of a status order (Ref. Table 3.3, order code = 1), has four bits whose functions are common to most devices; the four high order bits have device dependent functions. The functional meaning of the status bits is given in Table 3.4.

Interrupt Mechanism

The interrupt facility on the MICRODATA 1600 consists of:

TABLE 3.3

DEVICE ORDERS

ORDER CODE	OPERATION	DESCRIPTION
0	Data Transfer	A data byte will be transferred between the addressed device and the processor. Direction of the transfer will depend on whether the instruction is an input or an output.
1	Status/Function	A status byte will be input from the addressed device or a function byte will be output to the addressed device, depending on whether the instruction is an input or an output.
2	Block Input/INT	The addressed device will start a concurrent block input to memory and will generate an external interrupt at the conclusion of the transfer unless the interrupt has been subsequently disarmed. This order is normally sent by an output instruction.
3	Arm Interrupt	Permits the addressed device to make an external interrupt request upon the satisfaction of an interrupt condition. This order is normally sent by an output instruction.
4	Disconnect	The block transfer in progress by the addressed device is stopped and end of block interrupt will occur unless the interrupt has been disarmed. This order is normally sent by an output instruction.
5	Disarm Interrupt	Inhibits the addressed device from making an external interrupt request under any condition. This order is normally sent by an output instruction.
6	Block Output/INT	The addressed device will start a concurrent block output from memory and will generate an external interrupt at the conclusion of the transfer unless the interrupt has been subsequently disarmed. This order is normally sent by an output instruction.
7	Unassigned	This order, if assigned, may perform any required function as interpretted by the individual interface. If a byte transfer is desired the order may be sent by an input or an output instruction.

NOTE: This table should be used only as a general source of information. Documentation of specific controllers should be consulted before programming for that controller and related I/O device is attempted

TABLE 3.4

STATUS BYTES DEFINITION

BIT NUMBER	STATUS	DESCRIPTION
0	Ready	This bit is a 1-bit when the external device is in a ready state.
1	Input Flag	This bit is a 1-bit when the external device has a byte ready for input to the computer.
2	Output Flag	This bit is a 1-bit when the external device is ready to receive a byte from the computer.
3	Error	This bit is a 1-bit when an error has occurred during a transfer. Errors may be timing, or device malfunction. This bit is cleared when the status byte is input.
4-7		Device dependent

NOTE: This table represents typical controllers only.

1. Internal interrupt.
2. External interrupt and concurrent interrupt.

Before discussing the interrupt system, a brief description of the interrupt control microinstructions is given as follows:

DEI - Disable External Interrupts

This command causes the external interrupt system to be disabled. Interrupts are not lost when the interrupt system is disabled, but cannot be recognized by the processor.

EEI - Enable External Interrupts

The external interrupt system is enabled allowing the processor to recognize external interrupts.

DRT - Disable Real-Time Clock

The real time clock and interrupt are disabled.

ERT - Enable Real-Time Clock

The real-time clock and interrupt are enabled. The first interrupt will occur after a full interrupt interval.

Control Byte

Sending the control byte (arm/disarm for external interrupt; block input/block output for concurrent interrupt) via COX instruction, directs the individual external device controller to respond to the interrupt system.

As soon as an external device requests service or an internal interrupt is generated, it sets the pertinent bit in file register 0 and internal status byte to gain attention of the computer.

The internal status byte reflects the status of the internal interrupts. When an internal interrupt is present, bit 4 of file

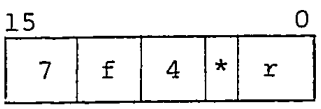
register 0 is set. The firmware normally responds by executing an Enter Internal Status (EIS)* microcommand which places the internal status byte on the A bus. The panel, real-time clock, and power fail interrupts are reset when the EIS command is executed. After sensing a flag bit ON, the corresponding internal interrupt routine is initiated to handle the interrupt. The format of the internal status byte is given in Table 3.5.

External interrupts are associated with peripheral devices and are used to indicate such conditions as data ready, error, or end of operation condition for a device. Individual interrupt may be handled by an external interrupt module which provides for arming/disarming individual interrupts and enabling/disabling recognition of interrupts.

The external interrupt system contains a single interrupt line, a priority line, and a select line; in other words the external interrupt system is organized in a daisy chain. A device may initiate an interrupt request when service is required only if it has the highest priority or priority has been received from higher

---

* Note: EIS Enter Internal Status



By executing this command, the eight internal status bits are placed in the file register designated by f, if * is zero, and in the register designated by r. The internal interrupt flag in file register 0 is reset by this command, along with the console interrupt, real-time clock interrupt, and power fail/restart status bits.

Table 3.5

Internal Status Byte

Bit Number	Status Meaning
0	Panel Interrupt
1	DMA Termination (or Spare)
2	Real Time Clock Interrupt
3	Spare
4	Spare
5	Spare
6	Panel Step Switch
7	Power Fail/Restart Interrupt

level interrupts on the priority chain. Devices not requiring interrupt service will propagate priority to the next device in line.

As an external interrupt occurs, it will automatically set bit 7 of file register 0 to signal the CPU accordingly. When the processor recognizes the interrupt signal, it enables the select line for the interrupt system. An IAK (Interrupt Acknowledge) microinstruction should be sent to respond to the interrupt.

By executing the IAK instruction a value of 5 is placed in the IC register, which enables the interrupt acknowledge signal, and device polling is started. Each device in order will interrogate the select line and, if not requesting, will propagate this signal to the next device in line. When the select signal is received by the requesting device, it will input its address on the I/O bus. This externally supplied address (ESA) has the following format.

7	6	5	4	3	2	1	0
0	0	Device Number					0

The interrupt acknowledge signal is not removed until a clear I/O command is executed. Upon removal of this signal the requesting device interface controller will reset the external interrupt request flag bit in file register 0.

Conventionally, (in MICRODATA supplied firmware), a band of memory locations is reserved for storing the addresses of individual external device interrupt handling routine (from location 0100 to 017F). The ESA is used to locate the appropriate interrupt handling routine address in memory. The address of the first of

two memory locations containing the interrupt subroutine address is formed by using the contents of ESA multiplied by 2 as the low order byte and 01 as the high order address byte. This is schematically shown in Fig. 3.3.

Concurrent Input/Output

The concurrent I/O allows for block transfers (e.g. transfer of 80 columns of data from a card reader) between an external device on the byte I/O bus and memory. Once started, it proceeds without program intervention until the whole block is transferred.

Concurrent I/O operations are started by executing the instruction COX with the proper control byte (device order code and address) in T register; for the standard controller, order code = 2 is used for concurrent block input; order code = 6 is used for concurrent block output. After a concurrent I/O operation is initiated, as the data (e.g. a column of data from the card reader) are ready for transfer to/from the memory from/to the specified peripheral, bit 3 of file register 0 is set to gain the attention of the processor. After sensing the concurrent interrupt flag, the CPU responds to the request by executing the CAK (Concurrent acknowledge) microinstruction.

During execution of the CAK command, a value of 4 is placed in the IC register which enables the concurrent interrupt acknowledge signal; polling of external devices starts in order to identify the requesting peripheral. When the requesting device is reached, it inputs an externally supplied address (ESA) on the byte I/O bus. The ESA is used by the processor to define the type of concurrent I/O operation request and to locate the appropriate address control

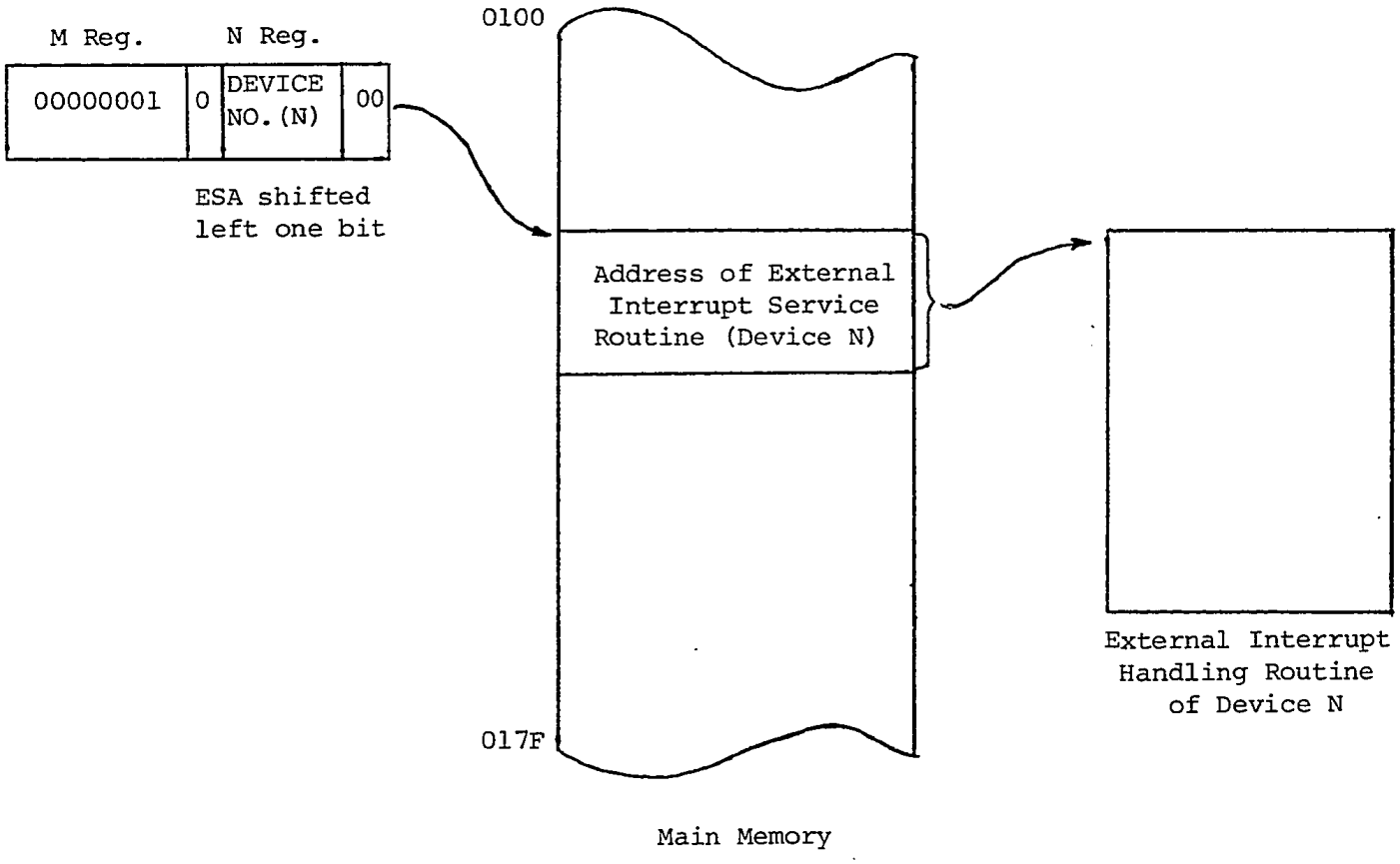


Fig. 3.3 Mapping of ESA to the External Interrupt Service Routine

words. The format of this ESA (different from external interrupt's ESA) is shown as follows:

7	6	5 4 3 2 1	0
I/O	0	Device NO. (DN)	0

ESA

bit 7 = 0, signifies an input transfer

= 1, signifies an output transfer

The concurrent acknowledge signal is not removed until a CIO command is executed. Upon removal of this signal, the requesting controller will reset the concurrent request flag bit in file register 0.

Conventionally, (in MICRODATA supplied firmware) concurrent I/O is directed by the following items:

1. ESA: Externally supplied address from the requesting device (described earlier).

2. Address Control: Concurrent I/O addresses for each external device are controlled by a pair of two-byte address words. These two words are located in memory starting at an address equal to four times the device number. The first word is the current address (CA) and contains the address of the next memory byte to be used for the transfer. The second word is the end address (EA) and contains the address of the last byte of the block. Conventionally, the first 128 locations in memory are reserved for storing concurrent I/O addresses for control of up to 32 external devices. The four bytes for each device have the following format:

	15 0
4*DN	Current Address (CA)
4*DN+2	End Address (EA)

Fig. 3.4 shows schematically how the ESA is mapped to obtain the current address and the end address for concurrent I/O.

After a concurrent I/O operation is initiated, byte transfer proceeds automatically until the last byte of the block is transferred. Following each transfer, the current address is incremented by one. When the current address (CA) is greater than the end address (EA), the disconnect order code should be sent to the device; this order code causes the concurrent I/O operation to cease, and the block transfer is finished.

3.3.2 NOVA Input/Output System

The I/O system of the NOVA computer consists of two main parts: input/output bus, and device controllers.

A. Input/Output Bus

The I/O bus is the communication link between the CPU and input/output devices. It consists of:

1. 6 device selection lines used to address the selected device. Only the selected device responds to control signals generated during an I/O instruction execution.
2. 16 bidirectional data lines. Via these 16 lines all data and addresses are transferred between the processor and the devices attached to the bus.
3. 19 control lines from the processor to devices govern the synchronization of all transfers on the data lines, start and stop device operations, and control the program interrupt and data channel.
4. 6 control lines from a device to the processor. Over these

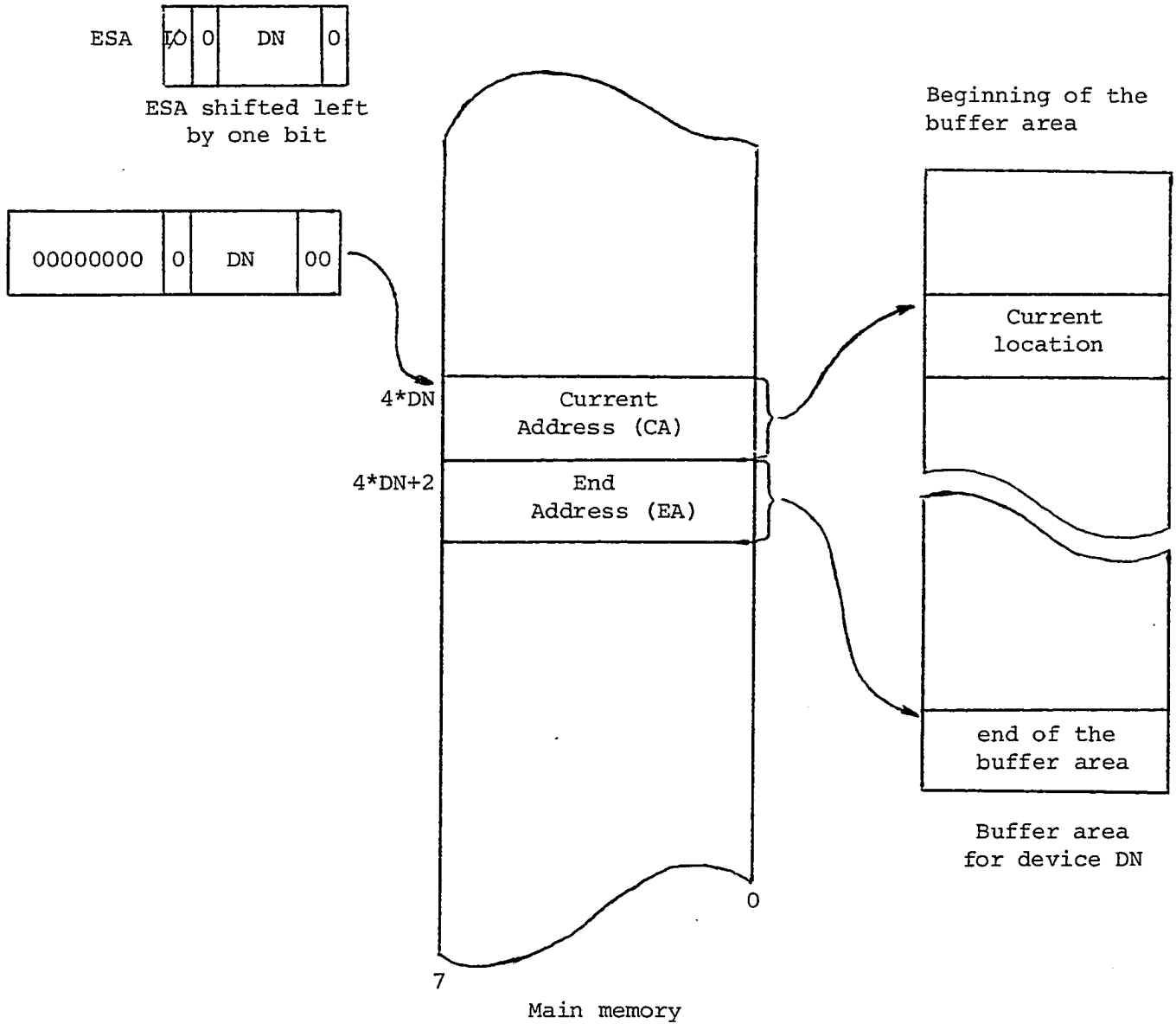


Fig. 3.4 Mapping of ESA to CA and EA in Concurrent Interrupt

control lines a device can indicate the state of its "Busy" and "Done" flags, and request a program interrupt or data channel access.

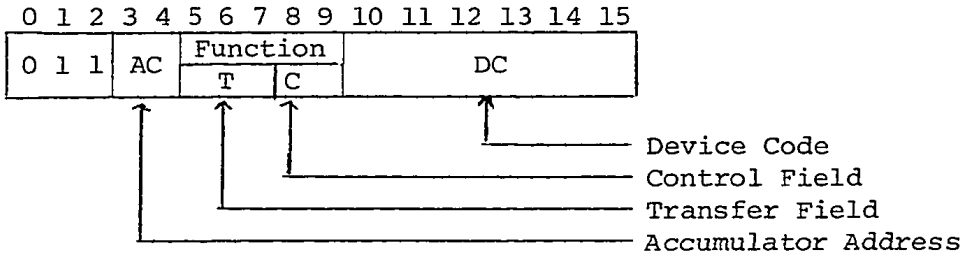
B. Device Control Unit

Every device connected to the I/O bus has certain fundamental circuit networks used to communicate with the processor to reveal its current status or to ensure the device will respond when it is selected.

Each device has a 6-bit device code which may be unique to it or be shared by more than one device. A device will respond when and only when it is addressed by the CPU over the I/O bus. There are "Busy" and "Done" flags associated with each peripheral device to denote the device state and interrupt request. When both "Busy" and "Done" are clear the device is idle. As the program sets the "Busy" flag it places the device in operation. When the device has processed a unit of data, it clears "Busy" and sets the "Done" flag to indicate that it is ready to receive new data for output or it has data ready for input. If the "Interrupt Disable" flag is clear and "Interrupt On" is set, the setting of the "Done" flag will signal the program by requesting an interrupt.

Input/Output Instruction Set

Instructions in the input/output class direct data transmission from and to the peripheral devices, and also perform various operations within the processor. The format of an I/O instruction is shown below:



If the transfer field contents are 0 or 7, there is no data transfer between a device and the CPU. In this case the control field specifies a control or skip function respectively. If any number from 1 to 6 is assigned to the transfer field, it will select one of three buffers in the device and govern data transfer between the accumulator addressed by bits 3 and 4 (AC field) and the device specified by the DC field. Device code 00 is not used and 77 is used for a number of special functions.

