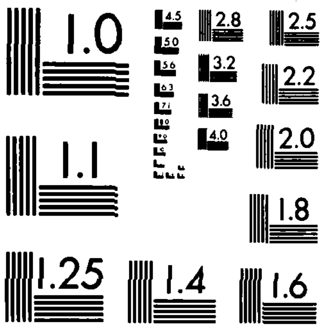
AD-A155 164 EVALUATION AND IMPLEMENTATION OF A FUNCTIONAL MICROPROGRAM GENERATOR(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA D B STILTNER DEC 84 1/3

UNCLASSIFIED F/G 9/2 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

(2)

AD-A155 164

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

EVALUATION AND IMPLEMENTATION OF A
FUNCTIONAL MICROPROGRAM GENERATOR

by

Deborah Regina Stiltner

December 1984

Thesis Advisor: Alan A. Ross

85 5

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A155114	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (*and Subtitle*) Evaluation and Implementation of a Functional Microprogram Generator		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis December 1984
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Deborah Regina Stiltner		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943		12. REPORT DATE December 1984
		13. NUMBER OF PAGES 222
14. MONITORING AGENCY NAME & ADDRESS (*If different from Controlling Office*)		15. SECURITY CLASS. (*of this report*)
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (*of this Report*)

Approved for public release; distribution is unlimited

17. DISTRIBUTION STATEMENT (*of the abstract entered in Block 20, if different from Report*)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (*Continue on reverse side if necessary and identify by block number*)

Microprogramming, microcode generator, microcode, computer-aided design, computer design, instruction set design, control unit, functional programming, menu-driven.

20. ABSTRACT (*Continue on reverse side if necessary and identify by block number*)

When a programmer writes a microprogram, as a part of a machine's instruction set or to implement an algorithm in microcode for faster execution, he must be concerned with the smallest details of the hardware in the machine. Microprogramming exists at the lowest (closest to the machine) level and is the most tedious computer "language" to program. In the field of computer design, where microprogramming is used extensively, designers use microprogramming to perfect instruction sets and to optimize frequently used routines.

A computer-aided design tool called a microcode generator is proposed in this thesis. It is an interactive, menu-driven functional programming tool. The user builds a microroutine by selecting functions from a series of menus as they are presented in a logical sequence. It is implemented in the language C on the Naval Postgraduate School Computer Science Department's VAX 780 computer using the Unix program development system components. The microcode generator is designed to produce microroutines targeted for a specific machine, the Am29203 Evaluation Board, an ALU implemented in bit-slice components.

Evaluation and Implementation of a Functional Microprogram Generator

by

Deborah Regina Stiltner
Lieutenant, United States Navy
B.S. in Applied Science, Miami University, 1977

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ENGINEERING SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1984

Author: Deborah Regina Stiltner
Deborah Regina Stiltner

Approved by: Alan A. Ross
Alan A. Ross, Thesis Advisor

[illegible]
Herschel H. Loomis, Second Reader

[illegible]
Bruce J. MacLennan, Chairman,
Department of Computer Science

[illegible]
John N. Dyer,
Dean of Science and Engineering

ABSTRACT

When a programmer writes a microprogram, as a part of a machine's instruction set or to implement an algorithm in microcode for faster execution, he must be concerned with the smallest details of the hardware in the machine. Microprogramming exists at the lowest (closest to the machine) level and is the most tedious computer "language" to program. In the field of computer design, where microprogramming is used most extensively, designers use microprogramming to perfect instruction sets and to optimize frequently used routines.

A computer-aided design tool called a microcode generator is proposed in this thesis. It is an interactive, menu-driven functional programming tool. The user builds a microroutine by selecting functions from a series of menus as they are presented in a logical sequence. It is implemented in the language C on the Naval Postgraduate School Computer Science Department's VAX 780 computer using the Unix program development system components. The microcode generator is designed to produce microroutines targeted for a specific machine, the Am29203 Evaluation Board, an ALU implemented in bit-slice components.

TABLE OF CONTENTS

ACKNOWLEDGEMENT

Thanks to family and special friends for their support during my eleven quarters at NPS. Special thanks to Col. Ross, my advisor, who, besides advising, wrote a significant amount of code in support of this thesis project.

I. INTRODUCTION

A. MICROCODE GENERATOR PROPOSED

Programming in microcode is not like programming in a structured language like Pascal or like machine level assembly language. High order language programs aren't written with concern for computer hardware or any of the internal details concerning program execution. Programmers concerned with the contents of registers; say, compiler writers, must be more familiar with the internal components of the computer. They can't easily generalize their programs to work with several different computers like a Fortran programmer, but it can be done. Microcode programmers must be even more specific in their concern for the hardware.

The microcode programmer is most likely to be a member of the design team for a new processor or specifically involved with computer control unit implementation. The ones and zeros (which make up the microword) are direct signals to the hardware components. Designs for machine instruction sets and supporting registers, PROMs (Programmable ROMs), control lines, etc. involve extensive detail, at a level much closer to the machine, which the microprogrammer must know.

Computer aided design is a relatively new field which sprang into being when we needed more complex computers to solve more complex problems. Designers could no longer hold all the detail in their heads, they needed a computer to help in the design process. For example, computer systems generate circuit diagrams in seconds; graphics stations display VLSI drawings, zooming in on selected areas and modifying them with a few commands from the user. Few areas of computer design are left where tedious tasks haven't been automated. One such area is microprogramming.

A microcode generator could aid the designer in several ways. The generator promotes structured design since it hides the details of how the microword is produced. The user could select a function such as "add the contents of two registers" and the generator would provide the microword which accomplishes the task. The function of microcoding could be modularized so that the user only needs to know what input is required and what output will be given by the generator. The importance of modularization and information hiding was introduced by D. L. Parnas in 1972. [Ref. 1] A microcode generator could also help reduce the simple coding errors easily made such as writing a '1' instead of a '0', or putting the '1' or '0' in the wrong bit position.

A good comparison of the use of a microcode generator would be to the use of a calculator. One knows how to add large strings of numbers using pencil and paper but the overall task is done faster, accurately and without tedious sidetrips by using the calculator. The difference between the microcode generator and the calculator is that while a parent may worry that with a calculator his sixth grader might forget the multiplication tables; the chief designer needn't worry that the use of a microcode generator might spoil his young team members. The point is that the programming level required to use a microcode generator is already that of expert microprogrammer. The generator merely frees the programmer to think higher level thoughts, and thus, is truly a needed utility in the area of computer aided design.

I have implemented a microcode generator for a specific machine; the Am29203 Evaluation Board which will be described in the following chapters. The generator is written in the high-level language C. C was chosen since a well defined standard for the language exists so future "porting" of the program to other machines will be easier. The utility program is designed to be interactive with the user and operates in real time like a language interpreter. The user selects functions from the menus displayed and is provided the appropriate microword. Microwords are saved in microroutines which can be stored for later use. The C program is currently implemented on the NPS VAX/Unix system. Future research plans are to modify it for use with microcomputers.

Chapter two will provide some background information on microprogramming and the microcode generator targeted machine, the Am29203 Evaluation Board. Chapter three will discuss the design approach used in this project. Chapter four will address specific implementation problems and data structures used. It will also introduce a preliminary user's manual. Chapter five will contain some conclusions and recommendations.

II. BACKGROUND

A. MICROPROGRAMMING

In order to discuss microprogrammed computer architecture, a reference point must be established. Computer architecture is the study of the systematic method in which the basic components of computers are arranged and interconnected. A convenient reference point is the high-level abstraction of a general computer system called the von Neumann machine. This idealized concept of the essential organization of a computer, developed in the mid 1940's, was the key to the construction of early computers. The von Neumann machine consists of five basic elements; the Control Unit, Arithmetic/Logic Unit (ALU), memory, input and output. In today's computer systems the elements containing the Control Unit and the ALU have been combined to form the Central Processing Unit (CPU) or the processor.

The Control Unit implements the designed machine level instruction set. The machine level instructions or macroinstructions can be executed in two ways. Implementation in hardware using combinatorial lo[illegible]s the method used in early computers and by some high speed machine[illegible]oday. However, the focus of this thesis is on the modern method [illegible]ich implements the control unit using memory or stored logic. The conte[illegible]s of the stored logic are called microinstructions. A microprogrammed control unit is one in which the more complex macroinstructions (op codes) are interpreted (executed one at a time) by sequences of primitive microinstructions which are stored in the special memory

B. *MAN/MACHINE INTERFACE CONSIDERATIONS*

Man/Machine interface discussions usually center on graphics capabilities. However, the design of the microcode generator is not so concerned with the way a general user views the screen, but with the way an expert user thinks through the problem of microprogramming. As stated in an earlier chapter, the user is assumed to be an expert in microprogramming and familiar with the specific machine for which this program is written to produce microcode. *This new utility will improve* his productivity and provide ease and flexibility in his design work. As he uses the program, he should be comfortable with the sequence of menus presented so that as he decides what the next step of the microprogram should be, the method to take that next step is before him on the screen as one of the menu selections. The most common theme heard from interactive systems designers is embodied in Hansen's First Principle: Know the User. [Ref. 11] This principle is of primary concern in this software implementation project.

There are more general human characteristics which must be dealt with in any interactive design. Humans have a short term memory capacity of only about 5-7 objects. Memorization can be minimized by using function selection versus command entry methods and using names instead of numbers for the choices. The microcode generator program menus do use names except when numbers are the object of the selection. "Muscle memory" refers to the idea that users develop the feel for frequently used keypresses. This means that the same function in different menus should be initiated in the same way; help or return to higher level, for example.

manipulation, it would have to violate the Security Principle. This principle states that no program should violate the definition of the language, or its own intended structure. [Ref. 9: p. 7] The C language contains powerful features that give it considerable flexibility. Like other languages in the structured category, C supports strong data typing, makes extensive use of pointers, and has a rich set of operators for computation and data manipulation. [Ref. 7: p. 277] So the definition of the C language contains the needed features that Pascal must violate its own definition to effect.

Modularity, information hiding and portability are a necessary influence on the design of this project. In order to achieve portability to other compilers and machines, some care is needed to provide for information hiding, i.e., generality within the modules and passing the machine and/or compiler dependent data through the interfaces between modules. Thus the modules themselves can be used for any machine or compiler and only the interfaces need be changed. The C language has a preprocessor feature which allows constants to be defined (using #define statements) and placed in a separate module to be "included" (using #include statements) as a separate file. Thus, we could have separate definition "include" files for different machines and compilers. This is a standard feature of the language.

The above principles are associated with good design and programming practices. Using them as guidelines for the selection of the basic hardware and software tools for this project ensures quality of the end product.

Such a large, complicated and interactive program is more easily developed on a medium to large computer. Using a microcomputer for the program development would involve more work on the programmer's part to manage a small memory, the storage of data arrays and the overhead of the interactive feature. The VAX 780 computer running the Unix operating system was chosen for its easy-to-use program development system, convenience (located on campus), and because previous thesis work on this project had been done on the VAX.

The programming language used to implement the program needed to be a modern structured high level language available for use on the VAX but portable enough to allow the program to be adapted for use at a microcomputer workstation. I chose the C language for its portability, since a well-defined and accepted standard had been established by its authors, Kernighan and Ritchie, [Ref. 10] and for its success as a systems programming language. The latter is important since quick, efficient code and access to system functions, the characteristics of a systems programming language, is needed in an interactive program performing in real time. The previous implementation of the microcode sequencer portion of the target machine had been written in Berkeley Pascal. I chose to rewrite this section in C since the two languages employ different parameter passing modes and data couldn't be passed between the different modules. Pascal, a language designed primarily as an educational tool for teaching structured languages, was also ruled out because of its limited capabilities in input/output processing. One of the main differences between Pascal and C is C's capabilities in systems programming. For Pascal to do some of the same sort of data in the design approach to the problem of building a microcode generator.

A. USING SOFTWARE ENGINEERING THEORY

The selection of the program development system and programming language to use depends on the main features of the project to be designed. The microcode generator utility is interactive, featuring menu-driven versus command-driven microinstruction functions, i.e., the functions are selected through a series of menus presented on a monitor. The program acts very much like an interpreter; a one-pass, step-by-step translator. The microcode generator functions are modularized and separated into function procedures which can be offered as possible selections by the user in one menu on one monitor screen. The combination of this modularity and the use of menu-driven function calls support the theory that structured software programs are better in that they are more understandable and easier to implement with structured programming languages. The program is relatively large and complicated. It has two main modules, the sequencer module contains eleven submodules and the ALU module contains fifteen submodules. There is a main utilities module which contains fourteen modules. The two main modules and a housekeeping module frequently call on the procedures in the utilities module, which do most of the bit manipulation within each microword. Thus, the main features of the microcode generator are its use of interactive communication with the user, menu-driven information flow and complexity.

III. DESIGN APPROACH

Design may be defined as the process of applying various techniques and principles for the purpose of defining a device, a process, or a system in sufficient detail to permit its physical realization. [Ref. 7: p. 128] The field of software engineering has two general aspects. One aspect contains the general theories and principles proposed and tested over the last fifteen years, as the art of producing software became a science, i.e., a more disciplined approach was needed to combat the recognized "software crisis" of skyrocketing development costs of software beginning in the 1970s. For example, Edsger W. Dijkstra wrote, in 1968, a now famous letter called "Go-To Considered Harmful." [Ref. 8] He had discovered that the difficulty in understanding programs making heavy use of go-to statements was a result of the "conceptual gap" between the static structure of the program in spatial terms and the dynamic structure of the program in temporal terms. This is called the Structure Principle. [Ref. 9: p. 137] Another accepted concept was mentioned in chapter one; the Information Hiding Principle first proposed by D. L. Parnas in 1972. The second aspect of software engineering is the large group of methodologies or techniques available which are specific design tools and solutions for virtually any software engineering problem in existence. Thus, we have principles on one hand and practices on the other. In this chapter I will explain some software principles and accepted practices I have used

35-32:	ALU function selection.
31-16:	Control fields for the Am2904, Status & Shift Control chip:
31-30:	Controls carry-in.
29-24:	Called I5-I0. These six bits control the micro and macro status registers. This field is highly dependent on other fields, and is extremely difficult to coordinate when programming.
23:	Enables micro status register.
22:	Enables macro status register.
21:	Command enable to use 19-16 as a command field for special functions of the ALU.
20:	Shift enable to use 19-16 as shift control.
19-16:	Used as a field for ALU special functions, for shift control, or for status enable.
15:	Monitor breakpoint bit.
14:	Not used.
13-0:	Control fields for Am2910 Sequencer:
13-4:	Branch address; or in conjunction with bits 47-45 could be used for data constants or to specify ALU registers as sources of data or addresses.
3-0:	Sequencer instructions.