Interrupt Facility

The interrupt facility of NOVA provides for enabling and disabling of devices from requesting service, establishing levels of priority interrupts, and servicing devices only when they request service. The interrupt system is enabled by the instruction INTEN, which sets the "Interrupt On" flag to allow the processor to respond to interrupt requests; it is disabled by the instruction INTDS, which clears the "Interrupt On" flag to prevent the processor from responding to further interrupt requests. Interrupt request by a device is governed by its "Done" and "Interrupt Disable" flags. When a device completes an operation, it sets the "Done" flag, and this action requests a program interrupt if "Interrupt Disable" is clear. The service request signal is a level signal, so once it is set, it will remain on the I/O bus until the program clears the

"Done" flag or sets "Interrupt Disable". If the program does set the "Interrupt Disable" flag in a device by the instruction MSKO, that device cannot cause an interrupt when its "Done" flag is set, and any request it may have already made is disabled; however, if the "Done" flag is left set, clearing "Interrupt Disable" will restore the interrupt request.

At the beginning of every memory cycle, the processor tests for any interrupt request. If a request does exist, the processor clears the "Interrupt On" flag, thus disabling any further interrupts, then saves the contents of the program counter in memory location 0. It then executes an indirect jump through location 1 (hardware function). Therefore, location 1 should contain the address of the interrupt service routine or an indirect address that will get it there.

The interrupt service routine has to determine which device is requesting interrupt. This is done either by using a polling technique which involves checking the "Done" flag of every device in descending order of priority, or by a broadcasting technique which uses the INTA AC (Interrupt Acknowledge) instruction. When this latter command is issued, the device requesting an interrupt, which is physically closest to the processor, responds with its device code which is deposited into the specified accumulator.

Once the interrupt service routine has determined the device to be served, it executes a jump to the specific device service program. Generally, a device service routine will save the contents of any accumulators, carry, or memory cells that may be destroyed by the service routine. The service program can simply

leave the interrupt off while serving the device by leaving "Interrupt On" clear, or it can enable interrupt and establish a priority structure that allows a higher priority device to interrupt the current device service routine. This priority is established by a new MSKO instruction that controls the states of the "Interrupt Disable" flags in the various devices. After serving a device, the device service routine should restore the processor to its original state prior to the interrupt, turn on the interrupt and jump to the interrupted program to continue processing.

The device priority mechanism of NOVA system is very flexible. There are several ways in which priorities are determined for or assigned to a device on the I/O bus. The most significant method is by specifying which devices can interrupt a service routine currently in progress. This is done through the use of a MSKO AC command. The "Interrupt Disable" flag of each device is wired to a particular data line on the bus and is effectively connected to one of the 16-bit positions in the accumulator AC. If the bit position in the AC contains a 1, all "Interrupt Disable" flip-flops connected to it are set, thus disabling the devices from requesting interrupts. If, however, the bit position contains a 0 (zero), all "Interrupt Disable" flip-flops connected to it are reset. This enables the corresponding devices to request interrupts and, therefore, these devices are regarded by the program as being of higher priority. By means of the MSKO instruction, programs can establish any priority structure, and because accumulator AC has 16 bit positions, there are 16 possible levels of interrupt priorities.

For further information on both systems, the reader is referred

to the manufacturer supplied literature [7], [8], [9].

3.3.3 NOVA I/O Emulation on the MICRODATA 1600

The I/O instruction set of NOVA computers can be classified into two types:

1. Peripheral device oriented instructions (device codes range from 01_8 to 76_8).
2. Special function instructions (device code = 77_8).

In the former class of instructions, the programmer can select the device buffers (named A buffer, B buffer, and C buffer) for holding the information being transferred. The number of device buffers available to the programmer is device dependent. Typically, all the simple data handling devices, such as Teletype or paper tape reader, have only the A buffer. In order to emulate this feature, local storage is used to simulate the device buffers for an individual device. For instance, the MICRODATA secondary file register 6 is assigned as the Teletype output (TTO) A buffer.

Similarly, the "Busy" and "Done" flags of an individual device are mapped into the host system to indicate the status of the emulated device to the emulator. The major resources of NOVA I/O system are mapped to most of the secondary file registers on the MICRODATA system. This mapping is shown in Table 2.1.

Besides these basic mappings, there is an additional virtual match created for the NOVA real time clock (RTC) instruction. The real time clock rate of MICRODATA 1600 is fixed at 1000HZ However, in the NOVA system it is possible to select one of four clock frequences: AC line frequency (60 HZ), 10 HZ, 100HZ and 1000HZ.

Therefore, for emulating NOVA RTC frequencies other than 1000HZ, a counter has to be set up in order to generate the RTC interrupt at the selected rate.

In NOVA the special function instructions facilitate various operations within the processor such as interrupt enable/disable, or read switch etc. Instructions for testing interrupt on or off conditions (SKPBN CPU, SKPBZ CPU) are provided in the repertoire. But there is no such facility in the MICRODATA system; the NOVA interrupt flag (IF) has to be emulated. Bit 7 of secondary file register 12 is assigned for this purpose. The IF bit can be used to distinguish whether the emulated system (NOVA) is actually in the interrupt mode or program mode in order to govern the emulator to take proper actions (whether to emulate NOVA interrupt mode operation or not). More details on this aspect are provided in the next section.

In the NOVA system, the interrupt enable command cannot take effect until the next instruction has been executed, allowing the return jump to be made successfully after the interrupt service routine is finished. This feature should also be faithfully incorporated in the emulator. Bit 6 of secondary file register 12 (NI - No Interrupt) is used to indicate whether a NOVA interrupt is to be serviced or not. (If NI = 0, then NOVA interrupts are serviced, otherwise they are not serviced.)

In the present version of the emulator, a basic NOVA system has been emulated. It includes the emulation of CPU and some fundamental I/O devices: Teletype input/output, high speed paper tape reader and real time clock. In NOVA, reassignment of priority

levels for these devices can be achieved by specifying a mask pattern in an accumulator which is then transferred to the mask register by a MSKO instruction. The high order byte of the mask register contains the mask priority bits for the disk pack and 16 line TTY multiplexor; the low order byte contains the mask bits for other devices e.g. Teletype, paper tape reader etc. Since the present version of the emulator does not emulate the NOVA disk pack or TTY multiplexor, a MICRODATA file register (secondary file number 10) is sufficient to emulate the low order byte of the mask register.

Before the emulator is entered into the emulation mode, it is necessary to initialize the host (MICRODATA) I/O system. Establishing an individual peripheral device's character format, baud rate control, and processor interrupt enable place the host I/O system in its operating mode. Subsequent initialization of the virtual system (NOVA) will clear the control flags (e.g. "Busy"/ "Done" flags of all devices, NOVA interrupt flag etc.). The NOVA program counter is then initialized to the appropriate starting address after which the emulator begins its major function by fetching a target instruction from the main memory and interpreting the instruction. The INT microroutine is executed immediately after the execution of each NOVA instruction. The major functions of this routine are to examine the status of the I/O devices, carry out any data transfer, and update the status flags of emulated I/O devices. If input data from a device is ready, the INT routine will issue I/O microinstructions to accept the data from the device and store it in the simulated device buffer. On

the contrary, if data is to be output to a specified device, the actual output operations are also carried out by the INT micro-routine when the device is ready to receive data. Flow-charts for the INT routine and NOVA TTI instruction emulation are shown in Fig. 3.5 and Fig. 3.6, respectively.

3.4 Summary

It is often the case that the peripherals of the host and the target systems are not compatible. In the simple case, achieving virtual compatibility may involve data conversion and/or record reformatting as mentioned in Section 3.2.4. In a more complex situation, it may even be required to achieve a virtual match between two completely different devices e.g., emulating a magnetic tape device on a disk in the host system. Therefore, I/O emulation requires not only a thorough understanding of the I/O control functions of both the target and the host machines, but also requires some knowledge of the devices themselves, such as a device's data representation, record structure, control sequence, timing, special functions and so on. I/O emulation process is not like the central processor emulation. Generally, different CPU's perform basically similar functions which are not time dependent and not influenced by any factors external to the CPU. This makes CPU emulation relatively straightforward. On the other hand, I/O emulation is much more involved and complex because the I/O system provides a link between the CPU and the outside world, resulting in time and device dependent operations.

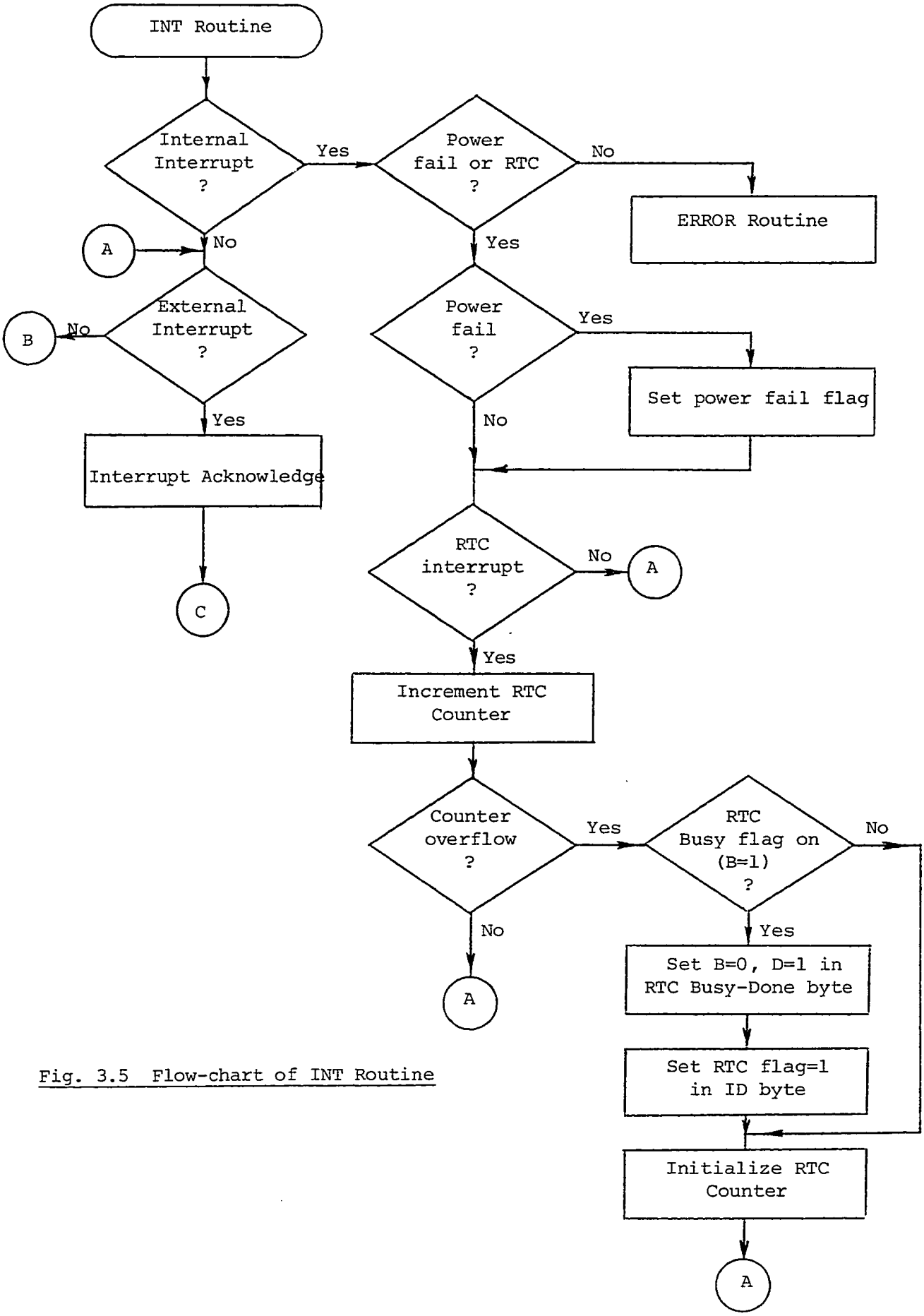


Fig. 3.5 Flow-chart of INT Routine

* Carriage Return

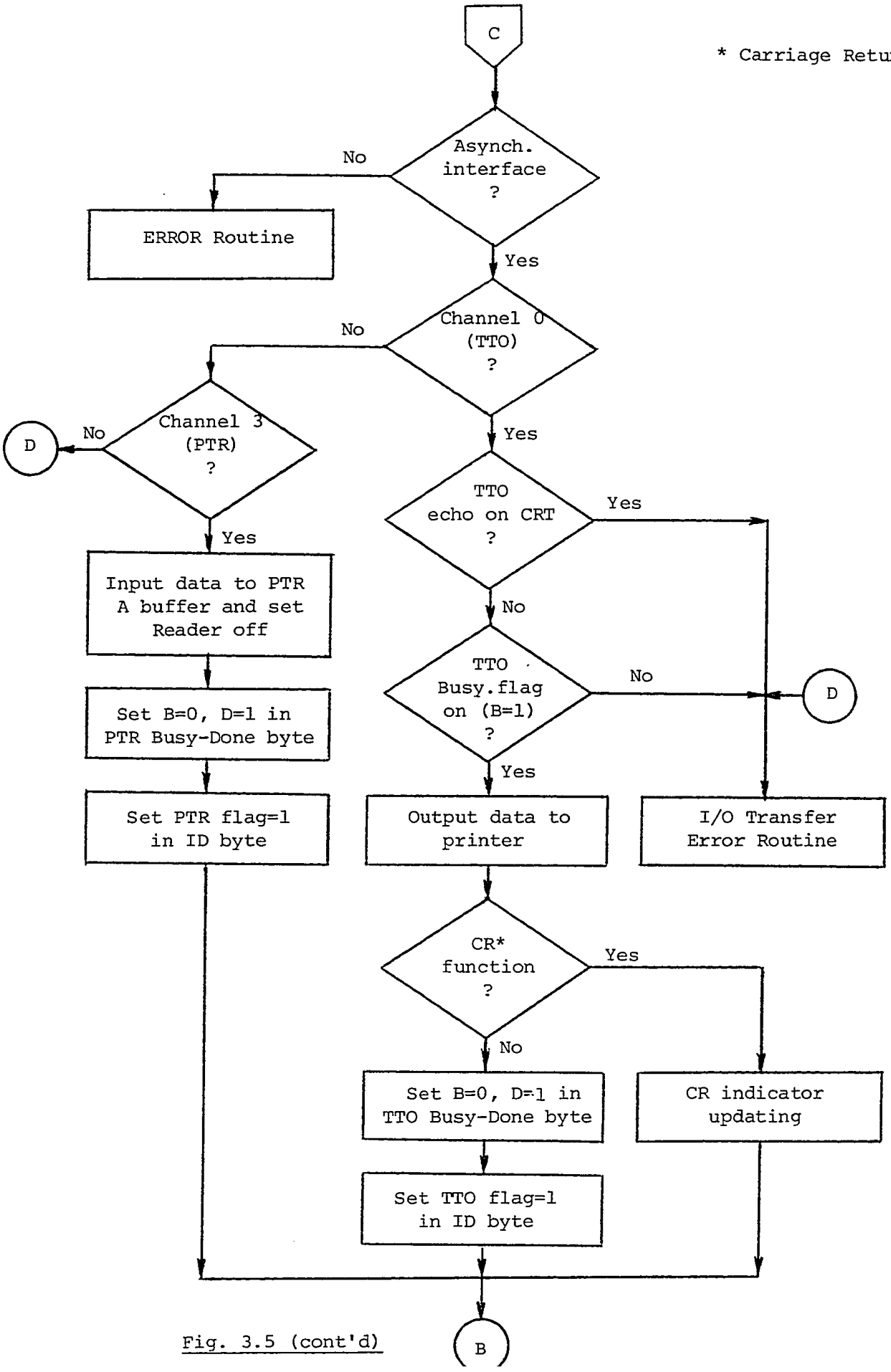


Fig. 3.5 (cont'd)

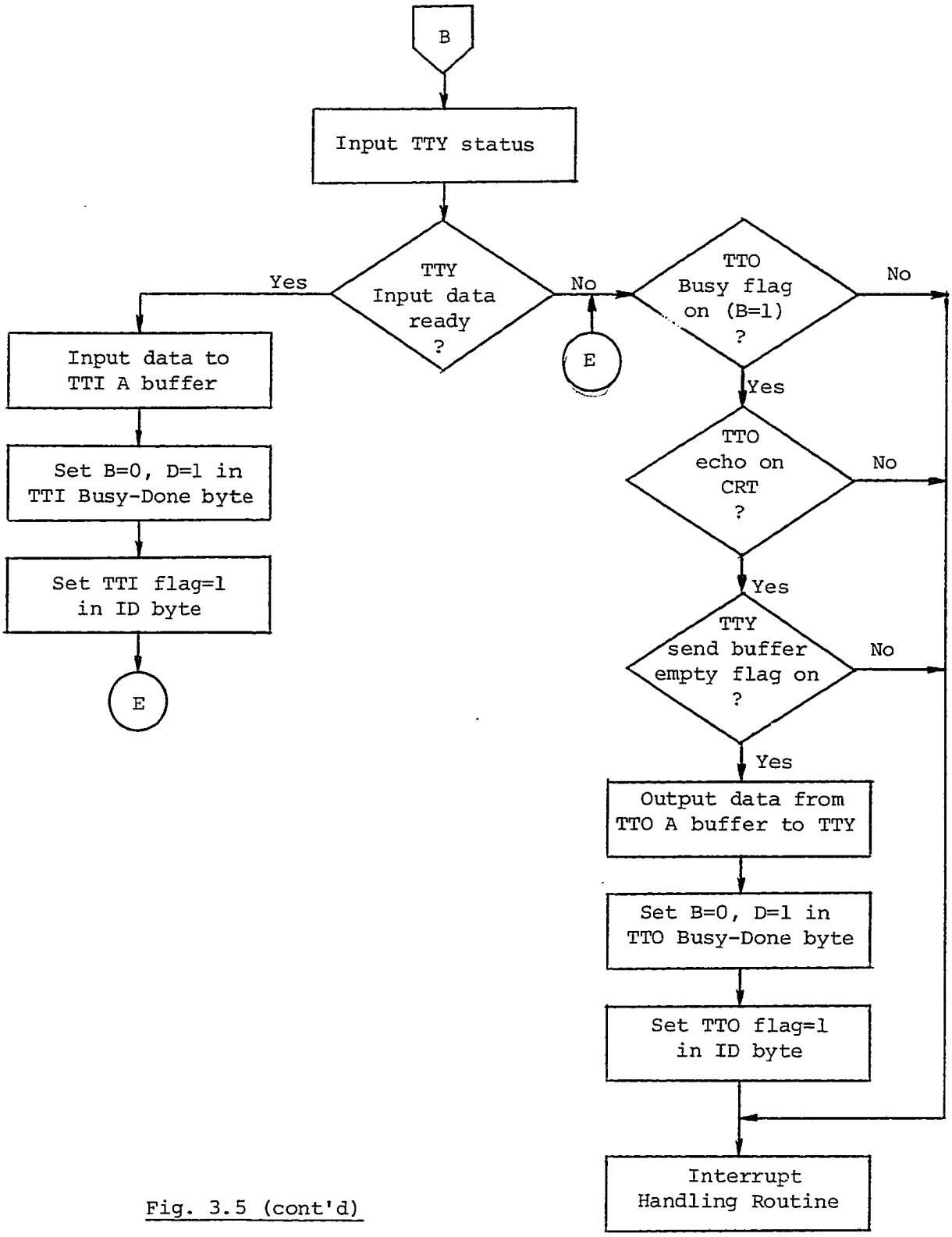

B
Input TTY status
TTY Input data ready ?
Yes
No
Input data to TTI A buffer
Set B=0, D=1 in TTI Busy-Done byte
Set TTI flag=1 in ID byte
E
E
TTO Busy flag on (B=1) ?
No
Yes
TTO echo on CRT ?
No
Yes
TTY send buffer empty flag on ?
No
Yes
Output data from TTO A buffer to TTY
Set B=0, D=1 in TTO Busy-Done byte
Set TTO flag=1 in ID byte
Interrupt Handling Routine