Following chapters will go into greater detail when addressing the design approach to implementing the functions of the different fields in the microword.

The Am29203 ALU Evaluation Board is set up with a monitor which can display and load all memories and registers. To run microprograms, load the microwords in hex into WCS; the status registers and macroinstruction registers should be loaded with appropriate data if conditional testing is to be performed. Enter the command G (for Go). After execution the registers and/or memory can be inspected (displayed) for indications of the intended results. The execution of a microroutine can be halted or paused and breakpoints can be set. The board is manipulated via the monitor much like a debug utility program.

4. Microword Format

The microword used to control the evaluation board is 48 bits wide. It incorporates the techniques of bit steering; using a bit or field of bits to determine the meaning of another field, [Ref. 5: p. 172] and vertical programming in some fields. The microword is organized into three main fields which control the three main IC's mentioned above. A discussion of the function of each bit of the microword follows, starting with the most significant bits: [Ref. 6: pp. 3.10-3.12]

47-45:	Controls the selection of the register address fields which select the microinstruction pipeline register (the output microword from WCS) or the macroinstruction register as either the sources for the ALU operands or the destination of the ALU operation result.
44-32:	Control fields for the ALU:
44:	Enables ALU output to ALU registers.
43:	Enables ALU output to the Y bus (the main data bus).
42-40:	ALU source operand selection.
39-36:	ALU destination selection.

allows pipelining, the overlapping of microinstruction execution with control store fetches. While the microword in the pipeline register is being executed, the next instruction is being fetched from WCS. All these improvements over the simple three element control unit provide the needed capabilities for an efficient microprogrammed control unit.

3. Am29203 Evaluation Board Description

An Am29203 Evaluation Board prototype is used for microprogramming experimentation in the NPS Computer Science Department. The evaluation board is built, using bit slice architecture, from various bipolar integrated circuits (IC's) produced by Advanced Micro Devices of Sunnyvale, CA. The Am29203 is an implementation of an Arithmetic/Logic Unit (ALU). This board is presently used in research at NPS as a representative microprogrammable digital system. As the target microprogrammable device for the microcode generator, the evaluation board is briefly described here, and in more detail in later chapters.

The board consists of three main IC types and memory for Writable Control Store (WCS). [Ref. 6] It uses an Am29203 four bit ALU slice. This ALU chip can perform seven arithmetic, nine logical and sixteen special functions on two four bit operands. The evaluation board cascades four Am29203's to provide a sixteen bit ALU. The board also uses the Am2904 Status and Shift Control Unit, which supports the functions of the ALU. The third main IC is the Am2910 Sequencer used in the board's control unit. It is the microprogram controller for the sequence of execution of microinstructions stored in the WCS. It provides both sequential access and conditional branching to addresses in the WCS.

important in computer system research and real-time, embedded computer systems (military applications) since updates and/or configuration changes are more easily handled with the flexibility of microprograms which can be over-written.

The simplest implementation of a microprogrammed control unit consists of a register or buffer, timing signals and a ROM (Read Only Memory). The register contains the macroinstruction which holds the op code to be implemented with a microroutine. The op code is used to derive the starting address in the ROM, called control store, of the appropriate microroutine to be executed. Each microword contains a code indicating that either it's the last microword in the routine or that the next sequential microword in control store is to be used next. This scheme causes fragmentation or unused portions in control store since each starting address is equidistant from the others and thus, all microroutines are alloted the same amount of space even though all routines are not the same size.

A mapping PROM (Programmable ROM) can be used to improve the addressing scheme. The fragmentation problem is solved since any set of addresses in the control store can be placed in the mapping PROM so the microroutines can be any size. With the use of a Writable Control Store (WCS) microroutines can be easily changed and new starting addresses updated in the mapping RAM (Random Access Memory). A MUX (multiplexor) for conditional codes can be added to include conditional branching capability.

Another important improvement has been the addition of a buffer register at the output of control store; the pipeline register. Its use are critical. There must be sufficient time available per macroinstruction for the microroutine instructions to be completed. The diode control store method posed no speed problems since the speed ratio was about twenty internal cycles per memory cycle. [Ref. 4: p.498] The trend of smaller cycle ratios developed quickly, due to improvements in memory technology. By the early 1960's main memory cycle speed had dropped to under one microsecond. [Ref. 4: p. 499] The ratio of internal machine cycles to memory cycles became one or two to one. It wasn't possible to decrease the control store access speed so parallelism was needed in data transfers. Multiple data transfers per machine cycle resulted in simultaneous control of internal resources. The microwords (microinstructions) were made wider to produce more control signals per machine cycle. Parallelism was increased with the addition of more control signals per microword. However, the use of wider microwords required more space in control storage.

Emulation, the use of control stored microprograms to interpret several different processors' instruction sets on one host system, [Ref. 5: 405] was implemented by IBM on the System/360 in the mid 1960's. This new application for microprogramming was very important for businesses which did not want to have to reprogram old software but did want to use new programming languages and develop new applications on newer higher performance machines.

Fast read/write control store was developed in 1970 using bipolar monolithic technology. [Ref. 4: p. 499] Thus, the control store has the same access time as combinatorial-logic gating delays, since they are made of the same material. Writable control storage can be very

Wilkes viewed the Control Store as consisting of two ferrite core matrices. [Ref. 4: p. 497] (See fig. 2.1) A portion of the macro-instruction called the operation code or op code was used as input to a decoding logic tree. The logic component accepted n bits as input and provided 2^n possible output lines, only one of which was selected. Thus, a four bit input line could select one of sixteen output lines which were the microinstructions. Each output line was configured with diodes to select any number of available control lines. The active current in the selected output line was passed on by a diode to the connected control line in the first matrix. So the configuration of diodes on the output line determined which control lines were activated for a particular clock cycle and a particular microword. When testing a design, these diodes were easier to change than rewiring a new hard-ware circuit for a particular function. The 2^n possible diode configured output lines represented primitive operations (the microinstructions) which when selected in sequences formed short subroutines which carried out the function specified by the op code. The second matrix was used for sequence control. Each microinstruction could select the next microinstruction to be executed. Wilkes' implementation also included a provision for some conditional testing and sequencing. These diode arrays were the first microprogram memory or Control Store.

2. Development

While macroinstructions generally effect changes to data in main memory, microinstructions reflect register to register data transitions. Cycle time ratios between main memory access and control store access

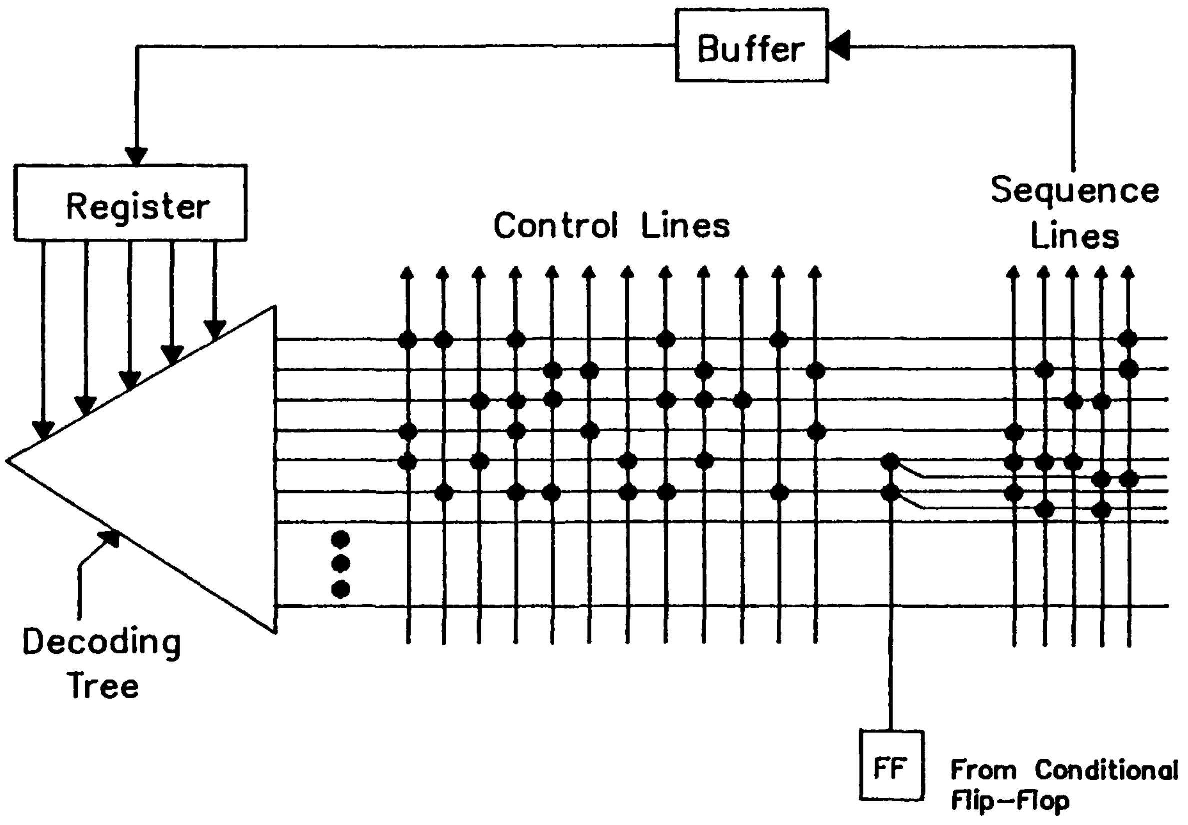


Fig. 2.1 Wilkes Control Unit

The disadvantages of hardware implementation are that any changes could mean an entire redesign; documentation was scarce; and it was difficult to test the implementation since much of it had to be working just to test one small portion. The advantages are that it can be the fastest running implementation, a small task (simple design) will have a simpler solution and the complexity and lengthy design time can be justified for high volume applications. In a microprogrammed machine each machine-level or macroinstruction is carried out one instruction at a time by an interpreter.

M. V. Wilkes, in 1951, first proposed a microprogrammed control unit in a computer. [Ref. 3: pp. 16-18] The microprogrammed method was easier to use in the computer design development and engineering phases, Wilkes proposed. He and his colleagues sought a means for rearranging the circuit design into a systematic order which was easy to implement, comprehend and maintain. They were more interested in simplifying the design task than in any savings of hardware. It is interesting to note that while Wilkes didn't believe there was any need for general purpose computers or the corresponding complex instruction set, the microprogrammed control scneme he presented made the concept of a general purpose computer feasible. The general purpose computers in use today have instruction set sizes on the order of hundreds of instructions. The design and implementation of a system of combinatorial logic with such a complex control system would be very expensive to manufacture.

called Control Store. These sequences of primitive microinstructions are called microprograms; stored programs that explicitly control the data-flow through the physical components of a processor. This method is an alternative to performing data-flow control with a network of sequential logic circuits. [Ref. 2: p. 5]

1. History

The early computers' instructions for arithmetic and boolean functions were directly implemented with hardware. One could look inside the computer and trace the circuits responsible for a particular function such as multiplication. These basic logic circuits such as AND, OR, etc. were constructed from switching devices such as vacuum tubes, diodes and transistors. The outputs of the resulting storage elements, i.e., flip flops and latches, control the execution of arithmetic or logical operations by issuing control pulses over the control lines to specific gates in the data flow. In a relatively few machine cycles, information is guided to flow over many paths and through many functional units in the specific order required for the execution of the macroinstruction. Hardware implemented control units require many basic components and intricate wiring efforts in their design. They have a haphazard appearance due to the mass of wires and circuits placed on the circuit board in any place they could be fit. Complex instruction sets were difficult to implement because the design task alone was a tremendous undertaking. It was a lack of modularity in the design process which forced the designers to maintain a complete understanding of the entire design in their heads.

Another important interface consideration is the input device(s). Thus far no other input devices are available for use with the VAX development system used in this project. However, an excellent input device for selection is the mouse. Its most common criticism is that it takes the user's hands from the keyboard, but this is important only to word processing applications where speed typing is advantageous. The microcode generator's user will have his attention focused on the screen which is perfect for using a mouse. When the code generator is adapted for use on a microcomputer workstation a mouse could be implemented as the input device for menu selection. Using a mouse also creates muscle memory which improves the user's productivity. Other input devices include joysticks, speech recognizers, tablets, etc. Speech synthesis and recognition quality is improving quickly but the mouse has the characteristics of simplicity, flexibility and cost effectiveness which make it a very popular device for use with interactive programs.

C. METHODOLOGY USED

Once software engineering concepts are understood, actual implementation of a project is the next step. The explanations of what is going to be done, using what guidelines, who is going to do it and why it's important, have all been presented. The next step is how to implement the microcode generator. Every company and government agency responsible for producing software has some formal or informal development methodology, a collection of methods, chosen to complement one another, along with rules for applying them. [Ref. 12: p. 14] The methods chosen to design and implement the code generator will be discussed in the following sections.