Fig. 3.5 (cont'd)

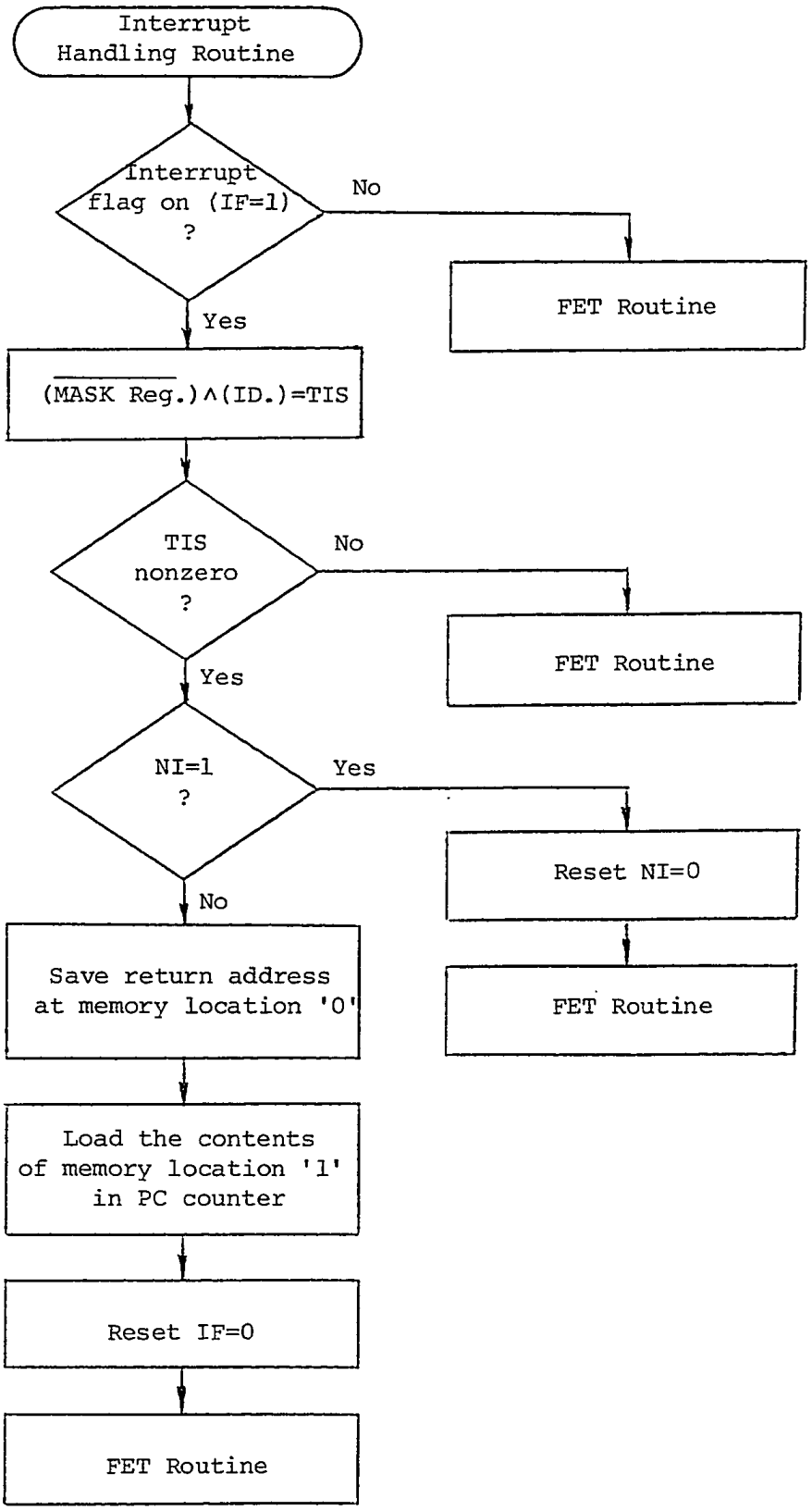

Interrupt Handling Routine
Interrupt flag on (IF=1) ?
No
FET Routine
Yes
(MASK Reg.)∧(ID.)=TIS
TIS nonzero ?
No
FET Routine
Yes
NI=1 ?
Yes
Reset NI=0
FET Routine
No
Save return address at memory location '0'
Load the contents of memory location '1' in PC counter
Reset IF=0
FET Routine

Fig. 3.5 (cont'd)

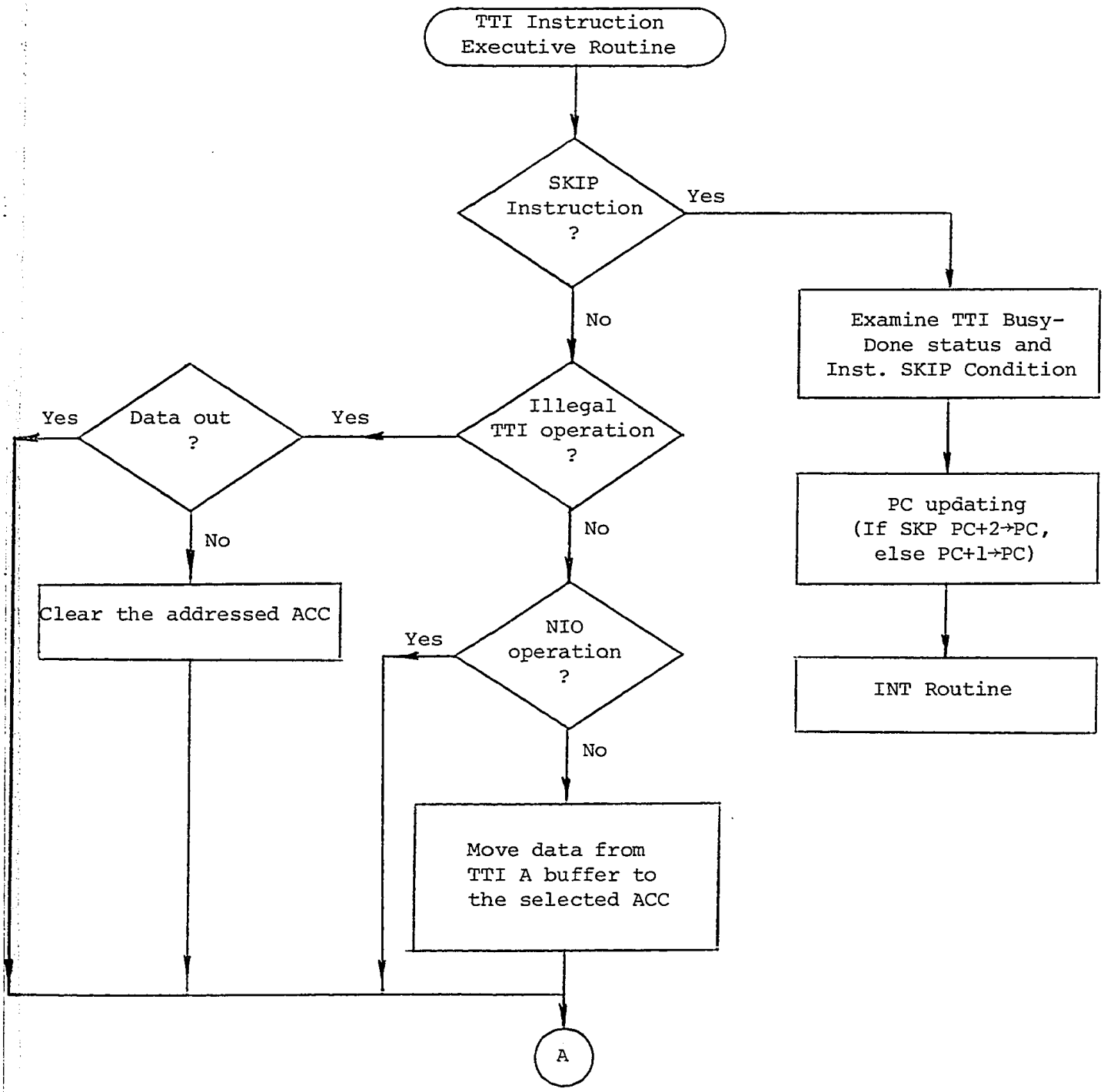


Fig. 3.6 Flow-chart of NOVA TTI Instruction Executive Routine

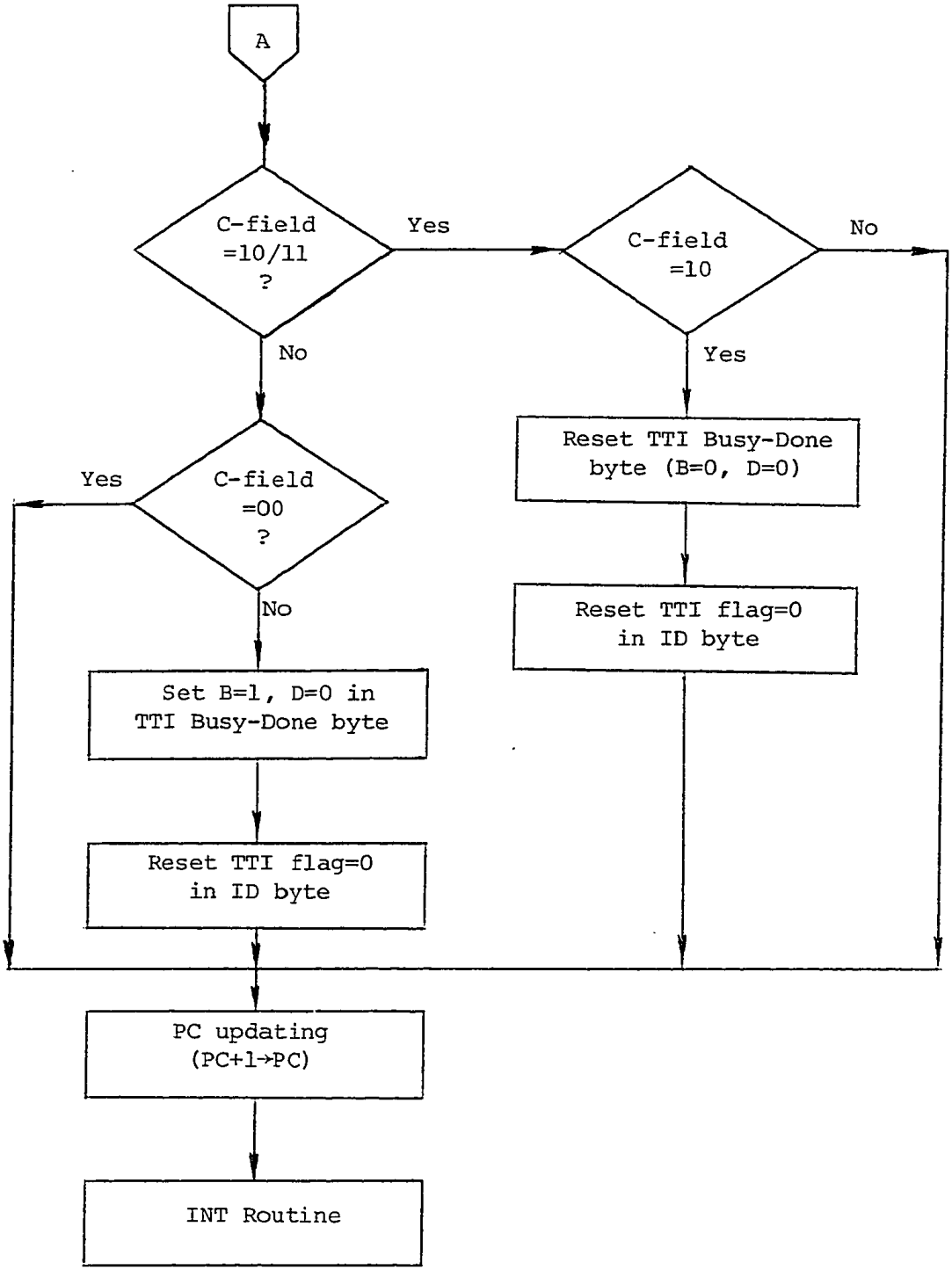

A
C-field =10/11 ?
Yes
C-field =10
No
No
Yes
Reset TTI Busy-Done byte (B=0, D=0)
Yes
C-field =00 ?
Reset TTI flag=0 in ID byte
No
Set B=1, D=0 in TTI Busy-Done byte
Reset TTI flag=0 in ID byte
PC updating (PC+1→PC)
INT Routine

Fig. 3.6 (cont'd)

The configurations of the emulated (NOVA) and the emulating (MICRODATA 1600) machines are shown in Table 3.6. At the present time the MICRODATA system of the University of Ottawa does not have a Teletype connected to it, instead a CRT display and keyboard are connected to the machine via the Teletype interface. The CRT display and keyboard are used as the operator's console in our system. A virtual match is created to simulate the Teletype functions on the operator's console — the keyboard simulates the TTI operations, and the CRT display simulates the TTO functions. Another version of the NOVA emulator provides the option of emulating the TTO functions either on the CRT display or on the Texas Instruments Silent 700 data terminal by setting/not setting sense switch 1 on the MICRODATA console. Because of a hardware restriction in the MICRODATA, only 8 lower console switches can be read into the machine. This factor makes it difficult to use the MICRODATA console switches to emulate the 16 NOVA console data switches which can be input into any accumulator (DIA -, CPU). Therefore, the "Read Console Switch" function of the NOVA has been emulated by the keyboard/display combination on the MICRODATA. A prompting character (s) is output on CRT display when the emulator executes the Read Console Switch command. In response, the operator must enter the NOVA console data switch values in octal via the keyboard.

The hardware features of the host system play a significant role in the emulation process. Sometimes added hardware and/or modifications to the host system may be needed to assist emulation. If this is not done, then it may be necessary to accept some

Table 3.6

The Configuration of the Host and Target Machines

Target Computer	Host Computer
NOVA	MICRODATA 1600
Console input	MUMAC M2480 Keyboard
TTY input	MUMAC M2480 Keyboard
TTY Output	MUMAC 2480 Display or TEXAS Instruments Silent 700 Data Terminal
Paper tape reader (PTR)	Digelab PTR01 Paper tape reader
REAL TIME CLOCK	REAL TIME CLOCK

reasonable compromises in emulating certain target machine functions. For example the MICRODATA real time clock rate of 1000 Hz makes it impossible to emulate the 60 Hz clock rate (one of NOVA's real time clock frequencies). So 50 Hz AC line frequency is assumed in the emulator in order to conform to the restriction imposed by MICRO-DATA's real time clock rate.

As mentioned earlier, the I/O transfer operations, in general, can be handled either in program mode or in interrupt mode at the conventional machine level. Both modes may be implemented separately in the emulator in the form of separate modules. However, in the implementation of the NOVA emulator, data transfer is always handled in the MICRODATA interrupt mode and the INT (internal house keeping) microroutine takes care of all data transfers. The routines for emulating a device's I/O instructions only set or reset some flags (e.g. Busy/Done flag etc.), or perform test and skip functions (e.g., SKP —— instruction). This approach is based on two main factors:

1. In the emulation of program mode operation, the data transfer routines have to examine the status flags of a device before issuing a data transfer command. Similarly, in the emulation of interrupt mode operation, checking the status flags of all the devices is necessary between two NOVA instructions. Therefore duplicated status checking operations will be performed if the two I/O modes are implemented in separate modules, resulting in unnecessary usage of control storage and redundant operations. For this reason, the INT microroutine handles both the interrupt and program mode data transfers. Interrupt driven I/O emulation is the only

way that the MICRODATA as the host machine can handle both the interrupt and program mode data transfer operations. In the emulator, a special flag — called the interrupt flag (IF) — has been used to indicate the program mode/interrupt mode status of the target machine. If the target system is in program mode (IF=0), then no interrupt functions are simulated, and the INT routine only handles the data transfer functions. On the other hand, if the target system is set in interrupt mode (IF=1), the INT routine not only handles the data transfer functions, but also simulates the interrupt strategy of the NOVA computer.

2. Because of the loop structure of the MICRODATA I/O system, once a device requests service and the request is acknowledged, no other device can get CPU attention before this device is served. In other words, request for service by any other device is completely ignored. The NOVA I/O system is more versatile, for it allows some kind of parallel control over all the I/O functions. For example, the MASK OUT (MSKO) instruction can disable a set of devices from interrupt request, but all devices can signal their Busy/Done status to the CPU simultaneously. The I/O system will not be locked up by any individual request. The CPU's response to these requests depends on the NOVA interrupt service routine. The parallelism of NOVA I/O functions has been emulated in the NOVA emulator. As stated earlier, the INT routine is designed to examine and update the status of individual device flags, and perform data transfers. As a result, when an emulated NOVA interrupt occurs, a snap shot of the target I/O status and data is already available in MICRODATA's local memory. Following this, when a

NOVA I/O data transfer instruction is executed, the data is retrieved from the MICRODATA's local memory without directly communicating with the emulated device.

In the specific case of NOVA I/O system emulation on the MICRODATA, it is not efficient or convenient to use MICRODATA software to handle emulated I/O data transfers. This is because executing the INT routine in software would involve constant swapping between the native (MICRODATA) mode and emulation (NOVA) mode between pairs of NOVA instructions, and this would involve time-consuming overhead.

CHAPTER 4

CONCLUDING REMARKS

4.1 Introduction:

In the previous chapters, the emulation of both the central processor and the I/O system has been discussed and illustrated using the NOVA emulator. In any emulation process, functions of the target machine have to be faithfully performed on the host system. However, there is a performance penalty for using a machine to execute a foreign (non-native) instruction set. The actual performance achieved normally depends on the degree to which new data paths, or function units, and other related hardware are added to the host. In addition, the performance is also affected by any new software and firmware added to the host system to facilitate emulation. Therefore, some form of hardware/software/firmware trade-off may be involved in the emulation process.

The main reasons for the trade-off among hardware, software and firmware are usually based on the following factors:

1. To achieve an otherwise unattainable performance goal.
2. To optimize overall control storage and/or main memory usage.
3. To reduce firmware and/or software complexity.
4. To optimize instruction execution time.
5. To decrease overall emulation cost.

A trade-off between these three components can significantly affect the performance of the emulator. For instance, as mentioned in Chapter 3, conversion from one coding scheme to another may be achieved by either firmware or software; however, both of them are

time-consuming and require a significant amount of control storage or main memory, respectively. If code conversion hardware is added in device's I/O interface, it will save the corresponding storage space or simplify the firmware complexity for data interpretation as well as speed up the associated instruction execution time; the only penalty is higher implementation cost.

4.2 Trade-offs in the NOVA Emulator Design:

In this thesis, the firmware-controlled emulation process has been adopted to emulate the NOVA system on the MICRODATA 1600 . Similarity in the structures of these machines has simplified the design of the emulator.

Instruction execution time and control storage usage are the main concerns in the present version of the NOVA Emulator. In order to make the emulator more readable and structurally organized, the overall design is modular (Fig. 4.1). For I/O instruction interpretation, an individual device not only has its own interpretation microroutine, but also the relevant Busy-Done flags in the local memory (Ref. Table 2.1). Obviously, code optimization can be attempted to optimize the use of control store and local memory, but it will not result in a well-organized modular structure.