1. Specification Phase

The basic requirements of the specification phase of a software development project are to describe the intended data flow and data structure to be realized in the program, provide a description of the program functions, and establish and maintain communication with the user. [Ref. 7: p. 95]

The internal structure of the C source program is modular in that each logical microprogramming function, selecting the branch address to be used in the sequencer portion of the microword for example, is implemented as a separate procedure. (Procedure is a generic term for what is called a function in C.) Also the "main" procedure does little but initialize, call the modules selectable from the master menu and help the user exit the program.

This modularity is necessary when a top-down design technique is used as in this program. Top-down or stepwise refinement begins with a high-level representation of software procedure. First, the "main" program or driver program is defined; then each procedure called by the driver program is written in code and so on. [Ref. 7: p. 131] With this technique, the program can be written and tested in more manageable pieces. This is a far better method of design than writing the entire program before trying to run it. The most important reason is that the user can be involved to provide early guidance from his, the most important, point of view.

The external structure is the dynamic flow of the program from the user's point of view. The motivation for writing the microcode generator is to make microprogramming easier and less time consuming. An example of a microcode generator in use is AMDASM™ which is available on Advanced Micro Devices System/29™. [Ref. 13: p. 10] This tool is not interactive and is quite complicated although it is quite general. The interactive nature of this project's code generator makes the program more "user-friendly." The menus help the user concentrate on the problem to be solved rather than how to run the program. Thus, the flow of the program is directed by the hierarchy of menus.

The descriptions of the functions implemented in the program must be specified in this phase. Any changes to these specifications in a later phase will mean a slow-down in the development of the software, since most work must stop while the implementer backtracks to effect the changes. A general description of the program's functions follow.

a. Sequencer Functions

Once the user selects a sequencer code, the sequencer module of the microcode generator determines what support data is needed. If the selected code requires a branch address and/or condition codes, it requests the data using further menus. Then it provides informational messages if the sequencer code selected is dependent on other sequencer codes which must precede or follow it. Provisions must be made for "remembering" the user's selections to ensure that subsequent changes to the sequencer code in the same microword remain consistent with other fields of the microword.

b. Support Functions for the Sequencer

Shift and condition code fields implemented as a result of selected sequencer code functions are incorporated into the sequencer module of the program. Conflicts between shared function fields are to be resolved or gracefully handled. That is error messages must be provided which don't "crash" the program or cause undetected erroneous results. A requested function may have up to seven possible bit patterns. C language structures which hold constant data for comparison of microword bit pattern options are used to resolve conflicts in shared function fields. If none of the possible patterns are compatible in the same microword fields then the user is informed and the requested function is denied.

c. ALU Functions

Once the ALU function code is selected, the ALU module of the program determines which type of ALU function it is, either a basic function or a special function. Following the same guidelines as provided for sequencer functions, it determines what follow-on data is needed to microcode the ALU fields of the microword. This is done by presenting appropriate menus based on user choices already selected. Each menu may lead to several additional menus, depending on the selection made. After the ALU function is selected the user is prompted for operand source data, results destination data with choices for shift register manipulations on the results, a decision to enable the Y bus, and source register selections if any. The same data structures and utilities that are applied in the sequencer module are used.

d. Support Functions for the ALU

The microcode for the shift fields, command fields and register selection fields associated with selected ALU functions are incorporated into the ALU module of the program. The same data structures, utilities and constraints concerning conflicts and errors are used as in the sequencer module.

e. Common Functions

Each menu has a selection available for help and return to a higher level. When help is selected, an informational message is displayed to clarify the menu presented and a reference to a manual or data book is provided if possible. Selecting return to a higher level displays the previous menu. The user can exit the program entirely by repeating the return selection.

f. Housekeeping Functions

These functions comprise the third major module of the program. They provide the capability for the user to build microroutines by adding microwords to a file as he creates them using the sequencer and ALU modules. The user may also save, list, scan, modify, delete and print microroutines. These latter functions will be available from the master menu and some of them will be available from the sequencer and ALU modules.

2. Design Phase

The design phase is the process through which requirements, as determined in the specification phase, are translated into a representation of software. [Ref. 7: p. 129] The goal in this phase is to produce a "model" from which the final product will be built. This step in the development of a software project can be compared to the building of a prototype in an engineering project.

The "models" used to represent the program are hierarchical module organization charts. In figure 3.1 the hierarchical structure between the major modules is shown. Figure 3.2 shows the relationship between the menu calling procedures. This model reflects the relationships between the menus and shows the program structure from the user's point of view. These two models are adjusted until the internal program structure (figure 3.1) supports the external structure (figure 3.2) which is the user's point of view of the program.

3. Implementation Phase

In very general terms, the implementation or coding phase translates a design representation of software into a programming language realization. This coding process begins when the programmer puts source code on paper and continues until an executable form is produced by the computer. Improper interpretation of design models is a primary concern in this phase. [Ref. 7: p. 267] The characteristics of the programming language used influence the way the programmer thinks when implementing the design. Earlier in this chapter I discussed the reasons why a modularly structured design should be implemented using a structured language. The ease of design-to-code translation is an indication of how well the language mirrors the design representation. C's support of structured programming and rich set of operators make the design-to-code translation very smooth.

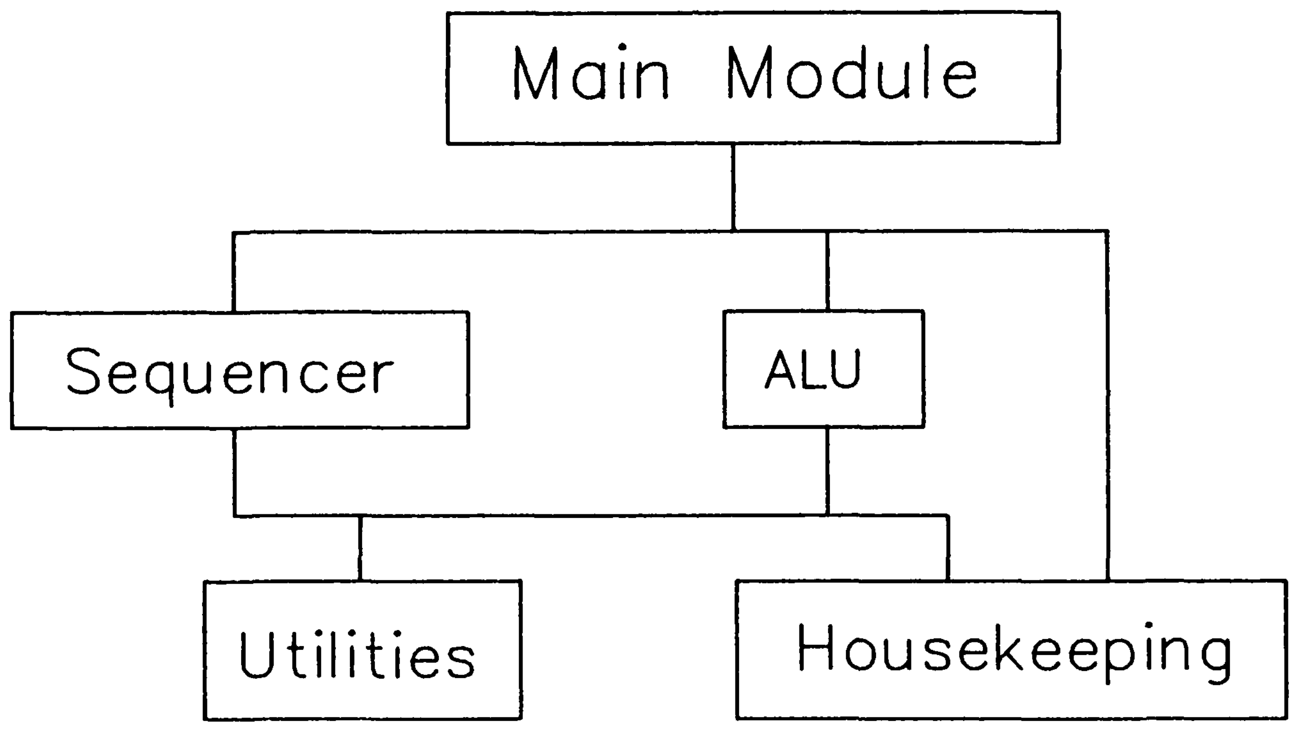


Fig. 3.1 Hierarchical Structure of Main Modules

Main Module

- Sequencer Command Menu
- ALU Basic Function Menu
- ALU Special Function Menu
- Routine Manipulation Menu

Sequencer Module

- Sequencer Branch Address Menu
- Sequencer Condition Select Menu
  - Am2904 Conditional Test Menu

ALU Module

- Called from Main Module:
  - ALU Basic and Special Function Menus

- ALU Operand Source Menus
- ALU Result Destination Menu
- ALU Register Address Menu
- ALU Direct Source Menu
- ALU Instruction and Output Enable Menu
- ALU RAM A Register Select Menu
- ALU RAM B Register Select Menu

Fig. 3.2 User's View of Menu System

4. Test and Evaluation Phase

Testing within the context of software engineering is actually a series of four steps that are implemented sequentially. [Ref. 7: p. 295] *Unit testing* is a test of each procedure as it is produced. *Integration testing* addresses the issues associated with the problems of verification and assembly of all modules in the program. *Validation testing* provides assurance that the software meets all functional and performance requirements. *System testing* verifies that the program meshes with other systems in the us^r's environment.

The procedures of the microcode generator were tested as units and integrated with the program as they were coded. Thus, the first two steps were executed concurrently.

E. W. Dijkstra is quoted as saying, "Program testing can be used to show the presence of bugs but never their absence." Could exhaustive testing (even if possible) prove a program correct? No, because you don't know when all tests are exhausted. In testing the code generator, the top-down design, modular structure of the program made the task easier since functions were broken into small pieces as procedures. Each unit was tested by running it with both expected and unexpected data.

5. Maintenance Phase

Software maintenance is far more than just fixing errors in a program; it consists of all support for the product once it released. There may be several versions of the same program which need support. The maintenance of existing software can account for over 60 percent of all effort expended by a software development company. [Ref. 7: p. 322]

Software maintenance may be defined by describing four activities that are undertaken after a program is released. *Corrective maintenance* includes diagnosis and correction of any errors which may exist after the program is released. *Adaptive maintenance* is the activity that modifies software to properly interface with changing system support. *Perfective maintenance* provides the software package with new capabilities, modification of existing functions and general enhancements requested by users of the product. *Preventive maintenance* is done to improve future maintainability or reliability. This type of maintenance is still relatively rare. [Ref. 7: p. 323]

Academic research projects don't have the same requirements for maintenance as a new product in the commercial market. The maintenance of the program generator will depend on the availability of interested students for further research. However, further work is needed in this project. This thesis is just a small step in the development of a fully generalized functional microcode generator.

D. SUMMARY

Software engineering is still more of an art than a science, even though the application of systematic methodologies began about a decade ago when software developers faced the software crises. But, the use of a more disciplined, engineering approach to software design has helped developers manage more efficiently the large, complex type of problem solutions undertaken in the 1980's.

The microcode generator implementation is a large and complex problem for one person. Thus, the "modularizing" of the process, i.e., breaking the project up into the distinct steps of specification, design, implementation, test and evaluation, and maintenance helped me to conceptualize the entire process. In shipboard administration terms, I formed a "Plan of Action and Milestones," a POA&M, and carried it out.

The next chapter will address specific data structure implementations, coding problems and introduce a preliminary user's manual.

IV. IMPLEMENTATION

The planned sequence of implementation was discussed in the preceding chapter. In this chapter I will address specific decisions concerning required data structures and the general composition of the source program for the microcode generator. This discussion will include the general utilities, the housekeeping utilities and the header files. A section on how to run the program is provided as a guide to using the microcode generator. The last section of this chapter addresses some ALU implementation difficulties.

A. MENUS

Interactive programs can be designed to interface with the user in two ways; command-driven or menu-driven. The command-driven method requires the user to know what commands are available, and the rules and syntax necessary to use them. The menu-driven method is more "user-friendly" since the user is allowed to pick from a menu of available functions; he doesn't need to memorize a great deal of detail. The tradeoffs between the two methods are speed and flexibility in the first case and ease of use in the latter case. The main decision point when choosing between the methods is at the point where a program becomes too complicated for even experienced users to remember all the commands needed to effectively use the program. The menu-driven method was chosen for this reason, i.e., there are too many possible functions available to be remembered in the command-driven method.

Current macro-assembler type microprogram generators use batch style execution, are thus command-driven programs and are less "user-friendly."

To estimate the number of commands that would be required in a command-driven program is not difficult once the menu-driven version exists. Each option on a menu represents a required command or subcommand in a command-driven version. The sequencer portion of the microcode generator program has 6 menus including a total of 54 options to choose. The ALU portion has 20 menus including a total of 214 options to choose. This totals 268 commands. But, each menu in both portions has a "Help" and "Return to Higher Level" option; so these can be combined to function under only two commands for all menus. So, we have a total of 268 - 2*(20 + 6) + 2 = 218 commands which would be required in a command-driven method. The interdependencies of some of the microword's bit fields would further complicate a command-driven structure, and these are only the functional microprogramming commands. The housekeeping functions like saving, listing, scanning and printing micro-routines also need commands. Since the average number of options per menu is 10, that number should be added to account for the housekeeping module. So, the final total of needed commands in the command-driven method is at least 228, not counting the effect of field dependencies. This is not a reasonable number of commands to expect a user to remember in a task like microprogramming.