4.3 Outlook and General Discussion:

The development of the experimental system (NOVA) on MICRODATA 1600 shows that with the help of microprogramming we can create

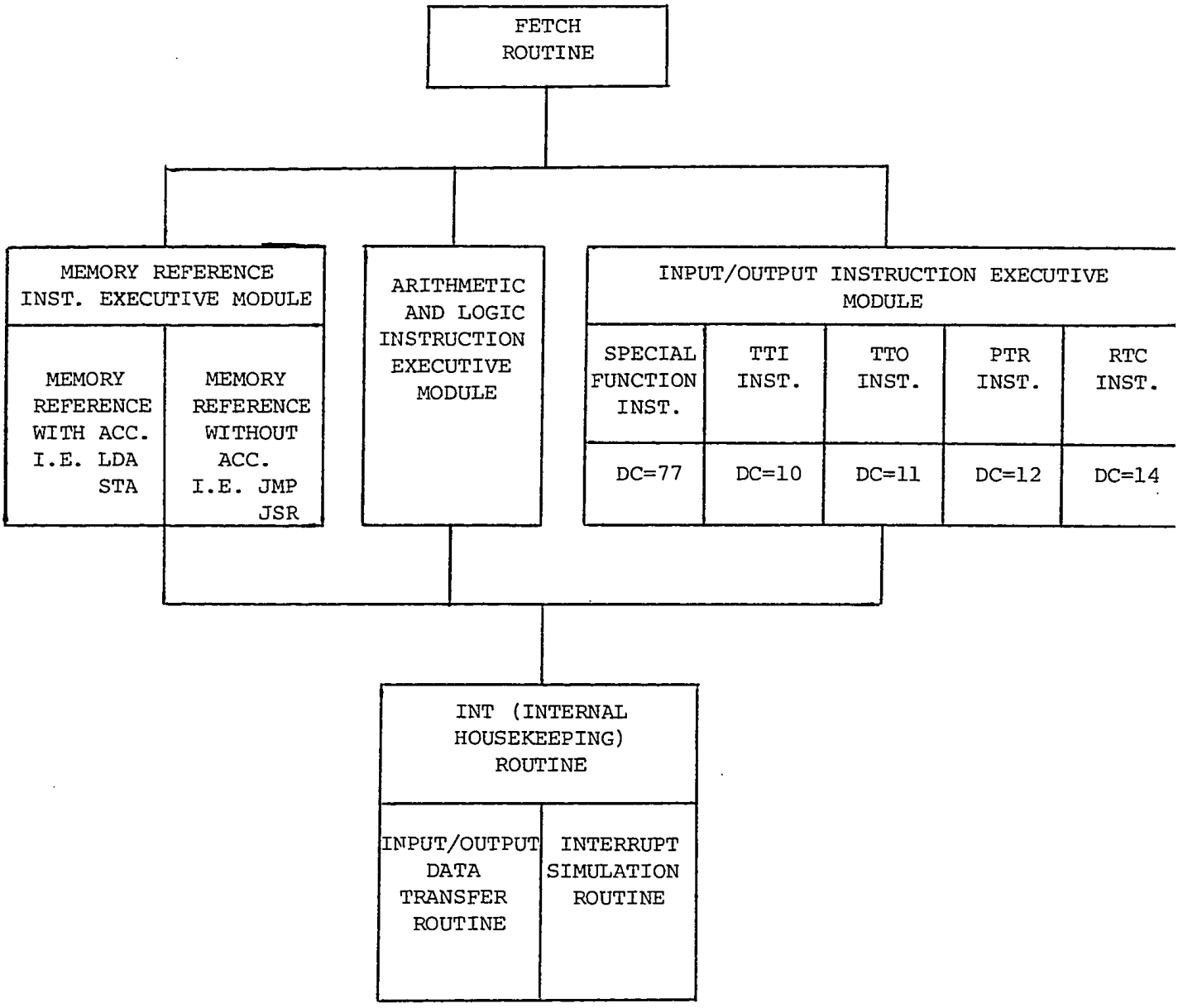


Fig. 4.1 BASIC MODULAR STRUCTURE OF NOVA EMULATOR

a virtual system based on the existing hardware resources. However, a given virtual machine may not be efficiently emulated by a host; in a majority of commercially available microprogrammable computers, the primary objective is to provide efficient emulation for the native-mode machine. The hardware resources and microinstruction repertoire of such machines are often specialized for this emulation and thus may be difficult to adapt for emulating other virtual systems. For example, a machine whose registers and operations were designed to handle 32-bit integers in two's complement notation would probably not be a good host for emulating a machine whose basic data type is 36-bit integers in sign-magnitude representation.

Even though microprogramming provides flexibility in defining a virtual machine, its utility as a general purpose emulation vehicle entirely depends on the organization of the hardware resources of the host machine. Ideally, a universal host would be equally efficient in the emulation of all target machines. As stated in previous chapters, the structure of the target machine's central processor unit and I/O system, as well as the format of its instructions play a significant role in the emulation process. Therefore, the design of a truly universal host machine must have some features to accommodate a range of target machine architectural parameters (e.g., data path width, processing functions, interrupt facilities, etc.).

Data width and functional processing facilities of the host have a significant impact on the efficiency of emulation of a given target machine. If all memory in the host is addressable to the bit level, in other words data addresses in main memory need not be

aligned on a byte, half word or word address, and there is no fixed word size of data items (i.e. data may have a length varying from 1 bit to any number of bits), then the instruction length or data formats of the target machine can be handled efficiently.

In addition, the capability of the host machine to handle diverse type of operations is an important aspects of its suitability as a universal host. For example, its ability to extract a field of arbitrary length from a register or memory location is a factor contributing to efficient decoding and analysis of target instructions.

The number of general and special purpose registers, availability and size of stacks etc., in the host machine affect the efficiency of emulation. If the host machine has fewer special and general purpose registers compared to the target machine, the programmer is forced to create virtual registers in the host using slower memory facilities and this affects the speed of target instruction execution.

The size and structure of the control store also affect the suitability of the host machine to act as a universal host. There are two basic types of control store: the first type consists of a control store which is separated from the main memory both in terms of physical construction and addressing logic, and the control store usually operates at a higher speed than does the main memory. In the second type, the control store is implemented in the same address space as the main memory. In this case, both memories have the same cycle time, and are accessed via the same addressing logic. In the latter structure, the size of control store is usually not fixed and the programmer is not worried too much about the control

store capacity. However, this type of organization requires a memory which is fast enough to be used as control store and yet cheap enough that a relatively large amount can be provided at a reasonable cost for use as the main memory. Also memory protection has to be provided to prevent the microprogram portions in the memory being destroyed. In the structure of first type, the size of the control store is fixed, so it may limit the emulation capability on the host. In concept, a virtual control store, similar to conventional virtual memory, would provide the need of control store for emulation. Microprograms may be stored in the control store as well as in the main memory; the most frequently executed microprograms can reside in the control store, whereas less frequently used microprograms can reside in main memory. The swapping of microprograms can be done whenever required. Although, some time is lost whenever swapping is done, this concept enlarges the (virtual) size of the control store considerably.

The I/O capability of the host processor usually is the limiting factor in the overall success of the host in supporting emulation. The relatively sophisticated I/O structures of most modern computer systems make it difficult for a single microprogrammable host processor to efficiently emulate both the target CPU instruction set as well as the target I/O system. The use of a separate processor to emulate all I/O functions such as data transfer between I/O devices and CPU, data conversion, record reformatting, updating the device status indicators, etc., would improve the efficiency of emulation. However, this would increase the cost of emulation, compared to firmware-controlled emulation, because of the additional

hardware and associated software/firmware for the I/O processor.

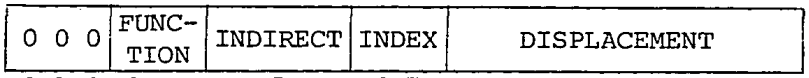
In conclusion, we recommend that further work be undertaken in the following areas:

1. Automated generation of firmware for emulation of CPU instructions.
2. A more detailed and quantitative study of I/O emulation, specially for the case when an additional processor is used for supporting I/O emulation.
3. Implementation of virtual control store and its effect on emulation.

APPENDIX A

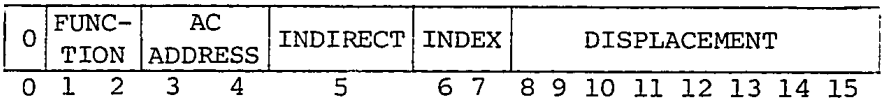
HARDWARE INSTRUCTIONS

MEMORY REFERENCE WITHOUT ACCUMULATOR



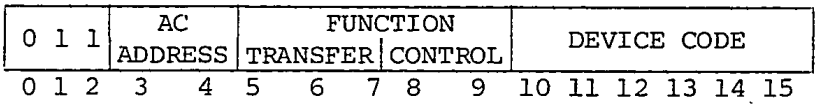
00 JMP
01 JSR
10 ISZ
11 DSZ

MEMORY REFERENCE WITH ACCUMULATOR



01 LDA
10 STA

INPUT-OUTPUT



000 NIO 00 **
001 DIA 01 S
010 DOA 10 C
011 DIB 11 P
100 DOB
101 DIC
110 DOC
111 SKP 00 BN
01 BZ
10 DN
11 DZ

APPENDIX A
(cont'd)

ARITHMETIC AND LOGIC

1	AC SOURCE ADDRESS	AC DEST. ADDRESS	FUNCTION	SHIFT	CARRY	NO LT.	SKIP
0	1 2	3 4	5 6 7	8 9	10 11	12	13 14 15

000 COM				001 SKP
001 NEG				010 SZC
010 MOV				011 SNC
011 INC				100 SZR
100 ADC				101 SNR
101 SUB	01 L	01 Z		110 SEZ
110 ADD	10 R	10 0		111 SBN
111 AND	11 S	11 C		

APPENDIX B

ALPHABETICAL LIST OF MICRODATA 1600 MICROINSTRUCTIONS

MNEMONIC	OPERAND CODE	OBJECT BASE	CLASS	COMMAND
ADD*	f,LITC(r)	8000	1	ADD FILE
AF	f,n	3000	2	ADD FILE WITH LITERAL
AND*	f,LFTC(r)	E000	1	AND WITH FILE
BSL	n	1B08	3	BANK SELECT
CAK*	f,(r)	70C0	1	CONCURRENT ACKNOWLEDGE
CIO*	f,(r)	7080	1	CLEAR I/O MODE
COX*	f,(r)	7090	1	CONTROL OUTPUT
CP	f,n	6000	2	COMPARE FILE
CPY*	f,LITC(r)	B000	1	COPY
CSP		1B04	4	CLEAR STACK POINTER
DCR		1002	4	DISABLE COMM. RATES
DEC*	f,c(r)	9040	1	DECREMENT FILE
DEI		1704	4	DISABLE EXTERNAL INTERRUPTS
DIX*	f,(r)	70E0	1	DATA INPUT
DOX*	f,(r)	70A0	1	DATA OUTPUT
DRT		1710	4	DISABLE REAL TIME CLOCK
DSP		1B02	4	DECREMENT STACK POINTER
ECR		1001	4	ENABLE COMM. RATES
ECS		7070	4	ENTER CONSOLE SWITCHES
EEI		1708	4	ENABLE EXTERNAL INTERRUPTS
EIS*	f,(r)	7040	1	ENTER INTERNAL STATUS
ELT	f,n	0000	2	EXECUTE, LITERAL TYPE
EOT*	f,LCTIFD(r)	0000	1	EXECUTE, OPERATE TYPE
ERT		1720	4	ENABLE REAL TIME CLOCK
ESS*	f,(r)	7010	1	ENTER SENSE SWITCHES
HLT		1780	4	HALT
IAK*	f,(r)	70D0	1	INTERRUPT ACKNOWLEDGE
ICR		1004	4	INPUT COMM. RATES
ILS		1B00	4	INHIBIT L SAVE
INC*	f,c(r)	8040	1	INCREMENT FILE
ISP		1B01	4	INCREMENT STACK POINTER
JE	n	0000	5	JUMP EXTENDED
JP	n	1400	3	JUMP IN 1 K
LE	n	1800	3	LOAD EIGHT CONTROL
LF	f,n	2000	2	LOAD FILE WITH LITERAL
LM	n	1200	3	LOAD M
LN	n	1300	3	LOAD N
LOR*	f,LFTC(r)	C000	1	LOGICAL OR WITH FILE
LS	n	1700	3	LOAD SEVEN CONTROL
LT	n	1100	3	LOAD T
LU	n	1600	3	LOAD U
LZ	n	1000	3	LOAD ZERO CONTROL
MLC		1A00	4	MODIFY LOWER COMMAND
MOV*	f,LC(r)	C000	1	MOVE FILE

APPENDIX B (cont'd)

MNEMONIC	OPERAND FIELD	OBJECT BASE	CLASS	COMMAND
NOP		1000	4	NO OPERATION
POF*	f,c(r)	B040	1	+1 TO FILE/REG.
RLT	n	1900	3	RETURN LOAD T
RMF*	f,LID(r)	A000	1	READ MEMORY, FULL CYCLE
RMH*	f,LID(r)	A020	1	READ MEMORY, HALF CYCLE
RSP		1060	4	RETURN, SELECT PRIMARY FILE
RSS		10A0	4	RETURN, SELECT SECONDARY FILE
RTN		1020	4	RETURN
SBO*	f,LTC(r)	9040	1	SUBTRACT FILE, ONES COMPLEMENT
SBT*	f,LTC(r)	9000	1	SUBTRACT FILE, TWOS COMPLEMENT
SFL*	f,LC(r)	F000	1	SHIFT FILE LEFT
SFR*	f,LC(r)	F020	1	SHIFT FILE RIGHT
SIX*	f,(r)	70F0	1	STACK/SPARE INPUT
SLI*	f,c(r)	F040	1	SHIFT FILE LEFT AND INSERT
SOX*	f,(r)	70B0	1	SPARE OUTPUT
SPF		1040	4	SELECT PRIMARY FILE
SRF*	f,(r)	7020	1	SHIFT FILE RIGHT 4
SRI*	f,c(r)	F060	1	SHIFT FILE RIGHT AND INSERT
SSF		1080	4	SELECT SECONDARY FILE
SSL		1BC0	4	SELECT STACK LOWER
SSP		1B90	4	SELECT STACK POINTER
SSU		1BA0	4	SELECT STACK UPPER
TN	f,n	5000	2	TEST NOT ZERO
TZ	f,n	4000	2	TEST IF ZERO
WMF*	f,LID(r)	A010	1	WRITE MEMORY, FULL CYCLE
WMH*	f,LID(r)	A030	1	WRITE MEMORY, HALF CYCLE
XOR*	f,LFTC(r)	D000	1	EXCLUSIVE OR WITH FILE
ZOF*	f,c(r)	B000	1	ZERO FILE/REG.