The microcode generator program was written using the following general concept of program flow: Write a menu to the screen, and then trap and test the user's response using the C language SWITCH construction (the equivalent of the Pascal CASE construct). [Ref. 10: p. 54-56] The SWITCH case which matches the user selected menu option implements the function requested, perhaps displaying further menus from which the user selects the proper parameters for the function desired. This menu, SWITCH, implement, menu, SWITCH, implement ... continues until the user selects the "Return to System" function. The program is roughly half menus and half SWITCH constructs.

B. DATA STRUCTURES

1. The Microword

The forty-eight bit microword for the Am29203 Evaluation Board was designed to provide a representation of the functions for a general purpose sixteen bit ALU.[1] The microword, as designed, uses the concepts of both *horizontal* and *vertical microprogramming*. [Ref. 4: p. 501] In a horizontal microword there are many bits for the control lines, providing a parallelism in resource handling. This method is costly in terms of memory space but is very fast since there is only one level of control. A vertical microword has fewer bits and needs further decoding to determine which control lines are affected. It is a memory space efficient method but usually executes slower than the horizontal word because it has several levels which are decoded to determine the control lines activated.

The specific fields in the microword are described in chapter two. Several of the Am29203 Evaluation Board functions share bit fields

[1] The Am29203 Evaluation Board was described in chapters one and two.

sources for ALU operations, the destinations for the ALU results, and the selection of registers for the ALU.

F. ALU IMPLEMENTATION DIFFICULTIES

The ALU module presents some difficult implementation problems since it involves more potential conflicts in the microword bit fields than does the sequencer module. The ALU functions which involve shifting and carry-in bits pose particular problems since they share bits with the conditional testing fields and the command field. Another hot spot in the microword is the sharing of the branch address fields and the register A and register B fields.

Compatibility is possible, in some cases, between functions which share fields since many of the shifting functions are satisfied with several alternate bit patterns (some have up to seven possible patterns). However, the program must be "smart" enough to determine when the conflicts occur and warn the user. This problem is the most difficult to solve in the implementation of the microcode generator. To optimize microprograms the programmer has to take every opportunity to code as many functions in each microword as possible. If he codes only one function per microword there would not be enough room in Control Store for all his routines. This task is the most complex and time consuming part of microprogramming. So, although the ALU implementation is not complete, the solution, in the form of a sample algorithm, is provided for the problem of automating the process of optimizing microroutines (Appendix C). This algorithm searches through C language STRUCTURES [Ref. 10: p. 119-141] that are set-up to store all the possible bit

```
                    ALU SPECIAL FUNCTION MENU

Enter the value corresponding to the function you wish to
perform:

        0  Unsigned multiply
        1  BCD to Binary Conversion
        M  Multiprecision BCD to Binary Conversion
        2  Two's Complement Multiply
        3  Decrement by 1 or 2
        4  Increment by 1 or 2
        5  Sign/Magnitude to Two's Complement Conversion
        6  Two's Complement Multiply
        7  BCD Divide by 2
        8  Single Length Normalize
        9  Binary to BCD Conversion
        Z  Multiprecision Binary to BCD Conversion
        A  Double Length Normalize; First Division
        B  BCD  Add
        C  Two's Complement Divide
        D  BCD Subtract  F = R - S - 1 + Carry In BCD
        E  Two's Complement Divide Correction and Remainder
        F  BCD Subtract  F = S - R - 1 + Carry In BCD
        H  for HELP with this menu
        R  to RETURN to higher level
```

Fig. 4.3 ALU Special Function Menu

```
                ALU BASIC FUNCTION MENU

Enter the value corresponding to the function you wish to
perform:

      0  F = High
      1  F = S - R - 1 + Carry In
      2  F = R - S - 1 + Carry In
      3  R + S + Carry In
      4  S + Carry In
      5  (NOT S) + Carry In
      6  R + Carry In
      7  F = (NOT R) + Carry In
      8  F = Low
      9  F = (NOT R) AND S
      A  F = R EXCLUSIVE OR S
      B  F = R EXCLUSIVE OR S
      C  F = R AND S
      D  F = R NOR S
      E  F = R NAND S
      F  F = R OR S
      H  for HELP with this program
      R  to RETURN to higher level
```

Fig 4.2 ALU Basic Function Menu

POP, and the TWB commands all require both branch addresses and conditional tests. The two sets of menus are displayed and the user is prompted for all selections as before.

```
                   AM2910 SEQUENCER COMMAND MENU

 Which AM2910 Sequencer Command do you wish to chose?

  Enter a  0  JUMP ZERO - JZ
           1  CONDITIONAL JUMP SUBROUTINE - CJS
           2  JUMP MAP - JMAP
           3  CONDITIONAL JUMP PIPELINE - CJP
           4  PUSH/CONDITIONAL LOAD REGISTER/PIPELINE - PUSH
           5  COND. JUMP SUB. VIA REG OR PIPELINE - JSRP
           6  CONDITIONAL JUMP VECTOR - CJV
           7  CONDITIONAL JUMP VIA REGISTER OR PIPELINE
           8  REPEAT LOOP, COUNTER NOT EQUAL 0 -  RFCT
           9  REPEAT PIPELINE, COUNTER NOT EQUAL 0 - RPCT
           A  CONDITIONAL RETURN FROM SUBROUTINE
           B  CONDITIONAL JUMP PIPELINE AND POP
           C  LOAD COUNTER AND CONTINUE - LDCT
           D  TEST FOR END OF LOOP - LOOP
           E  CONTINUE - CONT
           F  THREE WAY BRANCH - TWB
           H  HELP with this program
           R  RETURN to higher level
```

Fig. 4.1 Sequencer Command Menu

3. ALU

The ALU module of the microcode generator, although not fully implemented, will be used in the same fashion as the sequencer portion. The user selects ALU Basic Functions (fig. 4.2) or ALU Special Functions (fig. 4.3) from the main menu. The user is then prompted through several different levels of menus. He provides input such as the

1. Main Menu

From the user's point of view, this program is very easy to use. After executing the program, the sequencer module, the ALU module, or housekeeping functions like scan or print microroutines can be selected. Also, the Help and "Return to System/Higher Level" options are available as in nearly every menu in the program. Selecting Help in each menu will cause an informational message to be written to the screen, then the same menu is presented again. The "Return" function causes the user to go back to the previous menu.

2. Sequencer

In the sequencer portion of the program, the Sequencer Command Menu (fig. 4.1) is presented. This menu displays the sequencer codes used by the Am2910 Sequencer chips on the Evaluation Board. As discussed above, the sequencer codes are grouped in four classes. The JZ, JMAP, RPCT and CONT codes do not use branches or conditional testing, so only the sequencer code bits are set in the microword. The LDCT command requires an entry in the branch address field so the Branch Address Menu is then displayed and the user enters the desired values for the address. The maximum address that can be coded is 3FF hex. If a higher value is entered the user will see an error message and the menu is presented again. The CJV, Conditional Return from Subroutine, and LOOP commands require that conditional tests be set up. A menu called the Condition Select Menu is displayed and prompts the user through several levels of menus until the desired conditional test is set. The CJS, CJP, PUSH, JSRP, Conditional Jump Via Register or Pipeline, Conditional Jump Pipeline and lookup table. For example, the user selects choice '8' from a menu which is to be set in a four bit field; the procedure hex_field is called. Hex_field is passed the starting bit and calls the procedures bit_set and bit_clear to set the bit pattern 1000.

The housekeeping utilities will manage such functions as list, save routines, scan, modify and print microroutines. These utilities can be called from both the ALU and Sequencer modules.

D. HEADER FILES

The use of header files in C language programs increases flexibility and organization in the management of program segments. The microcode generator program uses two header files to centralize the definition of constants and externals. Declare.h contains all the constants defined using the #define preprocessor feature. Extern.h contains all the variables and pointers externally defined. The Declare.h file is "#included" with each module. The Extern.h file is included with all but the main module.

E. RUNNING THE CODE GENERATOR

As stated before, this tool is intended for the experienced micro-programmer. The microprogramming techniques and structures for the Am29203 Evaluation Board are contained in Reference 6, the Am29203 Evaluation Board User's Guide, and Reference 14, AMD's Data Book. The program's help messages, where implemented, are derived from these two references.

```
                    AM2910 SEQUENCER BRANCH ADDRESS MENU

You have chosen a command which requires a value in the
register/counter

     ENTER YOUR BRANCH ADDRESS FIELD
           H  for HELP with this program
           R  to RETURN to a higher level
222
This is the address being used. 222

     Docu-word:
        0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1 0 0 0 0 4

     Microword:
        XXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXX  XX10001000100111
              ffff              ffff              e227


          AM2910 SEQUENCER CONDITION SELECT MENU

You have chosen an AM2910 Sequencer Command which requires a
     conditional test

What do you want to do next?

     Type a  P  for FORCED PASS - unconditional
             F  for FORCED FAIL
             T  to TEST the condition
             H  for HELP with this program
             R  to RETURN to higher level
f

     Docu-word:
        0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 -1 0 0 0 0 4

     Microword:
        XXXXXXXXXXXXXXXX  XXXXXXXXXX0X1000  XX10001000100111
              ffff              ffd8              e227
```

C. UTILITIES

The Utilities module contains many support procedures. The procedures binary_field, dual_field, octal_field and hex_field convert a selected function to the needed bit pattern in respective binary, dual, octal and hex sized fields. This is done using a SWITCH statement as a the paragraph above) depending on the value in the present docu-field. The docu-field is reset with the code corresponding to the class of conditional test newly requested, and the microword is set. The resulting microword for the "conditional jump via register/pipeline" with requested branch address and conditional test is displayed, and the main menu is again written to the screen. The following is an edited version of the sequence of events as seen on the monitor screen when the above example is run using the microcode generator. Appendix A is an unedited record of a sample session using the code generator.

```
                     MASTER AM2910 SEQUENCER MENU

      What do you want to do next?
           Enter a  0  to select SEQUENCER COMMAND
                    H  for HELP with this program
                    R  to RETURN to system
0
                    AM2910 SEQUENCER COMMAND MENU

      Which AM2910 Sequencer Command do you wish to Chose?

      Enter a  0  JUMP ZERO - JZ
               1  CONDITIONAL JUMP SUBROUTINE - CJS
               2  JUMP MAP - JMAP
               .

               7  CONDITIONAL JUMP VIA REGISTER OR PIPELINE
               .

               F  THREE WAY BRANCH - TWB
               H  HELP with this program
               R  RETURN to higher level
7

      Docu-word:
         0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4

      Microword:
         XXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXX  XXXXXXXXXXXX0111
               ffff              ffff              fff7
```

address function needs no subdivision); and choice = "the pointer to the first digit of the branch address entered." The branch address' corresponding docu-field status is checked; if it is not zero, the user is again prompted to confirm the change. If the branch address docu-field is zero, or confirmation to change the branch address is received, then the address requested is placed (in binary) in the microword.

The branch address case (number 19) of Field_set also checks the status of the docu-field function for register A and B selection. This is done because the two functions; selecting the branch address and selecting the A and B registers, share the bits from 36 to 43. The compatibility algorithm is used to find a compatible bit pattern if possible.

After the branch address is selected the menu for selecting the desired conditional test is written to the screen. The user's options are to select a forced pass, forced fail or to go to another menu to select the exact test desired. Field_set is called with field_cnt = 25 for the conditional testing case of Field_set's SWITCH; sub_set = 1, 2, 3 or 4; choice = "pointer to the choice for the particular test desired." The sub_set code is assigned according to which fields need to be set for the conditional tests function. All the different conditional tests are set using four different groups of physical fields. The value 1 means that the Command Enable field (bit 26) is set; 2 means that the Command Enable field and the Command field (bits 28-31) are set; 3 and 4 mean that the fields at bit positions 26,28,18,20 are all set.

First the conditional tests function docu-field is checked; if it is not zero then the appropriate physical fields are erased (according to sequencer); sub_set = 4 (the sequencer command function divides the sequencer codes into four classes); and choice = '7'. A sub_set value of 1 would mean that only the sequencer logical field is coded. (The sequencer field is coded for all cases.) A 2 means that both the sequencer and branch address fields are coded. A 3 means that conditional tests are set. A 4 (the value in this example) means that all three functions; the sequencer field, the branch address field and conditional test fields are set.

The first step is to check the history of the function in the appropriate docu-word field. If the docu-field's value is zero, then it is set to 4 (passed from the sub_set parameter), this stores the information that a class 4 sequencer code is being set. Field_set then places the proper bit pattern for a choice '7', "conditional jump via register/pipeline", in the sequencer physical field; bits 44-47 are set to 0111.

However, if the docu-field for the sequencer function is already 1, 2, 3 or 4, then the sequencer code field has been set by a previous request. The user is then prompted to confirm that he desires to change the sequencer code. If so, then fields previously set are erased (set to 'XXX...'; these are indicated by the code in the docu-field), the new sequencer code is set, and the docu-field is set to 4.

After the sequencer code is set, the menu for selecting the branch address is written to the screen. The user is then prompted to enter the desired branch address. The branch address select procedure then calls Field_set with field_cnt = 19 (the SWITCH case in Field_set which manages coding the branch address); sub_set = -1 (the branch has been set previously, and a message is written to the screen for the user to confirm that he intends to change the function. If the user does not confirm, no changes are made to the microword or docu-word. If a change is confirmed then the docu-field and microword are both modified to represent the new choice. If the previous function set other physical fields then the old docu-field value will indicate which other physical fields need to be erased or reset.

When the requested function shares physical microword fields with other functions, the status of the docu-field's of those other functions must be checked also. If their docu-field values are zero then no conflict exists. However, if they have been set, then a potential conflict exists. If both functions have no alternate bit patterns, then a message is written to the screen informing the user that a conflict exists; no changes will be made. If one or both functions have alternate bit patterns then a compatibility checking algorithm provides a compatible bit pattern if one exists, or produces an error message. The compatibility algorithm is discussed later in this chapter.

For a fairly complex example illustrating the use of Docu and Field_set consider the selection of a sequencer control code. The user selects "sequencer command" from the main menu; then selects, for example, choice '7', the "conditional jump via register/pipeline" command. This is a sequencer command which requires a value be placed in the branch address field of the microword and provides for conditional testing as well. The main SWITCH statement in the command select procedure of the sequencer module calls the Field_set utility with parameters field_cnt = 24 (the Field_set SWITCH's case concerning the data structures since they establish, in the cases of SWITCH statements, the relationships and dependencies between the physical bit fields of the microword and the functional docu_fields. That is, they use the function requests and the "documentation" or history of requests for a particular microword as input and by following the logic contained in the SWITCH cases, produce the properly coded bit pattern for that microword.

4. Field_set Utility

The Field_set utility consists of a SWITCH statement with a case for each physical field. Field_set is called whenever a user selects a function to be microcoded. The parameters passed to Field_set are field_cnt, sub_set and choice. Field_cnt is the argument to Field_set's main SWITCH and indicates the case which refers to the appropriate physical field. Sub_set contains the integer code which is used in the docu-word to distinguish between classes of functions. Choice holds the character pointer indicating the menu option the user selected. In most menu functions, the "choice" is eventually converted by a SWITCH to the bit pattern used in the microword.

Each case in Field_set's SWITCH is a small procedure in itself. In general, when a case is selected it first checks the status of the corresponding docu-field. As discussed in the previous section, the corresponding docu-field value will be zero if it isn't set and a -1 or positive integer if it is set. When the appropriate case checks the docu-field status, if the value is zero, then the proper coded bit pattern is placed in the microword and the docu-field is set to -1 or a positive integer. If, however, the docu-field value is not zero, then it

2. The Docu-word

The Docu-word is a 24 element, integer array data structure used to "remember" the function choices requested by the user. This "documentation" feature of saving all function choices made by the user is necessary to provide such user-friendly features as warning the user when he has requested functions which produce conflicting microcode and allowing the user to change previously created microwords. A docu-word is created as each microword is built. Each element of the array corresponds to a "docu-field" which represents a function available in the microinstruction.

Each Evaluation Board function is represented as an element of the docu-word. The entire docu-word is initialized with zeros, indicating that no functions are requested. A docu-field assigned the value -1 means the function it represents has been requested by the user and the particular choice can be obtained from reading the appropriate bit field in the microword. This code is used when there is no overlapping or multiple function. A positive integer in a docu-field means the corresponding function has been requested and the value of the integer indicates a particular sub-function choice.

3. Docu Utility

The Docu utility consists of a SWITCH construct using the selected function's docu-field as the case value. Each case assigns the proper code, as described above, to the selected docu-field to store the user's function requests. Docu is called from the utility Field_set; both procedures are in the utilities module. Docu and Field_set, (Field_set will be described in the next section) although used as utilities, are also in the microword. The register select fields and branch address fields share microword bits 36-43.[2] The command and shift fields share bits 28-31. Steering bits 26 and 27 control the enabling of the command and shift fields respectively. That is, when bit 26, the command enable field is turned on, the command field is enabled in bits 28-31. When bit 27, the shift enable field, is on the shift field is enabled in bits 28-31. If both the steering bits are on then the field is shared in time as well as spatially. The microword won't make sense unless a compatible bit pattern between the shift and command functions can be found. When microprogramming by hand, this is the problem the programmer must solve in order to optimize his program. The ALU module of the design tool implements a procedure which checks for compatible bit patterns when potentially conflicting functions are requested. If no compatible pattern can be found, the two functions must be coded in two separate microwords. Further detail on the specific interrelationships of the microword functions can be found in Reference 6.

The microword, itself, is implemented in a character array data structure. It is initialized at the start of the program with X's which represent unassigned or "don't cares"; if any remain after the process, they are assigned a one automatically. The microword is displayed, at appropriate points in the user's session, bit-by-bit and as twelve hexadecimal values, using the utilities Display_word and Display_in_hex.

[2] Chapter two uses the conventional bit numbering method of right to left (47-0). However, in this chapter the microword bits are numbered 0-47 from left to right. This conforms to the element numbers in the microword array. This method is used since the programs' source code is written this way, and the reader can more easily refer to the listings provided in Appendix B.

patterns. It compares the requested function's bit pattern against the STRUCTURE, and then chooses a compatible pattern. Appendix C contains the source code of the test program which demonstrates this. This algorithm needs to be incorporated into Field_set's cases which incorporate the ALU functions. When there is a conflict, Field_set needs to read the microword's "history" by checking the docu-word and then make the compatibility check. If a compatible pattern is found, the microword is set, the docu-word is coded to reflect the new function added to the word, plus an indication is needed of the possible conflict. This is necessary in case the user wants to further modify the microword.

G. SUMMARY

The microcode generator's program modules and function implementations have been described detailing the data structures used and the support utilities and files provided. To show how easy the program is to use a guide to running the program was provided. The next chapter will discuss some conclusions and recommendations derived from this project.

V. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

The microcode generator is a needed tool for the microprogrammer and/or computer designer. The coding of microwords at the level of the machine's hardware involves detailed manipulation of the micro-programmed control unit's control lines, registers and functional units. The microword fields' complexity is directly correlated to the number of parallel functions that the microword can invoke. As detailed in chapter four, there are several shared bit fields in the microword used in the microcode generator. The program's ability to manage function conflicts in a "user-friendly" manner relieves the user of the "overhead" of tracking the potential conflicts as he designs his routines. The solution of this function conflict problem is demonstrated in the sample program provided in Appendix C. This compatibility algorithm compares the bit pattern of the requested function to a C STRUCTURE holding the possible bit patterns of a conflicting, previously selected function. If it exists, a compatible bit pattern is found and placed in the proper position in the microword. The microcode generator also prevents the user from making simple mistakes such as writing a '1' instead of a '0'. The program "writes" the proper digit once the user chooses the function desired.

The choice of C for the programming language in which to write the code generator was perfect for the application. The VAX 780/Unix system was predetermined in that it was convenient, accessible and the most powerful system in the department. The C language is an integral part of that system. C is difficult to learn because it is intended as a production language, not an educational tool. It's error messages are not very specific (eg. BUS ERROR - CORE DUMPED). It is, being a systems language, very flexible. There are few constructs to learn since everything is done with functions. The standard C function library provides all I/O functions since there are none in the language itself. The housekeeping functions which require using system calls to open/close files should be easier to implement in C. The Unix operating system is mostly written in C, so the two environments, C and Unix, are highly compatible.

The microcode generator design is approximately 75 percent operational. The Sequencer portion, most utilities and an elementary version of the ALU is completed. Initial testing on the Sequencer is complete. The program needs to be used in a design environment to find further bugs in either concept or implementation. Algorithms have been tested successfully which solve the shared field problems in the ALU. The function compatibility solution test program is shown in Appendix C.

I am satisfied that the decision to use the menu-driven method was the best way to implement the code generator. As discussed in the implementation chapter, the equivalent number of commands needed would be too great to use the command-driven method. The menus do become very familiar after prolonged use of the program. The slight impatience felt is a small price for the program's simplicity of execution.

B. WHAT'S NEXT

The obvious next step is to finish the implementation of the ALU and fill in the ALU portions of the Docu and Field_set utilities. The housekeeping functions also need implementation. As discussed previously, the compatibility test program will solve the function conflict problem. The housekeeping utilities can be completed by writing the routines to open files, and then using system calls to save and print the routines with user defined names.

A next step could be to adapt the program to run on a workstation used for computer design. Some research is needed to select an appropriate workstation. Some attributes should be: that it runs Unix (initially) and that it is readily accessible to students. A move to another Unix system would facilitate benchmarking between the VAX and the new system. Adapting the code generator to systems with different input devices is essential to studying the man/machine interface aspects of this project. As mentioned above, one drawback to the menu-driven method of interaction is that the experienced user of the program can become impatient as familiarity with the menus increases. The use of a mouse, for example, as an alternative input device might improve this situation. The mouse also presents the possibility of using more creative graphics to enhance the use of menus. For example, sensitive selection areas could be provided on the screen for the execution of frequently selected functions such as "display the microword", "erase a string of bits in the microword," etc. Individuals have different ideas of what the ideal method of communication with computers is; designers have to try to deal with all, or at least most computer users.

The ultimate goal in the level of complexity for this microcode generator is to be independent of the target machine. This project was targeted for a specific machine and so has not yet reached the ultimate goal. After this implementation is completed the next step is to write the program to handle some area of generalization. The user should enter certain constants concerning his machine either at the beginning or during each coding session. In the context of the C language, header files for many specific machines could be developed. The user obtains only the header files he needs. A configuration program may need to be developed so that the basic program could be configured by the user at one initial session. There are an infinite number of approaches to take in the continued development and maintenance of the code generator. In any case, the non-specific microcode generator would be an invaluable tool for a designer working with the development of microprogrammed instruction sets.

C. SUMMARY

The general topic of microprogramming was discussed in terms of the microprogrammed control unit. Software Engineering theory and practice was outlined in chapter three. The design approach used in this project was developed using these principles. Chapter four discussed significant points as addressed in the implementation of the C language program. An important point is that vertical microprogramming techniques, the sharing of function fields in the microword, give rise to potential conflicts between the bit patterns required for the conflicting functions. This is one of many tedious tasks for the microprogrammer.

This thesis has addressed the problem of automating the process of functional microprogramming and provided some solutions to the approach and implementation of a microcode generator.

LIST OF REFERENCES

1. Parnas, D. L., "On the Criteria To Be Used in Decomposing Systems into Modules," Communications of the ACM, December 1972.

2. Myers, G. J., Advances in Computer Architecture 2nd ed., Wiley, 1978.

3. Wilkes, M. V., "The Best Way to Design an Automatic Calculating Machine," paper presented at Manchester University Computer Inaugural Conference, Manchester, England, July 1951.

4. Stone, H. S., gen. ed., Introduction to Computer Architecture 2nd ed., SRA Computer Science Series, 1980.

5. Tanenbaum, A. S., Structured Computer Organization 2nd ed., Prentice-Hall, 1984.

6. Hartrum, T. C., Lamont, G. B. and Ross, A. A., "AMD Am29203 Evaluation Board User's Guide," preliminary draft, 1983.

7. Pressman, R. S., Software Engineering: A Practitioner's Approach, McGraw-Hill, 1982.

8. Dijkstra, E. W., "Go-To Considered Harmful," Letter to the Editor, Communications of the ACM, Vol 11, No. 3, March 1968.

9. MacLennan, B. J., Principles of Programming Languages, Holt, Rinehart and Winston, 1983.

10. Kernighan, B. W. and Ritchie, D. M., The C Programming Language, Prentice-Hall, 1978.

11. Shneiderman, B., "Human Factors Experiments in Designing Interactive Systems," IEEE Computer, December 1979.

12. Freeman, P., "Fundamentals of Design," Tutorial on Software Design Techniques, 4th ed., IEEE Computer Society Press, 1983.