* = FILE UPDATE INHIBIT
f = FILE EXPRESSION
(r) = DESTINATION REGISTER DESIGNATOR
n = LITERAL EXPRESSION
L,C,T,R,I,D = C. FIELD DESIGNATOR

```
                    *
                    *
                    *                          NOVA EMULATOR
                    *                        ON MICRODATA 1600
                    *
                    *
                    *
0000                F0      EQU   0          MICRODATA CONDITION FLAGS.
0001                F1      EQU   1          (BIT 1) CARRY BIT.
0002                IU      EQU   2          UPPER BYTE OF INST. REG..
0003                IL      EQU   3          LOWER BYTE OF INST. REG..
0004                LL      EQU   4          LOWER BYTE OF PROGRAM COUNTER.
0005                LU      EQU   5          UPPER BYTE OF PROGRAM COUNTER.
0006                A0L     EQU   6          LOWER BYTE OF AC0.
0007                A0U     EQU   7          UPPER BYTE OF AC0.
0008                A1L     EQU   8          LOWER BYTE OF AC1.
0009                A1U     EQU   9          UPPER BYTE OF AC1.
000A                A2L     EQU   10         LOWER BYTE OF AC2.
000B                A2U     EQU   11         UPPER BYTE OF AC2.
000C                A3L     EQU   12         LOWER BYTE OF AC3.
000D                A3U     EQU   13         UPPER BYTE OF AC3.
000E                OL      EQU   14         LOWER BYTE OF OPERAND REG..
000F                OU      EQU   15         UPPER BYTE OF OPERAND REG..
                    *
                    *   INITIALIZATION:-
                    *
0000 113C                   LT    X'3C'      * SILENT 700 DATA TERMINAL BAUD RATE CONTROL.
0001 0189                   JE    COX
0002 1103                   LT    X'03'      * SET BAUD RATE TO 300 BPS.
0003 0192                   JE    DOX
0004 115C                   LT    X'5C'      * SILENT 700 DATA TERMINAL CHARACTER FORMAT CONTROL.
0005 0189                   JE    COX
0006 1112                   LT    X'12'      * CHARACTER FORMAT SETTING: EVEN PARITY,
0007 0192                   JE    DOX        * 1 STOP BIT, 8 BITS CHARACTER LENGTH.
0008 113C                   LT    X'3C'      * PAPER TAPE READER BAUD RATE CONTROL.
0009 0189                   JE    COX
000A 1164                   LT    X'64'      * SET BAUD RATE TO 1200 BPS.
000B 0192                   JE    DOX
000C 115C                   LT    X'5C'      * PAPER TAPE READER CHARACTER FORMAT CONTROL.
000D 0189                   JE    COX
000E 1172                   LT    X'72'      * CHARACTER FORMAT SETTING : EVEN PARITY, 1 STOP BIT,
000F 0192                   JE    DOX        *                            8 BITS CHARACTER LENGTH.
0010 117C                   LT    X'7C'      * SET PAPER TAPE READER DATA READY MASK.
0011 0189                   JE    COX
0012 1162                   LT    X'62'
0013 0192                   JE    DOX
0014 11FC                   LT    X'FC'      * SET DATA TERMINAL READY CONTROL.
0015 0189                   JE    COX
0016 1101                   LT    X'01'
0017 0192                   JE    DOX
0018 119C                   LT    X'9C'      * ARM THE ASYNCHRONOUS MODEM INTERFACE INTERRUPT.
0019 0189                   JE    COX
001A 018E                   JE    DIX
001B 1708                   EEI              * PROCESSOR INTERRUPT ENABLE.
001C 1080                   SSF              * EMULATED  SYSTEM (NOVA) INITIALIZATION:
001D 21B2                   LF    1,X'B2'    * CLEAR ALL DEVICES' CONTROL FLAGS,  AND INTERRUPT
001E 026F                   JE    CLEAR      * DISABLE FLAGS.
```

```
001F 2108                LF     F1,X'08'    * LOAD THE STARTING ADDRESS INTO PC COUNTER THROUGH
0020 2F04                LF     OU,X'04'    * KEY BOARD INPUT.
0021 1D5C                JP     ECP
                  *
                  *  FETCH NOVA INST. FROM MEMORY AND STORE IN INST. REG..
                  *  THIS REQIRES 2 MEMORY READS.
                  *
0022 C401         FET    MOV    LL,(T)      * THE CONTENTS OF PROGRAM COUNTER ARE MOVED TO
0023 BE20                CPY    OL,T        * OPERAND REG..
0024 C501                MOV    LU,(T)
0025 BF20                CPY    OU,T
0026 002F                JE     MAP         * CALL SUBROUTINE MAP.
0027 A000                RMF    F0          * LOAD THE LOWER BYTE OF NOVA INST. IN IL REG..
0028 B320                CPY    IL,T
0029 CE03                MOV    OL,(N)      * LOAD THE UPPER BYTE OF NOVA INST. IN IU REG..
002A AF02                RMF    OU,(M)
002B B220                CPY    IU,T
                  *
                  *  INST. CLASS SEPARATION : INST. SET WITH MEMORY REFERENCE(I.E.  JMP,
                  *  STA, LDA ETC.), ONE WITHOUT (I.E. INPUT/OUTPUT INST., ARITH. AND LOGIC
                  *  INST.. THE OP. CODE OF THOSE INST. IS GREAT THAN 3.).
                  *  AND JUMP TO THE DESIGNATED SUBROUTINE.
                  *
002C 62A0                CP     IU,X'A0'    * SKIP, IF OP. CODE <3,
002D 1434                JP     MRE         * ELSE GO TO MRE ROUTINE.
002E 14A8                JP     IOA         * CONTINUE AT IOA ROUTINE.
                  *
                  *  MEMORY MAPPING: NOVA'S MEMORY IMAGE TO BE MAPPED TO MICRODATA'S MEMORY.
                  *
002F FE0C         MAP    SFL    CL          * OPERAND REG. CONTAINS THE NOVA INST. ADDRESS,
0030 FF80                SFL    OU,L        * ACCORDING TO THE MEMORY MAPPING EQUATION:
0031 8E48                INC*   OL,(N)      * THE CORRESPONDING MICRODATA MEMORY LOCATION =
0032 8F8A                ADD*   OU,L,(M)    * ( OPERAND REG. )*2.
0033 1020                RTN                * LOAD THE LOWER BYTE NOVA INST. ADDRESS IN M,N REG..
                  *
                  *  MEMORY REFERENCE INST. DECODING AND OPERAND ADDRESS CALCULATION.
                  *  (PAGE ZERO, RELATIVE, OR BASE REG. ADDRESSING.)
                  *
0034 2F00         MRE    LF     OU,X'00'    * CLEAR OU REG..
0035 C301                MOV    IL,(T)
0036 BE20                CPY    OL,T        * LOAD THE LOWER BYTE NOVA INST. INTO CL REG..
0037 6203                TN     IU,X'03'    * SKIP, IF RELATIVE OR BASE REG. ADDRESSING.
0038 144A                JP     DID         * ELSE GO TO DID ROUTINE(PAGE ZERO ADDRESSING).
                  *
                  *  ADDRESS CALCULATION FOR RELATIVE AND BASE REG. ADDRESSING.
                  *
0039 4380                TZ     IL,X'80'    * CHECK DISPLACEMENT SIGN BIT. POSITIVE INTERGER?
003A CF60                LOR    OU,T,F      * NO, EXTEND DISPLACEMENT SIGN BIT TO OU REG..
003B 1103                LT     X'03'       * DISTINGUISH THE TYPE OF ADDRESSING (BASE OR RELATIVE)
003C E229                AND*   IU,T,(T)    * AND LOAD  THE INDEX IN IL REG..
003D B320                CPY    IL,T
003E F300                SFL    IL          * INDEX REG. MAPPING.
003F 33C2                AF     IL,X'C2'
0040 4202                TZ     IU,X'02'    * RELATIVE ADDRESSING ?
0041 3304                AF     IL,X'04'    * NO, BASE ADDRESSING.
0042 C306                MOV    IL,(U)      * COPY  THE CONTENTS OF THE SPECIFIED INDEX REG. TO
0043 0C01                ELT*   F0,X'01'    * OPERAND REG..
```

```
0044 cE20                ADD    OL,T
0045 0101                ELT*   F1,X'01'
0046 8FA0                ADD    OU,L,T
0047 1600                LU     X'00'
0048 117F                LT     X'7F'
0049 EF20                AND    OU,T
                    *
                    *  EFFECTIVE ADDRESS CALCULATION. (DIRECT, INDIRECT, OR AUTO INCREMENT/DECREMENT
                    *  ADDRESSING )
                    *  ADDRESS CALCULATION FOR PAGE ZERO OR AUTO INCREMENT/DECREMENT ADDRESSING.
                    *
004A 5204           DID      TN     IU,X'04'    * SKIP, IF INDIRECT ADDRESSING,
004B 1471                    JP     SEP         * ELSE GO TO SEP ROUTINE(DIRECT ADDRESSING).
004C 4F7F           CKR      TZ     OU,X'7F'    * YES, CHECK FOR AUTO INCREMENT/DECREMENT ADDRESSING.
004D 146A                    JP     IDT         * (WHETHER THE MEMORY LOCATION IS IN THE RANGE OF 20-37)
004E 4EE0                    TZ     OL,X'E0'
004F 146A                    JP     IDT
0050 5E10                    TN     OL,X'10'
0051 146A                    JP     IDT
0052 2380                    LF     IL,X'80'    * MANIPULATION OF AUTO INCREMENT/DECREMENT ADDRESSING.
0053 4E08                    TZ     OL,X'08'    * SET INCREMENT/DECREMENT INDEX IN IL REG.
0054 2390                    LF     IL,X'90'
0055 002F                    JE     MAP         * MEMORY MAPPING.
0056 0C5C                    JE     MID         * ADDRESS CALCULATION.
0057 4F60           OVK      TZ     OU,X'60'    * TRANSFER TO OVF ROUTINE
0058 15E0                    JP     OVF         * IF MEMORY OVERFLOW OCCURS.
0059 5F80           REP      TN     OU,X'80'    * SKIP, IF INDIRECT ADDRESSING,
005A 1471                    JP     SEP         * ELSE GO TO SEP ROUTINE.
005B 144C                    JP     CKR         * CONTINUE AT CKR ROUTINE.
                    *
                    *  MEMORY CONTENTS INCREMENT OR DECREMENT SUBROUTINE.
                    *
005C C306           MID      MOV    IL,(U)      * IL REG. CONTAINS THE INCR./DECR. INDEX CODE .
005D A020                    RMH    F0          * RETRIEVE THE CONTENTS OF THE SPECIFIED MEMORY
005E B320                    CPY    IL,T        * LOCATION (20-37) AND INCR. OR DECR. THE WORD
005F 0351                    ELT    IL,X'51'    * RETRIEVED, THEN WRITE THE ALTERED WORD BACK INTO THE
0060 A030                    WMH    F0          * SAME LOCATION.
0061 CE03                    MOV    OL,(N)
0062 BE20                    CPY    OL,T
0063 AF22                    RMH    OU,(M)
0064 B320                    CPY    IL,T
0065 0391                    ELT    IL,X'91'
0066 A030                    WMH    F0
0067 BF20                    CPY    OU,T
0068 1600                    LU     X'00'
0069 1020                    RTN
                    *
                    *  ADDRESS CALCULATION FOR DIRECT OR INDIRECT ADDRESSING.
                    *
006A 002F           IDT      JE     MAP         * MEMORY MAPPING AND RETRIEVE THE LOWER AND UPPER
006B A000                    RMF    F0          * BYTE OF NOVA ADDRESS WORD FROM MEMORY AND LOAD
006C CE03                    MOV    OL,(N)      * THEM IN OPERAND REG..
006D BE20                    CPY    OL,T
006E AF02                    RMF    OU,(M)
006F BF20                    CPY    OU,T
0070 1457                    JP     OVK         * TEST MEMORY OVERFLOW.
                    *
```

```
                    *  MEMORY REF. INST. DECODING. (LDA, STA, JMP, JSR, ISZ, DSZ.)
                    *
0071 4260           SEP     TZ      IU,X'60'    * SKIP, IF MEMORY REF. WITHOUT ACC,
0072 1474                   JP      LST         * ELSE GO TO LST ROUTINE.
0073 1491                   JP      JID         * CONTINUE AT JID ROUTINE.
                    *
                    *  MEMORY REF. WITH ACCUMULATOR INST. DECODING.
                    *
0074 1118           LST     LT      X'18'       * EXTRACT  THE ACC. ADDRESS FROM NOVA INST. AND
0075 E229                   AND*    IU,T,(T)    * LOAD IT IN IL REG..
0076 B320                   CPY     IL,T        * REG. MAPPING
0077 F320                   SFR     IL
0078 F320                   SFR     IL
0079 002F                   JE      MAP         * MEMORY MAPPING.
007A 4240                   TZ      IU,X'40'    * SKIP, IF LDA INST.,
007B 1484                   JP      STA         * ELSE GO TO STA ROUTINE.
                    *
                    *  LDA INSTRUCTION EXECUTIVE ROUTINE.
                    *
007C A000                   RMF     F0          * LDA INST. EXECUTION.
007D 33B6                   AF      IL,X'B6'    * LOAD DATA FROM MEMORY TO THE SPECIFIED ACC..
007E C306                   MOV     IL,(U)
007F 0020                   ELT     F0,X'20'
0080 CE03                   MOV     OL,(N)
0081 AF02                   RMF     OU,(M)
0082 0120                   ELT     F1,X'20'
0083 148E                   JP      LUP
                    *
                    *  STA INSTRUCTION EXECUTIVE ROUTINE.
                    *
0084 33C6           STA     AF      IL,X'C6'    * STORE DATA FROM THE SPECIFIED ACC. INTO MEMORY.
0085 C306                   MOV     IL,(U)
0086 0001                   ELT     F0,X'01'
0087 A010                   WMF     F0
0088 CE03                   MOV     OL,(N)
0089 0101                   ELT     F1,X'01'
008A AF12                   WMF     OU,(M)
008B 148E                   JP      LUP
                    *
                    *  PC COUNTER UPDATING.
                    *
008C 8440           SKP     INC     LL          * INCREMENT PC COUNTER BY 2
008D 8580                   ADD     LU,L        * IF SKIP OPERATION,
008E 8440           LUP     INC     LL          * ELSE INCREMENT BY 1.
008F 8580                   ADD     LU,L
0090 15A4                   JP      INT
                    *
                    *  MEMORY REF. WITHOUT ACCUMULATOR INST. DECODING.
                    *  JMP AND JSR INST. DECODING AND EXECUTION.
                    *
0091 4210           JID     TZ      IU,X'10'    * MODIFY MEMORY INST. ?
0092 14A0                   JP      IDS         * YES, JUMP TO IDS ROUTINE.
0093 5208                   TN      IU,X'08'    * JSR INST. ?
0094 1498                   JP      JMP         * NO, JUMP TO JUMP ROUTINE.
0095 8449                   INC*    LL,(T)      * JSR INST. EXECUTION.
0096 BC20                   CPY     A3L,T       * PLACE THE  RETURN ADDRESS INTO ACC 3.
0097 8589                   ADD*    LU,L,(T)
```

```
0098 BD20                CPY    A3U,T
0099 117F                LT     X'7F'          * ENSURE 15-BIT EFFECTIVE ADDRESS.
009A ED20                AND    A3U,T
009B CE01          JMP   MOV    OL,(T)         * MOVE THE OPERAND ADDRESS TO PC COUNTER.
009C B420                CPY    LL,T
009D CF01                MOV    OU,(T)
009E B520                CPY    LU,T
009F 15A4                JP     INT
00A0 2380          IDS   LF     IL,X'80'       * MODIFY MEMORY INST. DECODING AND EXECUTION.
00A1 4208                TZ     IU,X'08'       * SET ISZ/DSZ INDEX  CODE IN IL REG..
00A2 2390                LF     IL,X'90'
00A3 002F                JE     MAP            * MEMORY MAPPING.
00A4 005C                JE     MID            * MEMORY CONTENTS INCREMENT/DECREMENT.
00A5 5C04                TN     F0,X'04'       * RESULT=0 ?
00A6 148E                JP     LUP            * NO, GO TO LUP.
00A7 148C                JP     SKP            * YES, GO TO SKP.
                   *
                   *  NON MEMORY REF. INST. DECODING.
                   *  ARITHMETIC & LOGICAL INST. DECODING.
                   *
00A8 5280          ICA   TN     IU,X'80'       * ARITH. AND LOGIC INST. ?
00A9 155A                JP     IOI            * NO, JUMP TO I/O ROUTINE.
00AA F120                SFR    F1             * LOAD CARRY BIT INTO A TEMPORARY LOCATION
00AB 5101                TN     F1,X'01'       * (BIT 0 OF FILE REG. 1).
00AC 14E0                JP     CRS
00AD F140                SLI    F1
00AE 5330          CCR   TN     IL,X'30'       * C BIT INITIATE.
00AF 14BC                JP     ACD            * USE CURRENT C BIT STATE.
00B0 2E01                LF     OL,X'01'       * SET UP MODIFIER IN OL AND OU REG. ACCORDING TO
00B1 2FC1                LF     OU,X'C1'       * C BIT = Z EXECUTE AND MICRO INST.
00B2 5310                TN     IL,X'10'       *         0 EXECUTE LOR MICRO INST.
00B3 14B9                JP     SEC            *         C EXECUTE XOR MICRO INST.
00B4 3F10                AF     OU,X'10'
00B5 4320                TZ     IL,X'20'
00B6 14B9                JP     SEC
00B7 2EFE                LF     OL,X'FE'
00B8 3F10                AF     OU,X'10'
00B9 CF06          SEC   MOV    OU,(U)         * UPDATE TEMPORARY CARRY BIT.
00BA CE01                MOV    OL,(T)
00BB 0020                ELT    F0,X'20'
00BC 2E60          ACD   LF     OL,X'60'       * EXTRACT  ACC SOURCE ADDRESS AND PLACE IN OL REG..
00BD C201                MOV    IU,(T)
00BE BE20                AND    OL,T
00BF 7E20                SRF    OL
00C0 3E10                AF     OL,X'10'
00C1 2F18                LF     OU,X'18'       * EXTRACT ACC DESTINATION ADDRESS AND PLACE IN OU REG..
00C2 EF20                AND    OU,T
00C3 FF20                SFR    OU
00C4 FF20                SFR    OU
                   *
                   * FUNCTION DECODING.
                   *
00C5 3EC6                AF     OL,X'C6'       * ACC ADDRESS MAPPING.
00C6 CE06                MOV    OL,(U)         * MOVE THE CONTENTS OF THE SPECIFIED ACC TO T REG..
00C7 0001                ELT    F0,X'01'
00C8 4204                TZ     IU,X'04'       * SKIP, IF FUNCTION CODE RANGE FROM 0 TO 3,
00C9 14E5                JP     B20            * ELSE GO TO B20 (FUNCTION CODE = 4 TO 7).
```

```
00CA 3F06                AF     OU,X'06'    * DEST. ACC ADDRESS MAPPING.
00CB 4202                TZ     IU,X'02'    * SKIP, IF COM /NEG INST.,
00CC 14E2                JP     B10         * ELSE GO TO B10 (MOV/INC INST.).
00CD 3E10                AF     OL,X'10'    * ONE'S COMPLEMENT OF THE CONTENTS
00CE CE06                MOV    OL,(U)      * OF SOURCE ACC.
00CF 0079                ELT    F0,X'79'
00D0 BE20                CPY    OL,T
00D1 01F9                ELT    F1,X'F9'
00D2 F230          SAM   SFR    IU,C        * TEST FOR EVEN OR ODD NO. FUNCTION CODE,
00D3 B220                CPY    IU,T        * THEN LOAD THE UPPER BYTE OF RESULT
00D4 1600                LU     X'00'       * INTO IU REG.,
00D5 5001                TN     F0,X'01'    * SKIP IF ODD NO. FUNCTION CODE(01/11).
00D6 1506                JP     CHS         * ELSE GO TO CHS ROUTINE.
00D7 8E40                INC    OL          * INCREMENT RESULT BY 1.
00D8 8290                ADD    IU,L,C
00D9 F1A0          UDC   SFR    F1,L        * SKIP , IF OVERFLOW OCCURS,
00DA F190                SFL    F1,L,C      * ELSE GO TO CHS ROUTINE.
00DB 5001                TN     F0,X'01'
00DC 1506                JP     CHS
00DD 1101                LT     X'01'       * UPDATE THE CONTENTS OF TEMPORARY CARRY
00DE D120                XOR    F1,T        * (BIT 0 OF FILE REG. 1).
00DF 1506                JP     CHS
00E0 F100          CFS   SFL    F1
00E1 14AE                JP     CCR
                   *
                   * MOV & INC INSTRUCTION EXECUTIVE ROUTINE.
                   *
00E2 BE20          B10   CPY    OL,T        * COPY THE CONTENTS OF SOURCE ACC TO DEST. ACC.
00E3 0101                ELT    F1,X'01'
00E4 14D2                JP     SAM         * CONTINUE TO DECODE MOV OR INC FUNCTION.
                   *
                   * ADC SUB ADD & AND INSTRUCTION DECODING AND EXECUTIVE ROUTINE.
                   *
00E5 4202          B2C   TZ     IU,X'02'    * SKIP, IF ADC OR SUB INST.,
00E6 14FA                JP     ADN         * ELSE GO TO ADN ROUTINE.
00E7 3F96                AF     OU,X'96'    * ONE'S COMPLEMENT DIFFERENCE OF
00E8 CF06                MOV    OU,(U)      * THE CONTENTS OF THE SOURCE ACC
00E9 0079                ELT    F0,X'79'    * FROM THE DEST. ACC.
00EA CE06                MOV    OL,(U)
00EB BE20                CPY    OL,T
00EC 0101                ELT    F1,X'01'
00ED CF06                MOV    OU,(U)
00EE 01F9                ELT    F1,X'F9'
00EF F1A0                SFR    F1,L
00F0 F190                SFL    F1,L,C
00F1 5001                TN     F0,X'01'    * SKIP, IF UNDERFLOW OCCURS,
00F2 14D2                JP     SAM         * ELSE CONTINUE AT SAM TO DECODE
00F3 F130                SFR    F1,C        * ADC OR SUB FUNCTION.
00F4 5001                TN     F0,X'01'    * UPDATE THE VALUE OF TEMPORARY CARRY.
00F5 14F8                JP     CAD
00F6 F100                SFL    F1
00F7 14D2                JP     SAM
00F8 F140          CAD   SLI    F1
00F9 14D2                JP     SAM
                   *
                   * AND & ADD INSTRUCTION EXECUTIVE ROUTINE.
                   *
```

```
LOCN  CODE  FLAGS  LABELS  OP  *   OPERANDS      COMMENTS
00FA  3F86         ADN     AF      OU,X'86'    * IF ADD FUNCTION EXECUTE ADD MICRO-ROUTINE
00FB  4201                 TZ      IU,X'01'    * TO ADD THE NO. FROM SOURCE ACC TO
00FC  3F60                 AF      OU,X'60'    * THE NO. FROM DESTINATION ACC.
00FD  CF06                 MOV     OU,(U)      * IF AND FUNCTION EXECUTE AND MICRO-ROUTINE
00FE  0039                 ELT     F0,X'39'    * TO LOGICAL AND THE WORD FROM SOURCE ACC AND
00FF  CE06                 MOV     OL,(U)      * THE WORD FROM DESTINATION ACC.
0100  BE20                 CPY     OL,T
0101  C101                 ELT     F1,X'01'
0102  CF06                 MOV     OU,(U)
0103  01B9                 ELT     F1,X'B9'
0104  B220                 CPY     IU,T
0105  14D9                 JP      UDC         * CONTINUE TO DECODE AND OR ADD FUNCTION.
                   *
                   * SHIFT FUNCTION DECODING AND EXECUTIVE ROUTINE.
                   *
0106  4380         CHS     TZ      IL,X'80'    * SKIP, IF LEFT SHIFT OR NO SHIFT,
0107  1514                 JP      SRS         * ELSE PERFORM RIGHT SHIFT OR SWAP.
0108  5340                 TN      IL,X'40'    * SKIP, IF LEFT SHIFT,
0109  152A                 JP      LAC         * ELSE GO TO LAC ROUTINE.
010A  1190                 LT      X'90'       * RESULT LEFT ROTATE ONE POSITION WITH CARRY.
010B  F120                 SFR     F1
010C  16F0                 LU      X'F0'
010D  1A00                 MLC
010E  0EFF                 ELT     OL,X'FF'
010F  1A00                 MLC
0110  02FF                 ELT     IU,X'FF'
0111  F180                 SFL     F1,L
0112  1600                 LU      X'00'
0113  152A                 JP      LAC         * CONTINUE.
0114  4340         SRS     TZ      IL,X'40'    * SKIP, IF RIGHT SHIFT,
0115  151F                 JP      SWP         * ELSE GO TO SWP ROUTINE.
0116  11B0                 LT      X'B0'       * RESULT RIGHT ROTATE ONE POSITION WITH CARRY.
0117  F120                 SFR     F1
0118  16F0                 LU      X'F0'
0119  1A00                 MLC
011A  02FF                 ELT     IU,X'FF'
011B  1A00                 MLC
011C  0EFF                 ELT     OL,X'FF'
011D  F180                 SFL     F1,L
011E  152A                 JP      LAC         * CONTINUE.
011F  CE01         SWP     MOV     OL,(T)      * SWAP THE HALVES OF THE 16 BIT RESULT.
0120  1080                 SSF
0121  B120                 CPY     1,T
0122  1040                 SPF
0123  C201                 MOV     IU,(T)
0124  BE20                 CPY     OL,T
0125  1080                 SSF
0126  C101                 MOV     1,(T)
0127  1040                 SPF
0128  B220                 CPY     IU,T
0129  152A                 JP      LAC         * CONTINUE.
                   *
                   * INHIBIT LOAD FUNCTION EXECUTIVE ROUTINE.
                   *
012A  4308         LAC     TZ      IL,X'08'    * SKIP, IF TO LOAD THE RESULT TO THE DEST. ACC,
012B  153F                 JP      CSK         * ELSE GO TO CSK ROUTINE.
012C  F138                 SFR*    F1,C        * LOAD TEMPORARY CARRY INTO CARRY BIT.
```

```
012D 5001              TN     FO,X'01'     * SKIP, IF TEMPORARY CARRY = 1,
012E 153C              JP     LCR          * ELSE GO TO LCR ROUTINE.
012F 1102              LT     X'02'        * SET CARRY BIT = 1.
0130 C120              LOR    F1,T
0131 DF69      LAD     XOR*   OU,F,T,(T)   * RESTORE ACC DEST. ADDRESS.
0132 3F10              AF     OU,X'10'     * LOAD THE RESULT INTO THE DEST. ACC.
0133 8F29              ADD*   OU,T,(T)
0134 EF20              AND    OU,T
0135 CE01              MOV    OL,(T)
0136 3FB0              AF     OU,X'B0'
0137 CF06              MOV    OU,(U)
0138 0020              ELT    FO,X'20'
0139 C201              MOV    IU,(T)
013A 0120              ELT    F1,X'20'
013B 153F              JP     CSK          * CONTINUE SKIP FUNCTION DECODING.
013C 11FD      LCR     LT     X'FD'        * CLEAR THE CARRY BIT.
013D E120              AND    F1,T
013E 1531              JP     LAD          * CONTINUE LOAD FUNCTION EXECUTION.
                       *
                       * SKIP OPERATION DECODING AND EXECUTIOVE ROUTINE.
                       *
013F CE10      CSK     MOV    OL,C         * TEST THE RESULT STATUS.
0140 C290              MOV    IU,L,C
0141 1650              LU     X'50'        * SET U REG. := TN OPERATION CODE.
0142 4301              TZ     IL,X'01'     * SKIP, IF SZC, SZR, SEZ, OR NOP,
0143 1553              JP     B15          * ELSE GO TO B15(DECODE SKIP, SNC, SNR, OR SBN).
0144 5304      CRC     TN     IL,X'04'     * DECODE THE BITS CONFIGURATIONS AND
0145 154D              JP     CCY          * EXECUTE ACCORDINGLY:
0146 0004              ELT    FO,X'04'     * MNEMONIC          FUNCTION
0147 1557              JP     B14          *    SZC      010 SKIP ON ZERO CARRY.
0148 5301              TN     IL,X'01'     *    SNC      011 SKIP ON NONZERO CARRY.
0149 148C              JP     SKP          *    SZR      100 SKIP ON ZERO RESULT.
014A 5302              TN     IL,X'02'     *    SNR      101 SKIP ON NONZERO RESULT.
014B 148C              JP     SKP          *    SEZ      110 SKIP ON EITHER CARRY OR RESULT IS ZERO.
014C 154F              JP     SBN          *    SBN      111 SKIP IF BOTH CARRY CARRY & RESULT ARE
014D 5302      CCY     TN     IL,X'02'     *                 NONZERO.
014E 148E              JP     LUP
014F F138      SBN     SFS*   F1,C
0150 0001              ELT    FO,X'01'
0151 148C              JP     SKP
0152 148E              JP     LUP
0153 1640      B15     LU     X'40'        * SET U REG. := TZ OPERATION CODE.
0154 4306              TZ     IL,X'06'     * SKIP IF SKIP OPERATION.
0155 1544              JP     CRC          * ELSE GO TO CRC ROUTINE.
0156 148C              JP     SKP          * INCREMENT PC COUNTER BY 2.
0157 5301      B14     TN     IL,X'01'
0158 154D              JP     CCY
0159 148E              JP     LUP
                       *
                       * INPUT/OUTPUT INST. DECODING AND EXECUTION.
                       *
015A 2F3F      IOI     LF     OU,X'3F'     * EXTRACT DEVICE CODE AND LOAD INTO OU REG..
015B C301              MOV    IL,(T)
015C EF20              AND    OU,T
015D 2E07              LF     OL,X'07'     * EXTRACT THE TRANSFER FUNCTION FIELD AND
015E C201              MOV    IU,(T)       * LOAD INTO OL REG..
015F EE20              AND    OL,T
```

```
0160 6FC1                CP     OU,X'C1'    * SKIP, IF DEVICE CODE = 77.
0161 1583                JP     NCPU        * TRANSFER TO NCPU ROUTINE.
                         *
                         * SPECIAL FUNCTION INST. HANDLING ROUTINE.
                         *
0162 6EF9                CP     OL,X'F9'    * SKIP, IF TRANSFER FUNCTION = 7.
0163 1574                JP     CPD         * OTHERWISE, JUMP TO CPD ROUTINE.
0164 5380                TN     IL,X'80'    * SKIP, IF TEST FOR POWER FAILURE STATUS.
0165 156C                JP     SKINT
0166 1641                LU     X'41'       * CONTROL FUNCTION   SKP-- CPU
0167 5340                TN     IL,X'40'    *   10               SKIP IF POWER FAILURE IS NONZERO
0168 1651                LU     X'51'       *   11               SKIP IF POWER FAILURE IS ZERO
0169 0040                ELT    0,X'40'
016A 148E                JP     LUP
016B 148C                JP     SKP
                         *
                         * INTERRUPT ON FLAG STATUS TEST AND SKIP ROUTINE.
                         *
016C 164C        SKINT   LU     X'4C'       *   00               SKIP IF INTERRUPT ON IS NONZERO
016D 5340                TN     IL,X'40'    *   01               SKIP IF INTERRUPT ON IS ZERO
016E 165C                LU     X'5C'
016F 1080                SSF
0170 0080                ELT    0,X'80'
0171 1DC4                JP     JLUP
0172 1040                SPF
0173 148C                JP     SKP
                         *
                         * CONTROL FIELD DECODING(SET/RESET INTERRUPT ON FLAG) FOR I/O INST. WITH
                         * DEVICE CODE = 77.
                         *
0174 4380        CPD     TZ     IL,X'80'    * SKIP, IF ENABLE INTERRUPT.
0175 157D                JP     DEI
0176 5340                TN     IL,X'40'    * IF CONTROL CODE = 00 JUMP TO DCPU ROUTINE.
0177 1C8B                JP     DCPU
0178 1080                SSF                * SET IF, AND NI FLAGS ON THE SECONDARY FILE REG. 12.
0179 11C0                LT     X'C0'
017A CC20                LOR    12,T
017B 1040        EEI     SPF
017C 1C8B                JP     DCPU
017D 4340        DEI     TZ     IL,X'40'    * SKIP, IF DISABLE INTERRUPT.
017E 1C8B                JP     DCPU        * GO TO DCPU ROUTINE. (CONTROL CODE = 11.)
017F 1080                SSF                * RESET IF AND NI FLAGS  ON THE SECONDARY FILE REG. 12.
0180 113F                LT     X'3F'
0181 EC20                AND    12,T
0182 1578                JP     EEI
                         *
                         * EXTERNAL DEVICE DECODING AND EXECUTION.
                         * (DEVICE CODE = 01 TO 76 )
                         *
0183 4F30        NCPU    TZ     OU,X'30'    * TRANSFER TO UDN ROUTINE, FOR I/O INST. WITH
0184 15D6                JP     UDN         * DEVICE CODE OTHER THAN 10, 11, 12, 14
0185 5F08                TN     OU,X'08'
0186 15D6                JP     UDN
0187 3FF0                AF     OU,X'F0'    * TRAP TO THE DEVICE CODE JUMP TABLE.
0188 CF05                MOV    OU,(K)
                         *
                         * BASIC INPUT OUTPUT SUBROUTINE.
```

```
                *
0189 7090       COX     COX     0               * CONTROL OUTPUT ROUTINE. SEND THE CONTROL CODE
018A 1000               NOP
018B 158C               JP      CIO             * TO A DEVICE CONTROLLER.
018C 7080       CIO     CIO     0
018D 1020               RTN
018E 70E0       DIX     DIX     0               * DATA INPUT ROUTINE. INPUT  DATA FROM A DEVICE
018F 1590               JP      ELT             * AND LOAD INTO THE TEMPAR Y STORAGE(FILE REG. 1).
0190 B120       ELT     CPY     1,Y
0191 158C               JP      CIO
0192 70A0       DOX     DOX     0               * DATA OUTPUT ROUTINE. OUTPUT DATA TO A DEVICE.
0193 158C               JP      CIO
                *
                * DATA TRANSFER ROUTINE.
                *
0194 1040       IDATD   SPF                     * LOAD DATA FROM FILE REG. TO ACC. AS SPECIFIED
0195 3FB6               AF      OU,X'B6'        * IN OU REG..
0196 CF06               MOV     OU,(U)
0197 0020               ELT     0,X'20'
0198 0100               ELT     1,X'00'
0199 1600               LU      X'00'
019A 1020               RTN
                *
                * CLEAR I/O DEVICE  EXECUTIVE ROUTINE.
                *
019B 1080       CLIC    SSF                     * SET REAL TIME CLOCK FREQ. TO AC LINE FREQ. (50 HZ.),
019C 246C               LF      4,X'6C'         * CLEAR ALL CONTROL FLAGS.
019D 256C               LF      5,X'6C'
019E 21B9               LF      1,X'B9'
019F 028F               JE      CLEAR
01A0 148E               JP      LUP
                *
                * HALT INST. EXECUTIVE ROUTINE.
                *
01A1 8440       HALT    INC     LL              * UPDATE PROGRAM COUNTERAND STOP THE PROCESSOR.
01A2 8580               ADD     LU,L
01A3 1780               HLT
                *
                * INTERNAL HOUSE KEEPING ROUTINE.
                *
01A4 1600       INT     LU      X'00'           * CHECK SYSTEM CONDITION.
01A5 1080               SSF                     * TRANSFER TO PFR, IF INTERNAL INTERRUPT,
01A6 4010               TZ      0,X'10'         * (RTC OR POWER FAIL)
01A7 1DAE               JP      PFR             * ELSE SKIP TO NEXT INST..
01A8 5080       MAIN    TN      0,X'80'         * SKIP IF EXTERNAL INTERRUPT,
01A9 1C43               JP      TTY             * ELSE JUMP TO TTY ROUTINE.
01AA 032B               JE      IAK             * INTERRUPT ACKNOWLEDGE.
01AB 1C00               JP      CDC             * GO TO CDC ROUTINE TO EXAMINE DEVICES' CONDITION.
                *
                * ERROR ROUTINE FOR UNEXPECTED DEVICE REQUESTS FOR SERVICE
                * IN ASYNCHRONOUS INTERFACE.
                *
01AC 111C       ERA     LT      X'1C'           * SEND CONTROL BYTE TO DEVICE INTERFACE.
01AD 0189               JE      COX
01AE 5110               TN      1,X'10'         * SKIP IF OUTPUT SUB-CHANNEL,
01AF 15B3               JP      ERI             * ELSE GO TO ERI ROUTINE.
01B0 1100       ERC     LT      X'00'           * OUTPUT DATA AND DISCARD.
```

```
01B1 0192                JE     DCX
01B2 1C43                JP     TTY          * CONTINUE IN TTY ROUTINE.
01B3 018E        EPI     JE     DIX          * INPUT DATA AND DISCARD.
01B4 1C43                JP     TTY          * CONTINUE IN TTY ROUTINE.
                 *
                 * ACC ADDRESS DECODING ROUTINE.
                 *
01B5 2F18        ACA     LF     OU,X'18'     * EXTRACT AC ADDRESS AND LOAD INTO OU REG..
01B6 C201                MOV    IU,(T)
01B7 EF20                AND    OU,T
01B8 FF20                SFR    OU
01B9 FF20                SFR    OU
01BA 1020                RTN
                 *
                 * INTERRUPT ACKNOWLEDGE EXECUTIVE ROUTINE.
                 *
01BB 01B5        INA     JE     ACA          * DECODE AND MAP THE AC ADDRESS.
01BC 3F86                AF     OU,X'86'
01BD 1080                SSF
01BE CC01                MOV    12,(T)       * PLACE THE DEVICE STATUS INTO TEMPARARY STORAGE
01BF B120                CPY    1,T          * (FILE REG. 1).
01C0 DA69                XOR*   10,T,F,(T)
01C1 E120                AND    1,T          * DETERMINE THE UNMASKED AND DONE FLAG SET DEVICES.
01C2 5104                TN     1,X'04'      * SKIP, IF REAL TIME CLOCK INTERRUPT.
01C3 15CA                JP     PTT
01C4 110C                LT     X'0C'        * LOAD DEVICE CODE '0C',
01C5 1040        ACK     SPF                 * INTO THE SPECIFIED AC .
01C6 CF06                MOV    OU,(U)
01C7 0020                ELT    0,X'20'
01C8 0100                ELT    1,X'00'
01C9 148E                JP     LUP
01CA 5110        PTT     TN     1,X'10'      * SKIP, IF PTR INTERRUPT.
01CB 15CE                JP     IOT
01CC 110A                LT     X'0A'        * LOAD DEVICE CODE '0A',
01CD 15C5                JP     ACK
01CE 5102        IOT     TN     1,X'02'      * SKIP, IF TTI INTERRUPT.
01CF 15D2                JP     TYO
01D0 1108                LT     X'08'        * LOAD DEVICE CODE '08',
01D1 15C5                JP     ACK
01D2 1100        TYO     LT     X'00'        * IF TTO INTERRUPT, LOAD DEVICE CODE '09' INTO
01D3 4101                TZ     1,X'01'      * THE SPECIFIED ACC. OTHERWISE LOAD '00'.
01D4 1109                LT     X'09'
01D5 15C5                JP     ACK
                 *
                 * UNDEFINED (NOT IMPLEMENTED) DEVICE'S I/O INST. EXECUTIVE ROUTINE.
                 *
01D6 6EF9        UDN     CP     OL,X'F9'     * SKIP IF SKIP INST..
01D7 15DB                JP     UIN          * ELSE GO TO UIN ROUTINE.
01D8 5340                TN     IL,X'40'     * INCREMENT PROGRAM COUNTER BY 2,
01D9 148E                JP     LUP          * IF BUSY OR DONE FLAG = 0,
01DA 148C                JP     SKP          * ELSE INCREMENT BY 1.
01DB 5E01        UIN     TN     OL,X'01'     * SKIP IF DATA INPUT INST.,
01DC 148E                JP     LUP          * ELSE GO TO LUP ROUTINE.
01DD 01B5                JE     ACA
01DE 0298                JE     CAC          * CLEAR THE ADDRESSED ACC.
01DF 148E                JP     LUP          * CONTINUE IN LUP ROUTINE.
01E0 119F        MVF     LT     X'9F'        * MEMORY CYCLIC.
```

```
01E1 EF20                AND    0U,7
01E2 1459                JP     REP
01E3 117C          EPPT  LT     X'7C'        * RESET SEND BUFFER-EMPTY MASK.
01E4 0189                JE     COX
01E5 1110                LT     X'10'
01E6 0192                JE     DOX
01E7 111C                LT     X'1C'        * SEND CONTROL BYTE TO ASYN. INTERFACE.
01E8 0189                JE     COX
01E9 4110                TZ     1,X'10'      * SKIP , IF SEND SUBCHANNAL.
01EA 15B0                JP     ERO          * ELSE GO TO ERO ROUTINE.
01EB C601                MOV    6,(T)        * OUTPUT 'NULL' TO SILENT 700
01EC 0192                JE     DOX          * TO RELEASE SCANNER.
01ED 1C43                JP     TTY          * CONTINUE AT TTY ROUTINE.
01EE 1780          ICDR  HLT                 * ILLEGAL REQUEST HALT PROCESSOR.
01EF 4340          HZ2   TZ     IL,X'40'     * SKIP IF IDLE RTC.
01F0 148E                JP     LUP
01F1 1100                LT     X'00'        * CLEAR RTC BUSY-DONE FLAGS.
01F2 1710                DRT                 * DISABLE RTC.
01F3 1DEF                JP     HZ3
                   *
                   * ILLEGAL TTO INSTRUCTION ROUTINE.
                   *
01F4 5201          U15   TN     IU,X'01'     * SKIP IF INPUT DATA TRANSFER.
01F5 1CE1                JP     NO1          * ELSE GO TO NO1.
01F6 029B                JE     CAC          * CLEAR THE ADDRESSED ACC.
01F7 1CE1                JP     NO1          * PERFORM THE CONTROL FUNCTION.
                   *
                   * IMPLEMENTED DEVICES' JUMP TABLE.
                   *
01F8 1CA8                JP     TTI          * TELETYPE INPUT.
01F9 1CD0                JP     TTO          * TELETYPE OUTPUT.
01FA 1CF5                JP     PTR          * PAPER TAPE READER.
01FB 15D6                JP     UDN          * UNUSED.
01FC 1DCE                JP     RTC          * REAL TIME CLOCK.
01FD 15D6                JP     UDN          * UNUSED.
01FE 15D6                JP     UDN          * UNUSED.
01FF 15D6                JP     UDN          * UNUSED.
                   *
                   * CHECK DEVICE CONDITION, PERFORM THE DATA TRANSFER IF REQUIRE
                   * AND UPDATE THE DEVICE'S STATUS.
                   *
0200 113E          CDC   LT     X'3E'        * GET DEVICE NUMBER.
0201 E120                AND    1,T          * SKIP IF ASYNCHRONOUS INTERFACE (DN = 1C) REQUEST
0202 61C8                CP     1,X'C8'      * FOR SERVICE, ELSE GO TO ICDR ROUTINE.
0203 15EE                JP     ICDR
0204 113C                LT     X'3C'        * INPUT STATUS BYTE FROM ASYNCHRONOUS INTERFACE.
0205 0189                JE     COX
0206 018E                JE     DIX
0207 11F0                LT     X'F0'        * TRANSFER TO ALP ROUTINE IF CHANNEL 0
0208 E120                AND    1,T          * REQUEST SERVICE.
0209 61EF                CP     1,X'EF'      * ELSE CONTINUE.
020A 1C25                JP     ALP
020B 61A0                CP     1,X'A0'      * SKIP IF PTR REQUEST SERVICE(CHANNEL NO. = 3),
020C 15AC                JP     ERA          * ELSE GO TO ERA ROUTINE.
020D 117C                LT     X'7C'        * SEND INTERRUPT MASK CONTROL BYTE.
020E 0189                JE     COX
020F 1172                LT     X'72'        * SET SEND-BUFFER-EMPTY MASK BIT.
```

```
LOCN  CODE  FLAGS  LABELS  OP *   OPERANDS      COMMENTS
0210  0192                 JE     DOX
0211  111C                 LT     X'1C'         * INPUT DATA FROM PTR AND
0212  0189                 JE     COX           * STORE DATA IN PTR BUFFER (FILE REG. 8).
0213  018E                 JE     DIX
0214  C101                 MOV    1,(T)
0215  B820                 CPY    8,T
0216  117C                 LT     X'7C'         * SEND INTERRUPT MASK CONTROL BYTE.
0217  0189                 JE     COX
0218  1170                 LT     X'70'         * RESET DATA READY MASK BIT.
0219  0192                 JE     DOX
021A  111C                 LT     X'1C'         * DISCONNECT PTR (SET READER NOT READY).
021B  0189                 JE     COX
021C  1193                 LT     X'93'
021D  0192                 JE     DOX
021E  4B81                 TZ     11,X'81'      * TRANSFER TO TTY ROUTINE IF BOTH PTR
021F  1C21                 JP     GCON          * BUSY DONE FLAGS ARE RESET,
0220  1C43                 JP     TTY           * ELSE GO TO GCON ROUTINE.
0221  2E01           GCON  LF     11,X'01'      * SET PTR DONE FLAG.
0222  1110                 LT     X'10'         * UPDATE DEVICES' DONE-FLAG STATUS IN FILE REG. 12.
0223  CC20                 LOR    12,T
0224  1C43                 JP     TTY           * CONTINUE AT TTY ROUTINE.
0225  111C           ALP   LT     X'1C'         * SEND ASYN. INTERFACE CONTROL BYTE.
0226  0189                 JE     COX
0227  5110                 TN     1,X'10'       * SKIP, IF SEND SUBCHANNEL REQUEST FOR SERVICE,
0228  1583                 JP     ERI           * ELSE GO TO ERI ROUTINE.
0229  7110                 ESS    1
022A  4110                 TZ     1,X'10'       * SKIP, IF TTO ECHO ON SILENT 700 DATA TERMINAL,
022B  15E3                 JP     ERPT          * ELSE GO TO ERPT ROUTINE.
022C  5F80                 TN     15,X'80'      * SKIP, IF TTO BUSY FLAG ON,
022D  15E3                 JP     ERPT          * ELSE GO TO ERPT ROUTINE.
022E  117C                 LT     X'7C'         * RESET SEND BUFFER-EMPTY MASK.
022F  0189                 JE     COX
0230  1110                 LT     X'10'
0231  0192                 JE     DOX
0232  111C                 LT     X'1C'         * OUTPUT DATA TO SILENT 700 DATA TERMINAL.
0233  0189                 JE     COX
0234  C601                 MOV    6,(T)
0235  0192                 JE     DOX
0236  F310                 SFL    3,C           * TEST TIME OUT FOR CR CHARACTER.
0237  5004                 TN     0,X'04'       * SKIP , IF END OF TIME OUT OR NON CR CHAR.,
0238  1C7F                 JP     TD1           * ELSE GO TO TD1 ROUTINE.
0239  117F                 LT     X'7F'         * TRANSFER TO TD2 ROUTINE
023A  E620                 AND    6,T           * IF OUTPUT CHARACTER := CR.
023B  11F3                 LT     X'F3'
023C  8630                 ADD    6,T,C
023D  4004                 TZ     0,X'04'
023E  1C7E                 JP     TD2
023F  2600                 LF     6,X'00'       * CLEAR TTO A BUFFER.
0240  2F01                 LF     15,X'01'      * SET TTO DONE FLAG.
0241  1101                 LT     X'01'         * UPDATE DEVICES' DONE-FLAG STATUS IN FILE REG. 12.
0242  CC20                 LOR    12,T
0243  1120           TTY   LT     X'20'         * INPUT TTY STATUS.
0244  0189                 JE     COX
0245  018E                 JE     DIX
0246  C101                 MOV    1,(T)         * STORE STATUS BYTE INTO FILE REG. 1 & 9.
0247  B920                 CPY    9,T
0248  5902                 TN     9,X'02'       * SKIP IF TTI DATA READY.
```

```
0249 1C52                JP     TTO          * ELSE GO TO ITO ROUTINE.
024A 1100                LT     X'00'        * INPUT DATA FROM TTY.
024B 0189                JE     COX
024C 018E                JE     DIX
024D C101                MOV    1,(T)        * STORE DATA INTO TTI BUFFER (FILE REG. 7).
024E B720                CPY    7,T
024F 1102                LT     X'02'        * SET TTI DONE FLAG.
0250 CC20                LOF    12,T         * UPDATE DEVICES' DONE FLAG STATUS.
0251 2E01                LF     14,X'01'
0252 5F80          ITO   TN     15,X'80'     * SKIP IF TTO BUSY FLAG ON.
0253 1C61                JP     DINT         * ELSE GO TO DINT ROUTINE.
0254 7110                ESS    1
0255 5110                TN     1,X'10'      * SKIP, IF TTO ECHO ON CRT,
0256 1C61                JP     DINT         * ELSE GO TO DINT ROUTINE.
0257 5904                TN     9,X'04'      * SKIP IF TTY SEND BUFFER EMPTY ON.
0258 1C61                JP     DINT         * ELSE GO TO DINT ROUTINE.
0259 1100                LT     X'00'        * SEND DATA TRANSFER CONTROL CODE
025A 0189                JE     COX          * TO TTO INTERFACE.
025B C601                MOV    6,(T)
025C 0192                JE     DOX
025D 2600                LF     6,X'00'      * CLEAR TTO BUFFER.
025E 2F01                LF     15,X'01'     * SET TTO DONE FLAG.
025F 1101                LT     X'01'        * UPDATE DEVICES' DONE FLAG STATUS.
0260 CC20                LOF    12,T
                   *
                   * EMULATED INTERRUPT EXECUTIVE ROUTINE.
                   *
0261 5C80          DINT  TN     12,X'80'     * SKIP IF INTERRUPT ENABLE FLAG ON.
0262 1C7C                JP     IFT          * ELSE GO TO IFT ROUTINE.
0263 4C40                TZ     12,X'40'     * SKIP, IF NI FLAG IS SET,
0264 1C7A                JP     CIT          * ELSE GO TO CIT ROUTINE.
0265 113F                LT     X'3F'        * TRANSFER TO IFT ROUTINE IF UNMASKED
0266 EC29                AND*   12,T,(T)     * DEVICE REQUEST FOR SERVICE.
0267 B120                CPY    1,T
0268 DA69                XOR*   10,T,F,(T)
0269 E120                AND    1,T
026A 513F                TN     1,X'3F'
026B 1C7C                JP     IFT
026C 117F                LT     X'7F'
026D EC20                AND    12,T
026E 1040                SPF
026F 1300                LN     X'00'        * STORE RETURN ADDRESS AT MEMORY LOCATION 0.
0270 A511                WMF    LU,(T)
0271 1301                LN     X'01'
0272 A411                WMF    LL,(T)
0273 1302                LN     X'02'         * LOAD THE CONTENTS OF MEMORY LOCATION 1
0274 A000                RMF    0            * (ADDRESS OF INTERRUPT HANDLING ROUTINE)
0275 B520                CPY    LU,T         * INTO PROGRAM COUNTER.
0276 1303                LN     X'03'
0277 A000                RMF    0
0278 B420                CPY    LL,T         * CONTINUE NEXT INSTRUCTION'S DECODING AND
0279 1422                JP     FET          * EXECUTIVE OPERATIONS.
027A 11BF          CIT   LT     X'BF'        * CLEAR NI BIT.
027B EC20                AND    12,T
027C 1040          IFT   SPF                 * CONTINUE IN FET ROUTINE.
027D 1422                JP     FET
027E 23F0          TO2   LF     3,X'F0'      * SET TIME OUT COUNTER(FILE REG. 3).
```

```
027F 117C        TD1     LT      X'7C'       * SET SEND BUFFER-EMPTY MASK.
0280 0189                JE      COX
0281 1112                LT      X'12'
0282 0192                JE      DOX
0283 1C43                JP      TTY         * CONTINUE AT TTY ROUTINE.
                 *
                 * MASK OUT INST. EXECUTIVE ROUTINE.
                 *
0284 C1B5        MASK    JE      ACA         * EXTRACT ACC ADDRESS.
0285 3FC6                AF      0U,X'C6'
0286 CF06                MOV     0U,(U)
0287 0001                ELT     FC,X'01'    * LOAD THE MASK PATTERN FROM THE SPECIFIED ACC TO THE
0288 1080                SSF                 * MASK REG. (FILE REG. 10).