13. Mick, J. and Brick, J., Bit-Slice Microprocessor Design, McGraw-Hill, 1980.

14. Advanced Micro Devices, Bipolar Microprocessor Logic and Interface Data Book, 1983.

APPENDIX A

The following is a record of a terminal session running the Sequencer module.

```
% test3
                   MASTER AM2910 SEQUENCER MENU

     XXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXX
           ffff              ffff              ffff

     The X s indicate bits which are not yet defined.

     What do you want to do next?
          Enter a  0  to select SEQUENCER COMMAND
                   H  for HELP with this program
                   R  to RETURN to system
0
```

```
                    AM2910 SEQUENCER COMMAND MENU

      Which AM2910 Sequencer Command do you wish to Chose?

 Enter a  0  JUMP ZERO - JZ
          1  CONDITIONAL JUMP SUBROUTINE - CJS
          2  JUMP MAP - JMAP
          3  CONDITIONAL JUMP PIPELINE - CJP
          4  PUSH/CONDITIONAL LOAD REGISTER/PIPELINE - PUSH
          5  CONDITIONAL JUMP SUB. VIA REG OR PIPELINE - JSRP
          6  CONDITIONAL JUMP VECTOR - CJV
          7  CONDITIONAL JUMP VIA REGISTER OR PIPELINE
          8  REPEAT LOOP, COUNTER NOT EQUAL 0 -  RFCT
          9  REPEAT PIPELINE, COUNTER NOT EQUAL 0 - RPCT
          A  CONDITIONAL RETURN FROM SUBROUTINE
          B  CONDITIONAL JUMP PIPELINE AND POP
          C  LOAD COUNTER AND CONTINUE - LDCT
          D  TEST FOR END OF LOOP - LOOP
          E  CONTINUE - CONT
          F  THREE WAY BRANCH - TWB
          H  HELP with this program
          R  RETURN to higher level
 0
```

```
                   MASTER AM2910 SEQUENCER MENU

        XXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXX  XXXXXXXXXXXX0000
              ffff              ffff              fff0

        The X s indicate bits which are not yet defined.

        What do you want to do next?
             Enter a  0  to select SEQUENCER COMMAND
                      H  for HELP with this program
                      R  to RETURN to system
   0
```

```
                 AM2910 SEQUENCER COMMAND MENU

     Which AM2910 Sequencer Command do you wish to Chose?

Enter a  0  JUMP ZERO - JZ
         1  CONDITIONAL JUMP SUBROUTINE - CJS
         2  JUMP MAP - JMAP
         3  CONDITIONAL JUMP PIPELINE - CJP
         4  PUSH/CONDITIONAL LOAD REGISTER/PIPELINE - PUSH
         5  CONDITIONAL JUMP SUB. VIA REG OR PIPELINE - JSRP
         6  CONDITIONAL JUMP VECTOR - CJV
         7  CONDITIONAL JUMP VIA REGISTER OR PIPELINE
         8  REPEAT LOOP, COUNTER NOT EQUAL 0 -  RFCT
         9  REPEAT PIPELINE, COUNTER NOT EQUAL 0 - RPCT
         A  CONDITIONAL RETURN FROM SUBROUTINE
         B  CONDITIONAL JUMP PIPELINE AND POP
         C  LOAD COUNTER AND CONTINUE - LDCT
         D  TEST FOR END OF LOOP - LOOP
         E  CONTINUE - CONT
         F  THREE WAY BRANCH - TWB
         H  HELP with this program
         R  RETURN to higher level
4
The sequencer code is already set.
Do you want to change it?
y
```

```
          AM2910 SEQUENCER BRANCH ADDRESS MENU

You have chosen a command which requires a value in the
register/counter

What do you want to do next?

     ENTER YOUR BRANCH ADDRESS FIELD
           H  for HELP with this program
           R  to RETURN to a higher level
123
This is the address being used.123
```

```
          AM2910 SEQUENCER CONDITION SELECT MENU

You have chosen an AM2910 Sequencer Command which requires a
     conditional test

What do you want to do next?

     Type a  P  for FORCED PASS - unconditional
             F  for FORCED FAIL
             T  to TEST the condition
             H  for HELP with this program
             R  to RETURN to higher level
f
```

```
                    REMINDER INFORMATION

You have chosen a PUSH/CONDITIONAL LOAD REGISTER/COUNTER
PUSH
     as the AM2910 Sequencer Command

This command MUST precede the following commands:

          RFCT  REPEAT LOOP, COUNTER NOT EQUAL 0
          CJPP  CONDITIONAL JUMP PIPELINE AND POP
          LOOP  TEST FOR END OF LOOP
          TWB   THREE WAY BRANCH


Press enter to continue
```

```
            AM2904 CONDITIONAL TEST MENU

      There are two steps to selecting a test condition
         1) select a REGISTER to be used
         2) select a TEST on that register

      This menu selects the register ot two special tests
      which combine two registers

      What do you want to do?

      Type a  0  for the Micro status register
              1  for the MACRO Status Register
              2  for the Immediate Status Inputs
              3  for Immediate Sign EXOR MACRO Sign
              4  for Immediate Sign EXNOR MARCO Sign
              H  for HELP with this menu
              R  to RETURN to a higher level
  0
```

```
          AM2910 SEQUENCER CONDITION SELECT MENU

You have chosen an AM2910 Sequencer Command which requires a
     conditional test

What do you want to do next?

     Type a  P  for FORCED PASS - unconditional
             F  for FORCED FAIL
             T  to TEST the condition
             H  for HELP with this program
             R  to RETURN to higher level
t
```

```
                  AM2910 SEQUENCER COMMAND MENU

     Which AM2910 Sequencer Command do you wish to Chose?

Enter a  0  JUMP ZERO - JZ
         1  CONDITIONAL JUMP SUBROUTINE - CJS
         2  JUMP MAP - JMAP
         3  CONDITIONAL JUMP PIPELINE - CJP
         4  PUSH/CONDITIONAL LOAD REGISTER/PIPELINE - PUSH
         5  CONDITIONAL JUMP SUB. VIA REG OR PIPELINE - JSRP
         6  CONDITIONAL JUMP VECTOR - CJV
         7  CONDITIONAL JUMP VIA REGISTER OR PIPELINE
         8  REPEAT LOOP, COUNTER NOT EQUAL 0 -  RFCT
         9  REPEAT PIPELINE, COUNTER NOT EQUAL 0 - RPCT
         A  CONDITIONAL RETURN FROM SUBROUTINE
         B  CONDITIONAL JUMP PIPELINE AND POP
         C  LOAD COUNTER AND CONTINUE - LDCT
         D  TEST FOR END OF LOOP - LOOP
         E  CONTINUE - CONT
         F  THREE WAY BRANCH - TWB
         H  HELP with this program
         R  RETURN to higher level
d
The sequencer code is already set.
Do you want to change it?
y
```

```
               MASTER AM2910 SEQUENCER MENU

      XXXXXXXXXXXXXXXX  XXXXXXXXXX1XXXXX  XXXXXXXXXXXX1010
            ffff              ffff              fffa

      The X s indicate bits which are not yet defined.

      What do you want to do next?
           Enter a  0  to select SEQUENCER COMMAND
                    H  for HELP with this program
                    R  to RETURN to system
 0
```

```
          AM2910 SEQUENCER CONDITION SELECT MENU

You have chosen an AM2910 Sequencer Command which requires a
     conditional test

What do you want to do next?

     Type a  P  for FORCED PASS - unconditional
             F  for FORCED FAIL
             T  to TEST the condition
             H  for HELP with this program
             R  to RETURN to higher level
P
```

```
                   AM2910 SEQUENCER COMMAND MENU

     Which AM2910 Sequencer Command do you wish to Chose?

Enter a  0  JUMP ZERO - JZ
         1  CONDITIONAL JUMP SUBROUTINE - CJS
         2  JUMP MAP - JMAP
         3  CONDITIONAL JUMP PIPELINE - CJP
         4  PUSH/CONDITIONAL LOAD REGISTER/PIPELINE - PUSH
         5  CONDITIONAL JUMP SUB. VIA REG OR PIPELINE - JSRP
         6  CONDITIONAL JUMP VECTOR - CJV
         7  CONDITIONAL JUMP VIA REGISTER OR PIPELINE
         8  REPEAT LOOP, COUNTER NOT EQUAL 0 -  RFCT
         9  REPEAT PIPELINE, COUNTER NOT EQUAL 0 - RPCT
         A  CONDITIONAL RETURN FROM SUBROUTINE
         B  CONDITIONAL JUMP PIPELINE AND POP
         C  LOAD COUNTER AND CONTINUE - LDCT
         D  TEST FOR END OF LOOP - LOOP
         E  CONTINUE - CONT
         F  THREE WAY BRANCH - TWB
         H  HELP with this program
         R  RETURN to higher level
a
The sequencer code is already set.
Do you want to change it?
y
```

```
                MASTER AM2910 SEQUENCER MENU

       XXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXX  XX10001000111100
             ffff              ffff              e23c

       The X s indicate bits which are not yet defined.

       What do you want to do next?
            Enter a  0  to select SEQUENCER COMMAND
                     H  for HELP with this program
                     R  to RETURN to system
   0
```

```
                    REMINDER INFORMATION

You have chosen a LOAD COUNTER AND CONTINUE -LDCT- as the
     AM2910 Sequencer Command

This command MUST precede the following:

          JRP   CONDITIONAL JUMP REGISTER OR PIPELINE
          RPCT  REPEAT PIPELINE, COUNTER NOT EQUAL 0


Press enter to continue
```

```
           AM2910 SEQUENCER BRANCH ADDRESS MENU

You have chosen a command which requires a value in the
register/counter

What do you want to do next?

     ENTER YOUR BRANCH ADDRESS FIELD
           H  for HELP with this program
           R  to RETURN to a higher level
223
This is the address being used.223
```

```
                    AM2910 SEQUENCER COMMAND MENU

        Which AM2910 Sequencer Command do you wish to Chose?

Enter a  0  JUMP ZERO - JZ
         1  CONDITIONAL JUMP SUBROUTINE - CJS
         2  JUMP MAP - JMAP
         3  CONDITIONAL JUMP PIPELINE - CJP
         4  PUSH/CONDITIONAL LOAD REGISTER/PIPELINE - PUSH
         5  CONDITIONAL JUMP SUB. VIA REG OR PIPELINE - JSRP
         6  CONDITIONAL JUMP VECTOR - CJV
         7  CONDITIONAL JUMP VIA REGISTER OR PIPELINE
         8  REPEAT LOOP, COUNTER NOT EQUAL 0 -  RFCT
         9  REPEAT PIPELINE, COUNTER NOT EQUAL 0 - RPCT
         A  CONDITIONAL RETURN FROM SUBROUTINE
         B  CONDITIONAL JUMP PIPELINE AND POP
         C  LOAD COUNTER AND CONTINUE - LDCT
         D  TEST FOR END OF LOOP - LOOP
         E  CONTINUE - CONT
         F  THREE WAY BRANCH - TWB
         H  HELP with this program
         R  RETURN to higher level
c
The sequencer code is already set.
Do you want to change it?
y
```

```
                    MASTER AM2910 SEQUENCER MENU

XXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXX  XXXXXXXXXXXX1001
      ffff              ffff              fff9

The X s indicate bits which are not yet defined.

What do you want to do next?
     Enter a  0  to select SEQUENCER COMMAND
              H  for HELP with this program
              R  to RETURN to system
0
```

```
                REMINDER INFORMATION

You have chosen one of the following AM2910 Sequencer
Commands:

          JRP    JUMP REGISTER OR PIPELINE
          RPCT   REPEAT PIPELINE, COUNTER NOT EQUAL 0

These commands MUST be preceded by a
          LDCT - LOAD COUNTER AND CONTINUE


Press enter to continue
```

```
                  AM2910 SEQUENCER COMMAND MENU

      Which AM2910 Sequencer Command do you wish to Chose?

Enter a  0  JUMP ZERO - JZ
         1  CONDITIONAL JUMP SUBROUTINE - CJS
         2  JUMP MAP - JMAP
         3  CONDITIONAL JUMP PIPELINE - CJP
         4  PUSH/CONDITIONAL LOAD REGISTER/PIPELINE - PUSH
         5  CONDITIONAL JUMP SUB. VIA REG OR PIPELINE - JSRP
         6  CONDITIONAL JUMP VECTOR - CJV
         7  CONDITIONAL JUMP VIA REGISTER OR PIPELINE
         8  REPEAT LOOP, COUNTER NOT EQUAL 0 -  RFCT
         9  REPEAT PIPELINE, COUNTER NOT EQUAL 0 - RPCT
         A  CONDITIONAL RETURN FROM SUBROUTINE
         B  CONDITIONAL JUMP PIPELINE AND POP
         C  LOAD COUNTER AND CONTINUE - LDCT
         D  TEST FOR END OF LOOP - LOOP
         E  CONTINUE - CONT
         F  THREE WAY BRANCH - TWB
         H  HELP with this program
         R  RETURN to higher level
9
The sequencer code is already set.
Do you want to change it?
y
```

```
                MASTER AM2910 SEQUENCER MENU

     XXXXXXXXXXXXXXXX  XXXXXXXXXX0X1000  XX01001000110100
           ffff                ffd8              d234

     The X s indicate bits which are not yet defined.

     What do you want to do next?
          Enter a  0  to select SEQUENCER COMMAND
                   H  for HELP with this program
                   R  to RETURN to system
0
```

```
                AM2904 CONDITIONAL TEST MENU

      What condition do you want reflected by the condition?

      Type a   0  for (SIGN exor OVR) or ZERO
               1  for (SIGN exnor OVR) and not ZERO
               2  for (SIGN exor OVR)
               3  for (SIGN exnor OVR)
               4  for ZERO
               5  for not ZERO
               6  for OVR
               7  for not OVR
               8  for (CARRY or ZERO)
               9  for (not CARRY) or (not ZERO)
               A  for CARRY
               B  for not CARRY
               C  for (not CARRY or ZERO)
               D  for (CARRY or not ZERO)
               E  for SIGN
               F  for not SIGN
               H  for HELP with this menu
               R  to RETURN to a higher level
  5
```

```
                REMINDER INFORMATION

You have chosen one of the following 2910 Sequencer Commands

          RFCT  REPEAT LOOP, COUNTER NOT EQUAL 0
          CJPP  CONDITIONAL JUMP PIPELINE AND POP
          LOOP  TEST FOR END OF LOOP
          TWB   THREE WAY BRANCH

These commands MUST be preceded by a

          PUSH - PUSH/CONDITIONAL LOAD REGISTER/COUNTER


Press enter to continue
```

```
                    MASTER AM2910 SEQUENCER MENU

        XXXXXXXXXXXXXXXX  XX0X0101XX0X1001  XXXXXXXXXXXX1101
              ffff               d5d9               fffd

        The X s indicate bits which are not yet defined.

        What do you want to do next?
             Enter a  0  to select SEQUENCER COMMAND
                      H  for HELP with this program
                      R  to RETURN to system
    0
```

```
                    AM2910 SEQUENCER COMMAND MENU

      Which AM2910 Sequencer Command do you wish to Chose?

Enter a  0  JUMP ZERO - JZ
         1  CONDITIONAL JUMP SUBROUTINE - CJS
         2  JUMP MAP - JMAP
         3  CONDITIONAL JUMP PIPELINE - CJP
         4  PUSH/CONDITIONAL LOAD REGISTER/PIPELINE - PUSH
         5  CONDITIONAL JUMP SUB. VIA REG OR PIPELINE - JSRP
         6  CONDITIONAL JUMP VECTOR - CJV
         7  CONDITIONAL JUMP VIA REGISTER OR PIPELINE
         8  REPEAT LOOP, COUNTER NOT EQUAL 0 -  RFCT
         9  REPEAT PIPELINE, COUNTER NOT EQUAL 0 - RPCT
         A  CONDITIONAL RETURN FROM SUBROUTINE
         B  CONDITIONAL JUMP PIPELINE AND POP
         C  LOAD COUNTER AND CONTINUE - LDCT
         D  TEST FOR END OF LOOP - LOOP
         E  CONTINUE - CONT
         F  THREE WAY BRANCH - TWB
         H  HELP with this program
         R  RETURN to higher level
7
The sequencer code is already set.
Do you want to change it?
y
```

```
          AM2910 SEQUENCER BRANCH ADDRESS MENU

You have chosen a command which requires a value in the
register/counter

What do you want to do next?

     ENTER YOUR BRANCH ADDRESS FIELD
           H  for HELP with this program
           R  to RETURN to a higher level
fff
This is the address being used.fff
Invalid input, the max hex number is 3FF.
Press enter to continue
```

```
          AM2910 SEQUENCER BRANCH ADDRESS MENU

You have chosen a command which requires a value in the
register/counter

What do you want to do next?

     ENTER YOUR BRANCH ADDRESS FIELD
           H  for HELP with this program
           R  to RETURN to a higher level
333
This is the address being used.333
```

```
          AM2910 SEQUENCER CONDITION SELECT MENU

You have chosen an AM2910 Sequencer Command which requires a
     conditional test

What do you want to do next?

     Type a  P  for FORCED PASS - unconditional
             F  for FORCED FAIL
             T  to TEST the condition
             H  for HELP with this program
             R  to RETURN to higher level
t
```

```
            AM2904 CONDITIONAL TEST MENU

There are two steps to selecting a test condition
   1) select a REGISTER to be used
   2) select a TEST on that register

This menu selects the register ot two special tests
which combine two registers

What do you want to do?

Type a  0  for the Micro status register
        1  for the MACRO Status Register
        2  for the Immediate Status Inputs
        3  for Immediate Sign EXOR MACRO Sign
        4  for Immediate Sign EXNOR MARCO Sign
        H  for HELP with this menu
        R  to RETURN to a higher level
4
```

```
                    REMINDER INFORMATION

You have chosen one of the following AM2910 Sequencer
Commands:

          JRP    JUMP REGISTER OR PIPELINE
          RPCT   REPEAT PIPELINE, COUNTER NOT EQUAL 0

These commands MUST be preceded by a
          LDCT - LOAD COUNTER AND CONTINUE


Press enter to continue
```

```
                MASTER AM2910 SEQUENCER MENU

       XXXXXXXXXXXXXXXX  XX001111XX0X1001  XX11001100110111
             ffff              cfd9              f337

       The X s indicate bits which are not yet defined.

       What do you want to do next?
            Enter a  0  to select SEQUENCER COMMAND
                     H  for HELP with this program
                     R  to RETURN to system
  0
```

```
                    AM2910 SEQUENCER COMMAND MENU

          Which AM2910 Sequencer Command do you wish to Chose?

Enter a  0  JUMP ZERO - JZ
         1  CONDITIONAL JUMP SUBROUTINE - CJS
         2  JUMP MAP - JMAP
         3  CONDITIONAL JUMP PIPELINE - CJP
         4  PUSH/CONDITIONAL LOAD REGISTER/PIPELINE - PUSH
         5  CONDITIONAL JUMP SUB. VIA REG OR PIPELINE - JSRP
         6  CONDITIONAL JUMP VECTOR - CJV
         7  CONDITIONAL JUMP VIA REGISTER OR PIPELINE
         8  REPEAT LOOP, COUNTER NOT EQUAL 0 -  RFCT
         9  REPEAT PIPELINE, COUNTER NOT EQUAL 0 - RPCT
         A  CONDITIONAL RETURN FROM SUBROUTINE
         B  CONDITIONAL JUMP PIPELINE AND POP
         C  LOAD COUNTER AND CONTINUE - LDCT
         D  TEST FOR END OF LOOP - LOOP
         E  CONTINUE - CONT
         F  THREE WAY BRANCH - TWB
         H  HELP with this program
         R  RETURN to higher level
f
The sequencer code is already set.
Do you want to change it?
y
```

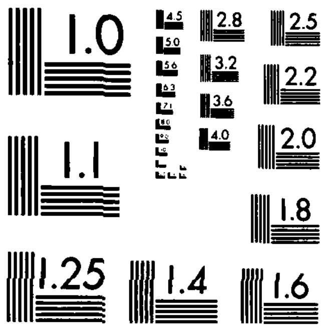
```
          AM2910 SEQUENCER BRANCH ADDRESS MENU

You have chosen a command which requires a value in the
register/counter

What do you want to do next?

     ENTER YOUR BRANCH ADDRESS FIELD
           H  for HELP with this program
           R  to RETURN to a higher level
211
This is the address being used.211
```

```
          AM2910 SEQUENCER CONDITION SELECT MENU

You have chosen an AM2910 Sequencer Command which requires a
     conditional test

What do you want to do next?

     Type a  P  for FORCED PASS - unconditional
             F  for FORCED FAIL
             T  to TEST the condition
             H  for HELP with this program
             R  to RETURN to higher level
t
```

```
          AM2904 CONDITIONAL TEST MENU

There are two steps to selecting a test condition
   1) select a REGISTER to be used
   2) select a TEST on that register

This menu selects the register ot two special tests
which combine two registers

What do you want to do?

Type a  0  for the Micro status register
        1  for the MACRO Status Register
        2  for the Immediate Status Inputs
        3  for Immediate Sign EXOR MACRO Sign
        4  for Immediate Sign EXNOR MARCO Sign
        H  for HELP with this menu
        R  to RETURN to a higher level
```
1

AD-A155 164 EVALUATION AND IMPLEMENTATION OF A FUNCTIONAL MICROPROGRAM GENERATOR(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA D B STILTNER DEC 84 2/3

UNCLASSIFIED F/G 9/2 NL


1.0
4.5
5.0
5.6
6.3
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

```
                 AM2904 CONDITIONAL TEST MENU

        What condition do you want reflected by the condition?

        Type a  0  for (SIGN exor OVR) or ZERO
                1  for (SIGN exnor OVR) and not ZERO
                2  for (SIGN exor OVR)
                3  for (SIGN exnor OVR)
                4  for ZERO
                5  for not ZERO
                6  for OVR
                7  for not OVR
                8  for (CARRY or ZERO)
                9  for (not CARRY) or (not ZERO)
                A  for CARRY
                B  for not CARRY
                C  for (not CARRY or ZERO)
                D  for (CARRY or not ZERO)
                E  for SIGN
                F  for not SIGN
                H  for HELP with this menu
                R  to RETURN to a higher level
    f
```

```
                    REMINDER INFORMATION

You have chosen one of the following 2910 Sequencer Commands

          RFCT  REPEAT LOOP, COUNTER NOT EQUAL 0
          CJPP  CONDITIONAL JUMP PIPELINE AND POP
          LOOP  TEST FOR END OF LOOP
          TWB   THREE WAY BRANCH

These commands MUST be preceded by a

          PUSH - PUSH/CONDITIONAL LOAD REGISTER/COUNTER


Press enter to continue
```

```
                    MASTER AM2910 SEQUENCER MENU

          XXXXXXXXXXXXXXXX  XX101111XX0X1001  XX10000100011111
                ffff              efd9              e11f

          The X s indicate bits which are not yet defined.

          What do you want to do next?
               Enter a  0  to select SEQUENCER COMMAND
                        H  for HELP with this program
                        R  to RETURN to system
     0
```

```
                    AM2910 SEQUENCER COMMAND MENU

       Which AM2910 Sequencer Command do you wish to Chose?

  Enter a  0  JUMP ZERO - JZ
           1  CONDITIONAL JUMP SUBROUTINE - CJS
           2  JUMP MAP - JMAP
           3  CONDITIONAL JUMP PIPELINE - CJP
           4  PUSH/CONDITIONAL LOAD REGISTER/PIPELINE - PUSH
           5  CONDITIONAL JUMP SUB. VIA REG OR PIPELINE - JSRP
           6  CONDITIONAL JUMP VECTOR - CJV
           7  CONDITIONAL JUMP VIA REGISTER OR PIPELINE
           8  REPEAT LOOP, COUNTER NOT EQUAL 0 -  RFCT
           9  REPEAT PIPELINE, COUNTER NOT EQUAL 0 - RPCT
           A  CONDITIONAL RETURN FROM SUBROUTINE
           B  CONDITIONAL JUMP PIPELINE AND POP
           C  LOAD COUNTER AND CONTINUE - LDCT
           D  TEST FOR END OF LOOP - LOOP
           E  CONTINUE - CONT
           F  THREE WAY BRANCH - TWB
           H  HELP with this program
           R  RETURN to higher level
  r
```

```
                         MASTER AM2910 SEQUENCER MENU

         XXXXXXXXXXXXXXXX  XX101111XX0X1001  XX10000100011111
               ffff              efd9              e11f

         The X s indicate bits which are not yet defined.

         What do you want to do next?
              Enter a  0  to select SEQUENCER COMMAND
                       H  for HELP with this program
                       R  to RETURN to system
    r
    Do you really want to leave?
    y
    %
```