0289 EA20                CPY     10,T
028A 1DC4                JP      JLUP        * JUMP TO PROGRAM COUNTER UPDATING ROUTINE.
                 *
                 * SPECIFIED I/O FUNCTION INST. DECODING ROUTINE. (WITH DEVICE CODE = 77 ).
                 *
028B 8EFF        DCPL    CP      0L,X'FF'
028C 148E                JP      LUP         * TRANSFER TO LUP ROUTINE FOR NO I/O TRANSFER.
028D 3EA0                AF      0L,X'A0'
028E CE04                MOV     0L,(L)      * TRAP TO THE SPECIAL FUNCTION INST. JUMP TABLE.
                 *
                 * CLEAR A BAND OF FILE REGISTERS (FROM THE SPECIFIED FILE REG. TO FILE REG. 15).
                 *
028F C106        CLEAR   MCV     1,(U)       * CLEAR ALL THE CONTROL FLAGS.
0290 0000                ELT     0,X'00'
0291 8140                INC     1
0292 6140                CP      1,X'40'
0293 1C8F                JP      CLEAR
0294 1600                LU      X'00'
0295 1040                SPF
0296 1020                RTN
                 *
                 * ILLEGAL TTI INSTRUCTION ROUTINE.
                 *
0297 5201        UI4     TN      IU,X'01'    * SKIP IF INPUT DATA TRANSFER,
0298 1CC2                JP      NIO         * ELSE GO TO NIO
0299 029B                JE      CAC         * CLEAR THE ADDRESSED ACC.
029A 1CC2                JP      NIO         * PERFORM THE CONTROL FUNCTION.
                 *
                 * CLEAR THE SPECIFIED ACCUMULATOR.
                 *
029B 3FB6        CAC     AF      0U,X'B6'
029C CF06                MOV     0U,(U)
029D 0000                ELT     0,X'00'
029E 0100                ELT     1,X'00'
029F 1600                LU      X'00'
02A0 1020                RTN
                 *
                 * SPECIAL FUNCTION INSTRUCTION JUMP TABLE ( DEVICE CODE = 77 ).
                 *
02A1 1D5A                JP      SWT         * READ SWITCHES
02A2 148E                JP      LUP         * UNUSED
02A3 15BB                JP      INA         * INTERRUPT ACKNOWLEDGE
02A4 1C84                JP      MASK        * MASK OUT
02A5 159B                JP      CLIO        * CLEAR I/O DEVICE
```

```
02A6 19A1                JP     HALT          * HALT
02A7 148E                JP     LUP           * UNUSED
                  *
                  * TELETYPE INPUT INSTRUCTION EXECUTIVE ROUTINE.
                  *
02A8 6EF9         TTI    CP     OL,X'F9'      * SKIP IF SKP INST.,
02A9 1CBA                JP     T14           * ELSE GO TO T14.
02AA 1080                SSF
02AB CE01                MOV    14,(T)        * COPY TTI BUSY-DONE STATUS TO T REG..
                  *
                  * SKIP DEVICE BUSY-DONE STATUS ROUTINE.
                  *
02AC 1C40         CBD    SPF
02AD BF20                CPY    OU,T          * COPY BUSY-DONE STATUS TO OU REG..
02AE 1130                LT     X'30'         * SET T REG. = 30 IF SKPDN/SKPDZ,
02AF 5380                TN     IL,X'80'      * ELSE SET T REG. = 10.
02B0 1110                LT     X'10'
02B1 2E40                LF     OL,X'40'      * SET OL REG. = 40 IF SKPBN/SKPDN,
02B2 4340                TZ     IL,X'40'      * ELSE SET OL REG. = 50.
02B3 3E10                AF     OL,X'10'
02B4 1400                MLC
02B5 FF30                SFR    OU,C          * SHIFT LEFT IF TEST BUSY CONDITION,
02B6 CE06                MOV    OL,(U)        * ELSE SHIFT RIGHT.
02B7 0001                ELT    0,X'01'
02B8 148C                JP     SKP           * PERFORM SKIP/NONSKIP OPERATION.
02B9 148E                JP     LUP
                  *
                  * TTI DATA INPUT INSTRUCTION (DIA - TTI) EXECUTIVE ROUTINE.
                  *
02BA 01B5         T14    JE     ACA           * GET ACC ADDRESS.
02BB 4206                TZ     IU,X'06'      * SKIP IF LEGAL TTI INST.,
02BC 1C97                JP     U14           * ELSE GO TO U14.
02BD 6EFF                CP     OL,X'FF'      * SKIP IF DATA INPUT,
02BE 1CC2                JP     NIO           * ELSE GO TO NIO.
02BF 1080                SSF
02C0 C701                MOV    7,(T)         * INPUT DATA FROM TTI BUFFER
02C1 0194                JE     IDATA         * TO THE SPECIFIED ACC.
                  *
                  * TTI CONTROL FUNCTION DECODING.
                  *
02C2 4380         NIO    TZ     IL,X'80'      * SKIP IF 'S' FUNCTION,
02C3 1CC8                JP     CLP           * ELSE GO TO CLP.
02C4 5340                TN     IL,X'40'      * SKIP IF START THE DEVICE (TTI).
02C5 148E                JP     LUP
02C6 1080                SSF
02C7 2F80                LF     14,X'80'      * SET TTI BUSY FLAG.
02C8 11FD         SIM    LT     X'FD'         * RESET TTI DONE FLAG
02C9 EC20         RSI    AND    12,T          * IN THE DEVICES' DONE FLAG STATUS BYTE.
02CA 1DC4                JP     JLUP          * ( FILE REG. 12 )
02CB 4340         CLP    TZ     IL,X'40'      * SKIP IF IDLE THE DEVICE.
02CC 148E                JP     LUP
02CD 1080                SSF
02CE 2F00                LF     14,X'00'      * CLEAR TTI BUSY AND DONE FLAGS.
02CF 1CC8                JP     SIM
                  *
                  * TELETYPE OUTPUT INSTRUCTION EXECUTIVE ROUTINE.
                  *
```

```
02D0 6EF9          TTO     CP      DL,X'F9'    * SKIP IF SKP INST.,
02D1 1CD5                  JP      T15         * ELSE GO TO T15.
02D2 1080                  SSF
02D3 CF01                  MOV     15,(T)      * COPY TTO BUSY-DONE STATUS INTO T REG.,
02D4 1CAC                  JP      CBD         * GO TO SKIP CONDITION DECODING ROUTINE.
                   *
                   * TTO DATA OUTPUT INSTRUCTION (DOA - TTO) EXECUTIVE ROUTINE.
                   *
02D5 01B5          T15     JE      ACA         * GET ACC ADDRESS.
02D6 4205                  TZ      IU,X'05'    * SKIP IF LEGAL TTO INST.,
02D7 1DF4                  JP      U15         * ELSE GO TO U15.
02D8 6EFF                  CP      DL,X'FF'    * SKIP IF DATA OUTPUT ,
02D9 1CE1                  JP      N01         * ELSE GO TO N01.
02DA 3FC6                  AF      OU,X'C6'    * COPY THE CONTENTS OF SPECIFIED ACC
02DB CF06                  MOV     OU,(U)      * INTO TTO BUFFER.
02DC 0001                  ELT     0,X'01'
02DD 1600                  LU      X'00'
02DE 1080                  SSF
02DF B620                  CPY     6,T
02E0 1040                  SPF
                   *
                   * TTO CONTROL FUNCTION DECODING.
                   *
02E1 4380          N01     TZ      IL,X'80'    * 'C'/'P' OPERATION ?
02E2 1CF0                  JP      CL1         * YES, GO TO CL1.
02E3 5340                  TN      IL,X'40'    * SKIP IF START THE DEVICE.
02E4 148E                  JP      LUP
02E5 1080                  SSF
02E6 7110                  ESS     1           * TRANSFER TO BD1 ROUTINE
02E7 4110                  TZ      1,X'10'     * IF TTO ECHO ON CRT.
02E8 1CED                  JP      BD1         * CONTINUE AT BD1 ROUTINE.
02E9 117C                  LT      X'7C'       * SET SEND BUFFER EMPTY MASK.
02EA 0189                  JE      C0X
02EB 1112                  LT      X'12'
02EC 0192                  JE      D0X
02ED 2F80          BD1     LF      15,X'80'    * SET TTO BUSY FLAG.
02EE 11FE          PN0     LT      X'FE'       * RESET THE TTO DONE FLAG IN
02EF 1CC9                  JP      RSI         * THE DEVICES' DONE FLAG STATUS BYTE.
02F0 4340          CL1     TZ      IL,X'40'    * SKIP IF IDLE THE DEVICE.
02F1 148E                  JP      LUP
02F2 1080                  SSF
02F3 2F00                  LF      15,X'00'    * CLEAR TTO BUSY-DONE FLAGS.
02F4 1CEE                  JP      PN0
                   *
                   * PAPER TAPE READER INSTRUCTION EXECUTIVE ROUTINE.
                   *
02F5 6EF9          PTR     CP      DL,X'F9'    * SKIP IF SKP INST.,
02F6 1CFA                  JP      T11         * ELSE GO TO T11.
02F7 1080                  SSF
02F8 CB01                  MOV     11,(T)      * COPY PTR BUSY-DONE STATUS TO T REG.,
02F9 1CAC                  JP      CBD         * GO TO SKIP CONDITION DECODING ROUTINE.
                   *
                   * PTR DATA INPUT INSTRUCTION (DIA - PTR) EXECUTIVE ROUTINE.
                   *
02FA 01B5          T11     JE      ACA         * GET ACC ADDRESS.
02FB 4206                  TZ      IU,X'06'    * SKIP IF LEGAL PTR INST.,
02FC 1DC6                  JP      U11         * ELSE GO TO U11.
```

```
02FD 6EFF            CP     OL,X'FF'    * SKIP IF DATA INPUT,
02FE 1D02            JP     NO5         * ELSE GO TO NO5.
02FF 1080            SSF
0300 C801            MOV    8,(T)       * INPUT DATA FROM PTR BUFFER
0301 0194            JE     IDATA       * TO THE SPECIFIED ACC.
                     *
                     * PTR CONTROL FUNCTION DECODING.
                     *
0302 4380     NO5    TZ     IL,X'80'    * 'C'/'P' OPERATION ?
0303 1D26            JP     TR1         * YES, GO TO TR1.
0304 5340            TN     IL,X'40'    * SKIP IF START THE DEVICE.
0305 148E            JP     LUP
0306 1080            SSF
0307 117C            LT     X'7C'       * SEND BUFFER EMPTY MASK.
0308 0189            JE     COX
0309 1172            LT     X'72'
030A 0192            JE     DOX
030B 5080     STM    TN     0,X'80'     * SKIP IF EXTERNAL INTERRUPT, ELSE WAIT.
030C 1D0B            JP     STM
030D 032B            JE     IAK         * INTERRUPT ACKNOWLEDGE.
030E 113E            LT     X'3E'       * GET DEVICE CODE.
030F E120            AND    1,T
0310 61C8            CP     1,X'C8'     * SKIP IF 8 CHANNEL ASYCHONOUSE INTERFACE,
0311 1780            HLT                * ELSE HALT PROCESSOR.
0312 113C            LT     X'3C'       * INPUT STATUS BYTE.
0313 0189            JE     COX
0314 018E            JE     DIX
0315 11F0            LT     X'F0'       * TRANSFER TO EEE ROUTINE IF CHANNEL 0
0316 E120            AND    1,T         * (TTO) REQUEST FOR SERVICE.
0317 61EF            CP     1,X'EF'
0318 1D31            JP     EEE
0319 6190            CP     1,X'90'     * SKIP IF SEND BUFFER EMPTY,
031A 1D55            JP     NOG         * ELSE GO TO NOG.
031B 117C            LT     X'7C'       * CLEAR SEND BUFFER EMPTY MASK.
031C 0189            JE     COX
031D 1170            LT     X'70'
031E 0192            JE     DOX
031F 111C            LT     X'1C'       * READY PTR.
0320 0189            JE     COX
0321 1191            LT     X'91'
0322 0192            JE     DOX
0323 2B80            LF     11,X'80'    * SET PTR BUSY FLAG.
0324 11EF     TR2    LT     X'EF'       * RESET PTR DONE FLAG IN THE
0325 1CC9            JP     RSI         * DEVICES' DONE FLAG STATUS BYTE.
0326 4340     TR1    TZ     IL,X'40'    * SKIP IF IDLE THE DEVICE.
0327 148E            JP     LUP
0328 1080            SSF
0329 2B00            LF     11,X'00'    * CLEAR PTR BUSY-DONE FLAGE.
032A 1D24            JP     TR2
032B 70D0     IAK    IAK    0           * SEND INTERRUPT ACKNOWLEDEGE CONTROL ORDER,
032C 1000            NOP
032D 1D2E            JP     CIAK        AND RECEIVE THE DEVICE NO. IN FILE REG 1.
032E B120     CIAK   CPY    1,T
032F 7080            CIO    0
0330 1C20            RTN
0331 5110     EEE    TN     1,X'10'     * SKIP IF SEND SUBCHANNEL REQUEST
0332 1D51            JP     INE         * FOR SERVICE, ELSE GO TO INE ROUTINE.
```

```
0333 117C              LT     X'7C'        * RESET SEND BUFFER-EMPTY MASK.
0334 0189              JE     COX
0335 1110              LT     X'10'
0336 0192              JE     DOX
0337 111C              LT     X'1C'        * OUTPUT DATA TO SILENT 700 DATA TERMINAL.
0338 0189              JE     COX
0339 C601              MOV    6,(T)
033A 0192              JE     DOX
033B 5F80              TN     15,X'80'     * SKIP , IF TTO BUSY FLAG ON,
033C 1D0B              JP     STM          * ELSE GO TO STM ROUTINE.
033D F310              SFL    3,C          * TEST TIME OUT FOR CR CHARACTER.
033E 5004              TN     0,X'04'      * SKIP, IF END OF TIME OUT OR NON CR CHAR..
033F 1D4C              JP     TD3          * ELSE GO TO TD3 ROUTINE.
0340 117F              LT     X'7F'        * TRANSFER TO TD4 ROUTINE
0341 E620              AND    6,T          * IF OUTPUT CHAR. := CARRIAGE RETURN.
0342 11F3              LT     X'F3'
0343 E630              ADD    6,T,C
0344 4004              TZ     0,X'04'
0345 1D48              JP     TD4
0346 2600              LF     6,X'00'      * CLEAR TTO A BUFFER.
0347 2F01              LF     15,X'01'     * SET TTO DONE FLAG.
0348 1101              LT     X'01'        * UPDATE DEVICES' DONE FLAG STATUS IN FILE REG. 12.
0349 CC20              LOF    12,T
034A 1D0B              JP     STM          * CONTINUE AT STM ROUTINE.
034B 23F0      TD4     LF     3,X'F0'      * SET TIME OUT COUNTER(FILE REG. 3).
034C 117C      TD3     LT     X'7C'        * SET SEND BUFFER-EMPTY MASK.
034D 0189              JE     COX
034E 1112              LT     X'12'
034F 0192              JE     DOX
0350 1D0B              JP     STM          * CONTINUE AT STM ROUTINE.
0351 111C      INE     LT     X'1C'        * INPUT DATA AND DISCARD.
0352 0189              JE     COX
0353 018E              JE     DIX
0354 1D0B              JP     STM          * CONTINUE AT STM ROUTINE.
0355 111C      NOG     LT     X'1C'        * OUTPUT 'NULL' CHAR. TO FREE SCANNER.
0356 0189              JE     COX
0357 1100              LT     X'00'
0358 0192              JE     DOX
0359 1D0B              JP     STM
                *
                * READ SWITCH INST. DECODING AND EXECUTIVE ROUTINE.
                *
035A 01B5      SWT     JE     ACA          * EXTRACT AND MAP AC ADDRESS.
035B 3F06              AF     0U,X'06'
035C 1120      ECP     LT     X'20'        * CLEAR THE ADDRESSED AC.
035D 8F2E              ADD*   0U,T,(U)
035E 0000              ELT    0,X'00'
035F 0100              ELT    1,X'00'
0360 1600              LU     X'00'
0361 1153              LT     X'53'        * SET T REG. := 'S'.
0362 1080              SSF
0363 297F              LF     9,X'7F'      * SET INPUT INDICATOR.
0364 7110      CCP     ESS    1            * TRANSFER TO EPT ROUTINE IF ECHO ON
0365 5110              TN     1,X'10'      * SILENT 700 DATA TERMINAL,
0366 1DA7              JP     EPT          * ELSE ECHO ON CRT.
0367 B120              CPY    1,T
0368 1100              LT     X'00'
```

```
0369 0189        SWE     JE      COX
036A C101                MOV     1,(T)
036B 0192                JE      DOX             * OUTPUT DATA ON CRT SCREEN.
036C F950                SLT     9,C             * UPDATE INPUT INDICATOR.
036D 4001                TZ      F0,X'01'        * WAIT FOR KEYBOARD INPUT?
036E 1D79                JP      CDAT            * NO, INPUT DATA READY.
036F 1120        AGAIN   LT      X'20'           * CHECK TTY STATUS.
0370 0189                JE      COX
0371 018E                JE      DIX
0372 5102                TN      1,X'02'         * KIP, IF INPUT DATA READY.
0373 1D6F                JP      AGAIN
0374 1100                LT      X'00'           * INPUT KEYBOARD DATA.
0375 0189                JE      COX
0376 018E                JE      DIX
0377 C101                MOV     1,(T)
0378 1D64                JP      CCP             * INPUT DATA CHECKING AND DISPLAY.
                 *
                 * KEYBOARD INPUT DATA CHECKING ROUTIN.
                 *
0379 F100        CDAT    SFL     1
037A 111A                LT      X'1A'
037B 9138                SBT*    1,T,C
037C 5004                TN      F0,X'04'        * SKIP, IF CARRY RETURN.
037D 1D8E                JP      CKD             * CHECK FOR LEGAL DIGIT.
037E 7110                ESS     1               * TRANSFER TO ECRT ROUTINE IF TTY ECHO ON CRT.
037F 4110                TZ      1,X'10'         * ELSE CONTINUE.
0380 1D89                JP      ECRT
0381 117C                LT      X'7C'           * RESET SEND BUFFER-EMPTY MASK.
0382 0189                JE      COX
0383 1110                LT      X'10'
0384 0192                JE      DOX
0385 111C                LT      X'1C'           * OUTPUT 'NULL' CHAR. TO SILENT 700 DATA TERMINAL.
0386 0189                JE      COX
0387 1100                LT      X'00'
0388 1192                LT      DOX
0389 1040        ECRT    SPF
038A 5108                TN      F1,X'08'        * SKIP, IF SYSTEM INITIALIZATION.
038B 148E                JP      LUP
038C 2100                LF      F1,X'00'        * RESET SYSTEM START BIT.
038D 15A4                JP      INT             * GO TO INT ROUTINE.
                 *
                 * INPUT DATA CHECKING ROUTINE.
                 *
038E 61A0        CKD     CP      1,X'A0'         * IF DATA IS LEGAL ACTUAL DIGIT LOAD DATA
038F 1D92                JP      BEGIN           * INTO THE SPECIFIED ACC;
0390 6191                CP      1,X'91'         * OTHERWISE, RE-START THE OPERATION.
0391 1D94                JP      SHB
0392 1040        BEGIN   SPF
0393 1D5C                JP      ECP
                 *
                 * STACK INPUT DIGITS INTO THE SPECIFIED ACC.
                 *
0394 F121        SHB     SFR     1,(T)           * USED IL AS A COUNTER.
0395 1040                SPF                     * SHIFT THE ADDRESSED AC LEFTWISE 3 POSITIONS.
0396 B220                CPY     IU,T
0397 23C0                LF      IL,X'C0'
0398 11F0        SHA     LT      X'F0'
```

```
0399 8F2E              ADD*  CU,T,(U)
039A 0000              ELT   0,X'00'
039B 0180              ELT   1,X'80'
039C F310              SFL   IL,C
039D 4001              TZ    FO,X'01'
039E 1D98              JP    SHA
039F 11C0              LT    X'C0'          * ADD THE LEAST SIGNIFICANT 3 BITS OF INPUT DATA
03A0 8F2E              ADD*  CU,T,(U)       * TO THE LOWER 3 BIT POSITIONS OF THE SPECIFIED AC.
03A1 1107              LT    X'07'
03A2 E221              AND   IU,T,(T)
03A3 0020              ELT   0,X'20'
03A4 1600              LU    X'00'
03A5 1080              SSF
03A6 1D6F              JP    AGAIN          * WAIT FOR ANOTHER INPUT DIGIT FROM KEYBOARD.
03A7 8120       EPT    CPY   1,T            * SET SEND BUFFER-EMPTY MASK.
03A8 117C              LT    X'7C'
03A9 C189              JE    COX
03AA 1112              LT    X'12'
03AB 0192              JE    DOX
03AC 111C              LT    X'1C'
03AD 1D69              JP    SWE            * CONTINUE AT SWE ROUTINE.
                *
                * INTERNAL INTERRUPT DECODING AND EXECUTIVE ROUTINE.
                *
03AE 7140       PFR    EIS   1              * INPUT INTERNAL STATUS BYTE.
03AF 5184              TN    1,X'84'        * SKIP IF POWER FAIL OR RTC INTERRUPT,
03B0 1780              HLT                  * ELSE HALT THE SYSTEM.
03B1 5180              TN    1,X'80'        * SKIP IF POWER FAIL INTERRUPT,
03B2 1DB7              JP    IRT            * ELSE GO TO RTC INTERRUPT ROUTINE.
03B3 1140              LT    X'40'          * SET POWER FAIL FLAG.
03B4 1040              SPF
03B5 C120              LOR   F1,T
03B6 1080              SSF
03B7 5104       IRT    TN    1,X'04'        * SKIP IF RTC INTERRUPT,
03B8 15A8              JP    MAIN           * ELSE GO TO MAIN ROUTINE.
03B9 8450              INC   4,C            * INCREMENT RTC COUNTER BY 1.
03BA 5001              TN    0,X'01'        * SKIP IF OVERFLOW OCCURS,
03BB 15A8              JP    MAIN           * ELSE GO TO MAIN ROUTINE.
03BC 5D80              TN    13,X'80'       * SKIP IF RTC BUSY BIT WAS SET,
03BD 1DC1              JP    NEG            * ELSE GO TO NEG ROUTINE.
03BE 1104              LT    X'04'          * SET RTC DONE FLAG,
03BF CC20              LOR   12,T           * AND UPDATE DEVICES' DONE FLAG STATUS.
03C0 2D01              LF    13,X'01'
03C1 C501       NEG    MOV   5,(T)          * RESET RTC COUNTER.
03C2 8420              CPY   4,T
03C3 15A8              JP    MAIN           * CONTINUE AT MAIN ROUTINE.
03C4 1040       JLUP   SPF
03C5 148E              JP    LUP
                *
                * ILLEGAL PTR INSTRUCTION EXECUTIVE ROUTINE.
                *
03C6 5201       U11    TN    IU,X'01'       * SKIP, IF INPUT DATA TRANSFER,
03C7 1D02              JP    NO5            * ELSE GO TO NO5.
03C8 0298              JE    CAC            * CLEAR THE ADDRESSED ACC.
03C9 1D02              JP    NO5            * PERFORM THE CONTROL FUNCTION.
                *
                * ILLEGAL RTC INSTRUCTION EXECUTIVE ROUTINE.
```

```
                *
03CA 5201       U13     TN      IU,X'01'    * SKIP, IF INPUT DATA TRANSFER,
03CB 1DEA               JP      NO3         * ELSE GO TO NO3.
03CC 029B               JE      CAC         * CLEAR THE ADDRESSED ACC.
03CD 1DEA               JP      NO3         * PERFORM THE CONTROL FUNCTION.
                *
                * REAL TIME CLOCK INSTRUCTION EXECUTIVE ROUTINE.
                *
03CE 6EF9       RTC     CP      OL,X'F9'    * SKIP IF SKP INST.,
03CF 1DD7               JP      T13         * ELSE GO TO T13.
03D0 1080               SSF                 * COPY RTC BUSY-DONE STATUS TO T REG.
03D1 CD01               MOV     13,(T)
03D2 1CAC               JP      CBD         * GO TO SKIP CONDITION DECODING ROUTINE.
03D3 1176       HZ1     LT      X'76'       * LOAD THE INITIAL RTC COUNTER VALUE:
03D4 4E01               TZ      OL,X'01'    * SET T = 76 IF 100 HZ (10)
03D5 117F               LT      X'7F'       *       T = 7F     1000 HZ (11)
03D6 1DE5               JP      SCT         * GO TO RTC FREQ. SETTING ROUTINE.
                *
                * RTC INSTRUCTION EXECUTIVE ROUTINE.
                *
03D7 4205       T13     TZ      IU,X'05'    * SKIP, IF LEGAL RTC INST.,
03D8 1DCA               JP      U13         * ELSE GO TO U13.
03D9 6EFF               CP      OL,X'FF'    * SKIP IF DATA OUTPUT,
03DA 1DEA               JP      NO3         * ELSE GO TO NO3.
03DB 01B5               JE      ACA         * GET ACC ADDRESS.
03DC 3FC6               AF      OU,X'C6'    * ACC ADDRESS MAPPING, AND
03DD CF06               MOV     OU,(U)      * DETERMINE THE CLOCK FREQENCE.
03DE 0001               ELT     0,X'01'     * LOAD THE CONTENTS OF ADDRESSED ACC
03DF BE20               CPY     OL,T        * INTO OL REG..
03E0 4E02               TZ      OL,X'02'    * SKIP IF 50/10 HZ,
03E1 1DD3               JP      HZ1         * ELSE GO TO HZ1.
03E2 111C               LT      X'1C'       * LOAD THE INITIAL RTC COUNTER VALUE:
03E3 5F01               TN      OL,X'01'    * SET T = 6C IF 50 HZ (00)
03E4 116C               LT      X'6C'       *       T = 1C    10 HZ (01)
03E5 1080       SCT     SSF                 * LOAD THE INITIAL FREQ. VALUE
03E6 B520               CPY     5,T         * INTO RTC COUNTER.
03E7 B420               CPY     4,T
03E8 1720               ERT                 * ENABLE RTC .
03E9 1040               SPF
03EA 4380       NO3     TZ      IL,X'80'    * TRANSFER TO HZ2 IF C/P RTC FUNCTION.
03EB 15EF               JP      HZ2
03EC 5340               TN      IL,X'40'    * SKIP, IF RTC START FUNCTION,
03ED 148E               JP      LUP         * ELSE GO TO LUP ROUTINE.
03EE 1180               LT      X'80'       * SET RTC BUSY FLAG.
03EF 1080       HZ3     SSF                 * RESET THE RTC DONE FLAG IN
03F0 BD20               CPY     13,T        * THE DEVICES' DONE FLAG STATUS BYTE.
03F1 11FB               LT      X'FB'
03F2 FC20               AND     12,T
03F3 1DC4               JP      JLUP        * CONTINUE OPERATION.
                        END
```

REFERENCES

[1] Samin S. Husson, "Microprogramming: Principles and Practices" Prentice-Hall, Inc. 1970.

[2] Efrem G. Mallach, "Emulator Architecture" Computer, August 1975, p. 24-31.

[3] Ashok K. Agrawala, Tomlinson G. Rauscher, "Foundations of Microprogramming, Architecture, Software, and Application" Academic Press, Inc. 1976.

[4] Guy G. Boulaye, "Microprogramming" The MacMillan Press Ltd. 1975.

[5] Andrew S. Tanenbaum, "Structured Computer Organization" Prentice-Hall, Inc. 1976.

[6] Maurice Wilkes, "The Best Way to Design An Automatic Calculating Machine" Proceedings of the Manchester University Computer Inaugural Conference. London: Ferranti; July 1951.

[7] "How to use the NOVA Computers", Data General Corporation, 1971.

[8] "Microprogramming Handbook", Microdata Corporation, 1972.

[9] "Computer Reference Manual — MICRODATA 1600/30", Microdata Corporation 1973.

[10] Alan B. Salisbury, "Microprogrammable Computer Architectures", American Elsevier Publishing Company Inc., 1976.

[11] S.G. Tucker, "Emulation of Large System", Communications of the ACM, December 1965, p. 753-762.

[12] Robert W. Cook, Michael J. Flynn, "System Design of a Dynamic Microprocessor", IEEE Transactions on Computers, March 1970, p. 213-222.

[13] Michael J. Flynn, Robert F. Rosin, "Microprogramming: An Introduction and a Viewpoint", IEEE Transactions on Computers, July 1971, p. 727-731.

[14] P.M. Davies, "Readings in Microprogramming", IBM Systems Journal, No. 1, 1972, p. 16-40.

[15] W.T. Wilner, "Design of the Burroughs B1700", AFIPS Conference Proceedings, Fall Joint Computer Conference, 1972, p. 489-497.

[16] S.H. Fuller, V.R. Lesser, C.G. Bell and C. Kaman, "Microprogramming and its Relationship to Emulation and Technology", Infotech State of the Art Report 23, 1975, p. 409-431.

[17] Michael J. Flynn, Lee W. Hoevel and Charles J. Nenhauser, "The Stanford Emulation Laboratory", June 1976, Technical Report No. 118, Digital System Laboratory, Stanford University.

VITA

NAME: Shing-Choy Hung

DATE OF BIRTH: November 30, 1946

PLACE OF BIRTH: Hong Kong

EDUCATION: National Taiwan University
Taipei, Taiwan
B.Sc. in Electrical Engineering (1970)