The following is a record of a terminal session running the ALU module.

```
% test2
               MASTER AM29203 ALU MENU


     XXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXX
          ffff              ffff              ffff
The X s indicate bits which are not yet defined
The defaults for the AM29203 ALU are:
     Register Address Select - bits 47-45 - A,B pipeline =
111
     Instruction Enable - bit 44 - Disable = 1
     Output Enable - bit 43 - Disable = 1
     Source - bits 42-40 - DAQ = 111
     Destination - bits 39-36 - YBUS = 1111
     ALU Function - bits 35-32 - OR = 1111

What do you want to do next?
     type a  B  to choose ALU FUNCTIONS
          S  to choose SPECIAL FUNCTIONS
          H  for HELP with this program
          R  to RETURN to higher level
b
```

```
          AM29203 ALU BASIC FUNCTION SELECT

Enter the value corresponding to the function you wish to
perform
          0  F = High
          1  F = S - R - 1 + Carry In
          2  F = R - S - 1 + Carry In
          3  R + S + Carry In
          4  S + Carry In
          5  (NOT S) + Carry In
          6  R + Carry In
          7  F = (NOT R) + Carry In
          8  F = Low
          9  F = (NOT R) AND S
          A  F = R EXCLUSIVE OR S
          B  F = R EXCLUSIVE OR S
          C  F = R AND S
          D  F = R NOR S
          E  F = R NAND S
          F  F = R OR S
          H  for HELp with this program
          R  to RETURN to higher level
0
```

```
                   AM29203 ALU SOURCE MENU

You have chosen one of the following AM29203 ALU functions:
     F = High
     F = R + Carry In
     F + (NOT R) + Carry In
     F = LOW

For these functions, the only allowed AM29203 ALU Sources
are:

     Operand R       Operand S       Mnemonic

     RAMA            Q Register      RAMAQ
     Direct A        Q Register      DAQ

Type a  2  for RAMAQ
        6  for DAQ
        H  for HELP with this program
        R  to RETURN to a higher level
6
```

```
                    AM29203 ALU DESTINATION MENU

     Enter the value corresponding to the destination you
desire
        0  RAMDA -  F to RAM, Arithmetic Down Shift
        1  RAMDL -  F to RAM, Logical Down Shift
        2  RAMQDA - Double Precision Arithmetic Down Shift
        3  RAMQDL - Double Precision Logical Down Shift
        4  RAM - F to RAM with parity
        5  QD - F to Y, Down Shift Q
        6  LOADQ - F to Q with parity
        7  RAMQ - F to RAM with parity
        8  RAMUPA - F to RAM, Arithmetic Up Shift
        9  RAMUPL - F to RAM, Logical Up Shift
        A  RAMQUPA - Double Precision Arithmetic Up SHift
        B  RAMQUPL - Double Procision Logical Up SHift
        C            -  F to Y only
        D             - F to Y, Up SHift Q
        E  SIGNEXT - SIOO to Y(i)
        F  RAMEXT - F to Y, Sign extend LSB
        I  Instruction Register
        M  Main Memory
        H  for HELP with this program
        R  to RETURN to higher level
5
```

```
You have chosen a down shift for this microword.  There are
16 possible shift patterns, coded 0 thru F in bits I9
thru I6.  Choose the shift pattern you desire from the
following set:
     zero  =   0    -> RAMn,      0  -> Qn
      one  =   1    -> RAMn,      1  -> Qn
      two  =   0    -> RAMn,   RAM0 -> Mc,    Mn   -> Qn
     three =   1    -> RAMn,   RAM0 -> Qn
     four  =   Mc   -> RAMn,   RAM0 -> Qn
     five  =   Mn   -> RAMn,   RAM0 -> Qn
      six  =   0    -> RAMn,   RAM0 -> Qn
     seven =   0    -> RAMn,   RAM0 -> Qn,    Q0   -> Mc
     eight =  RAM0  -> RAMn,    Q0  -> Qn,   RAM0  -> Mc
     nine  =   Mc   -> RAMn,    Q0  -> Qn,   RAM0  -> Mc
       A   =  RAM0  -> RAMn,    Q0  -> Qn
       B   =   Ic   -> RAMn,   RAM0 -> Qn
       C   =   Mc   -> RAMn,   RAM0 -> Qn,    Q0   -> Mc
       D   =   Q0   -> RAMn,   RAM0 -> Qn,    Q0   -> Mc
       E   =  In exor I0vr -> RAMn,      RAM0 -> Qn
       F   =   Q0   -> RAMn,   RAM0 -> Qn
       H to get help with this procedure
       N to back up one frame.
 4
```

```
          AM29203 ALU INSTRUCTION AND OUTPUT ENABLE MENU

       Do you want the ALU results to appear on the Y-bus?
          Type an  Y  for YES
          Type a   N  for NO



y
       Do you want to change the contents of any ALU
register
       during this ALU operation?

          Type an  Y  for YES
          Type an  N  for NO
y
```

```
                MASTER AM29203 ALU MENU

     XXX0011001010000  XXXXXXXXXXX00100  XXXXXXXXXXXXXXXX
           e650              ffe4              ffff
The X s indicate bits which are not yet defined
The defaults for the AM29203 ALU are:
     Register Address Select - bits 47-45 - A,B pipeline =
111
     Instruction Enable - bit 44 - Disable = 1
     Output Enable - bit 43 - Disable = 1
     Source - bits 42-40 - DAQ = 111
     Destination - bits 39-36 - YBUS = 1111
     ALU Function - bits 35-32 - OR = 1111

What do you want to do next?
     type a  B  to choose ALU FUNCTIONS
             S  to choose SPECIAL FUNCTIONS
             H  for HELP with this program
             R  to RETURN to higher level
r
Do you really want to return to mastermenu?
y
% test2
```

```
              MASTER AM29203 ALU MENU

     XXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXX
           ffff              ffff              ffff
The X s indicate bits which are not yet defined
The defaults for the AM29203 ALU are:
     Register Address Select - bits 47-45 - A,B pipeline =
111
     Instruction Enable - bit 44 - Disable = 1
     Output Enable - bit 43 - Disable = 1
     Source - bits 42-40 - DAQ = 111
     Destination - bits 39-36 - YBUS = 1111
     ALU Function - bits 35-32 - OR = 1111

What do you want to do next?
     type a  B  to choose ALU FUNCTIONS
          S  to choose SPECIAL FUNCTIONS
          H  for HELP with this program
          R  to RETURN to higher level
b
```

```
                MASTER AM29203 ALU MENU

     XXX0010101111100  XXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXX
           e57c              ffff              ffff
The X s indicate bits which are not yet defined
The defaults for the AM29203 ALU are:
     Register Address Select - bits 47-45 - A,B pipeline =
111
     Instruction Enable - bit 44 - Disable = 1
     Output Enable - bit 43 - Disable = 1
     Source - bits 42-40 - DAQ = 111
     Destination - bits 39-36 - YBUS = 1111
     ALU Function - bits 35-32 - OR = 1111

What do you want to do next?
     type a  B  to choose ALU FUNCTIONS
             S  to choose SPECIAL FUNCTIONS
             H  for HELP with this program
             R  to RETURN to higher level
r
Do you really want to return to mastermenu?
y
% test2
```

```
          AM29203 ALU INSTRUCTION AND OUTPUT ENABLE MENU

       Do you want the ALU results to appear on the Y-bus?
          Type an  Y  for YES
          Type a   N  for NO



y
       Do you want to change the contents of any ALU
register
       during this ALU operation?

          Type an  Y  for YES
          Type an  N  for NO
y
```

```
                    AM29203 ALU DESTINATION MENU

     Enter the value corresponding to the destination you
desire
       0  RAMDA -  F to RAM, Arithmetic Down Shift
       1  RAMDL -  F to RAM, Logical Down Shift
       2  RAMQDA - Double Precision Arithmetic Down Shift
       3  RAMQDL - Double Precision Logical Down Shift
       4  RAM - F to RAM with parity
       5  QD - F to Y, Down Shift Q
       6  LOADQ - F to Q with parity
       7  RAMQ - F to RAM with parity
       8  RAMUPA - F to RAM, Arithmetic Up Shift
       9  RAMUPL - F to RAM, Logical Up Shift
       A  RAMQUPA - Double Precision Arithmetic Up SHift
       B  RAMQUPL - Double Procision Logical Up SHift
       C           -  F to Y only
       D            - F to Y, Up SHift Q
       E  SIGNEXT - SIO0 to Y(i)
       F  RAMEXT - F to Y, Sign extend LSB
       I  Instruction Register
       M  Main Memory
       H  for HELP with this program
       R  to RETURN to higher level
7
```

```
                    AM29203 ALU SOURCE MENU

The source control default is DAQ

          Operand R         Operand S     Mnemonic
Enter a  0  RAMA             RAMB          RAMAB
         1  RAMA             Direct B      RAMADB
         2  RAMA             Q Register    RAMAQ
         4  Direct A         RAMB          DARAMB
         5  Direct A         DirectB       DADB
         6  Direct A         Q Register    DAQ
         I      Instruction Register
         P      Pipeline Register
         H      for H with this program
         R      to RETURN to higher level
5
```

```
          AM29203 ALU BASIC FUNCTION SELECT

Enter the value corresponding to the function you wish to
perform
          0  F = High
          1  F = S - R - 1 + Carry In
          2  F = R - S - 1 + Carry In
          3  R + S + Carry In
          4  S + Carry In
          5  (NOT S) + Carry In
          6  R + Carry In
          7  F = (NOT R) + Carry In
          8  F = Low
          9  F = (NOT R) AND S
          A  F = R EXCLUSIVE OR S
          B  F = R EXCLUSIVE OR S
          C  F = R AND S
          D  F = R NOR S
          E  F = R NAND S
          F  F = R OR S
          H  for HELp with this program
          R  to RETURN to higher level
c
```

```
                    MASTER AM29203 ALU MENU

     XXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXX
           ffff              ffff              ffff
The X s indicate bits which are not yet defined
The defaults for the AM29203 ALU are:
     Register Address Select - bits 47-45 - A,B pipeline =
111
     Instruction Enable - bit 44 - Disable = 1
     Output Enable - bit 43 - Disable = 1
     Source - bits 42-40 - DAQ = 111
     Destination - bits 39-36 - YBUS = 1111
     ALU Function - bits 35-32 - OR = 1111

What do you want to do next?
     type a  B  to choose ALU FUNCTIONS
             S  to choose SPECIAL FUNCTIONS
             H  for HELP with this program
             R  to RETURN to higher level
b
```

```
                         MASTER AM29203 ALU MENU

     0011000101110000  XXXXXXXXXXXXXXXX  XXXX0101XXXXXXXX
          3170               ffff              f5ff
The X s indicate bits which are not yet defined
The defaults for the AM29203 ALU are:
     Register Address Select - bits 47-45 - A,B pipeline =
111
     Instruction Enable - bit 44 - Disable = 1
     Output Enable - bit 43 - Disable = 1
     Source - bits 42-40 - DAQ = 111
     Destination - bits 39-36 - YBUS = 1111
     ALU Function - bits 35-32 - OR = 1111

What do you want to do next?
     type a  B  to choose ALU FUNCTIONS
             S  to choose SPECIAL FUNCTIONS
             H  for HELP with this program
             R  to RETURN to higher level
r
Do you really want to return to mastermenu?
y
% test2
```

```
          AM29203 ALU RAM A REGISTER SELECT

              Enter the value corresponding to the RAM A
Register
    you wish to select

              0  RAMA A Register 0
              1  RAMA A Register 1
              2  RAMA A Register 2
              3  RAMA A Register 3
              4  RAMA A Register 4
              5  RAMA A Register 5
              6  RAMA A Register 6
              7  RAMA A Register 7
              8  RAMA A Register 8
              9  RAMA A Register 9
              A  RAMA A Register A
              B  RAMA A Register B
              C  RAMA A Register C
              D  RAMA A Register D
              E  RAMA A Register E
              F  RAMA A Register F
              H  for HELP with this menu
              R  to RETURN to a higher level
5
```

```
                AM29203 ALU REGISTER ADDRESS MENU

      The default source selection is Source A - pipeline,
      Source B - pipeline, Destination C - pipeline

      Enter the value corresponding to the register address
you desire
          Source A       Source B       Destination C
        0 Pipeline       Pipeline       Pipeline
        1 Instruction    Pipeline       Pipeline
        2 Pipeline       Instruction    Pipeline
        3 Instruction    Instruction    Pipeline
        4 Pipeline       Pipeline       Instruction
        5 Instruction    Pipeline       Instruction
        6 Pipeline       Instruction    Instruction
        7 Instruction    Instruction    Instruction
1
```

```
                    AM29203 ALU INSTRUCTION AND OUTPUT ENABLE MENU

               Do you want the ALU results to appear on the Y-bus?
                  Type an  Y  for YES
                  Type a   N  for NO



        y
               Do you want to change the contents of any ALU
        register
               during this ALU operation?

                  Type an  Y  for YES
                  Type an  N  for NO
        n
```

```
                    AM29203 ALU SOURCE SELECT

            You have chosen an AM29203 ALU Special Function

            What sources do you want to use

                        Operand R      Operand S      Mnemonic
            Enter a  0  RAMA A         RAM B          RAMAB
                     1  RAM A          DIRECT B       RAMADB
                     4  DIRECT A       RAM A          DARAMB
                     5  DIRECT A       DIRECT B       DADB
                     H  for HELP with this menu
                     R  to RETURN to a higher level
       1
```

```
                 AM29203 ALU SPECIAL FUNCTION MENU

Enter the value corresponding to the function you wish to
perform
       0  Unsigned multiply
       1  BCD to Binary Conversion
       M  Multiprecision BCD to Binary Conversion
       2  Two's Complement Multiply
       3  Decrement by 1 or 2
       4  Increment by 1 or 2
       5  Sign/Magnitude to Two's Complement Conversion
       6  Two's Complement Multiply
       7  BCD Divide by 2
       8  Single Length Mormalize
       9  Binary to BCD Conversion
       Z  Multiprecision Binary to BCD Conversion
       A  Double Length Normalize; First Division
       B  BCD  Add
       C  Two's Complement Divide
       D  BCD Subtract  F = R - S - 1 + Carry In BCD
       E  Two's Complement Divide Correction and Remainder
       F  BCD Subtract  F = S - R - 1 + Carry In BCD
       H  for HELP with this menu
       R  to RETURN to higher level
7
```

```
                    MASTER AM29203 ALU MENU

     XXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXX
           ffff              ffff              ffff
The X s indicate bits which are not yet defined
The defaults for the AM29203 ALU are:
     Register Address Select - bits 47-45 - A,B pipeline =
111
     Instruction Enable - bit 44 - Disable = 1
     Output Enable - bit 43 - Disable = 1
     Source - bits 42-40 - DAQ = 111
     Destination - bits 39-36 - YBUS = 1111
     ALU Function - bits 35-32 - OR = 1111

What do you want to do next?
     type a  B  to choose ALU FUNCTIONS
          S  to choose SPECIAL FUNCTIONS
          H  for HELP with this program
          R  to RETURN to higher level
s
```

```
                AM29203 ALU BASIC FUNCTION SELECT

Enter the value corresponding to the function you wish to
perform
          0  F = High
          1  F = S - R - 1 + Carry In
          2  F = R - S - 1 + Carry In
          3  R + S + Carry In
          4  S + Carry In
          5  (NOT S) + Carry In
          6  R + Carry In
          7  F = (NOT R) + Carry In
          8  F = Low
          9  F = (NOT R) AND S
          A  F = R EXCLUSIVE OR S
          B  F = R EXCLUSIVE OR S
          C  F = R AND S
          D  F = R NOR S
          E  F = R NAND S
          F  F = R OR S
          H  for HELp with this program
          R  to RETURN to higher level
2
```

```
     The Carry into the least significant stage of the ALU
is controlled by bits I12 and I11, and sometimes bits
I5, I3, I2, and I1. There are seven possible choices:
     Type a zero to select ZERO as the carry-in.
     Type a one to select ONE as the carry-in.
     Type a two to select Cx, the Z output of the 29203.
     Type a three to select the carry bit from the micro reg
     Type a four to select the micro carry bit complemented
     Type a five to select the MACRO carry bit
     Type a six to select the MACRO carry bit complemented
     Type an H for help.
1
```

```
                    AM29203 ALU SOURCE MENU

The source control default is DAQ

         Operand R        Operand S      Mnemonic
Enter a  0  RAMA          RAMB            RAMAB
         1  RAMA          Direct B        RAMADB
         2  RAMA          Q Register      RAMAQ
         4  Direct A      RAMB            DARAMB
         5  Direct A      DirectB         DADB
         6  Direct A      Q Register      DAQ
         I      Instruction Register
         P      Pipeline Register
         H      for H with this program
         R      to RETURN to higher level
0
```

```
                  AM29203 ALU DESTINATION MENU

     Enter the value corresponding to the destination you
desire
       0  RAMDA -   F to RAM, Arithmetic Down Shift
       1  RAMDL -   F to RAM, Logical Down Shift
       2  RAMQDA - Double Precision Arithmetic Down Shift
       3  RAMQDL - Double Precision Logical Down Shift
       4  RAM - F to RAM with parity
       5  QD - F to Y, Down Shift Q
       6  LOADQ - F to Q with parity
       7  RAMQ - F to RAM with parity
       8  RAMUPA - F to RAM, Arithmetic Up Shift
       9  RAMUPL - F to RAM, Logical Up Shift
       A  RAMQUPA - Double Precision Arithmetic Up SHift
       B  RAMQUPL - Double Procision Logical Up SHift
       C            -   F to Y only
       D             - F to Y, Up SHift Q
       E  SIGNEXT - SIO0 to Y(i)
       F  RAMEXT - F to Y, Sign extend LSB
       I  Instruction Register
       M  Main Memory
       H  for HELP with this program
       R  to RETURN to higher level
1
```

```
You have chosen a down shift for this microword.  There are
16 possible shift patterns, coded 0 thru F in bits I9
thru I6.  Choose the shift pattern you desire from the
following set:
     zero  =    0    -> RAMn,      0   -> Qn
      one  =    1    -> RAMn,      1   -> Qn
      two  =    0    -> RAMn,    RAM0 -> Mc,   Mn   -> Qn
     three =    1    -> RAMn,    RAM0 -> Qn
     four  =    Mc   -> RAMn,    RAM0 -> Qn
     five  =    Mn   -> RAMn,    RAM0 -> Qn
      six  =    0    -> RAMn,    RAM0 -> Qn
     seven =    0    -> RAMn,    RAM0 -> Qn,   Q0   -> Mc
     eight =  RAM0   -> RAMn,     Q0  -> Qn,  RAM0  -> Mc
     nine  =    Mc   -> RAMn,     Q0  -> Qn,  RAM0  -> Mc
       A   =  RAM0   -> RAMn,     Q0  -> Qn
       B   =    Ic   -> RAMn,    RAM0 -> Qn
       C   =    Mc   -> RAMn,    RAM0 -> Qn,   Q0   -> Mc
       D   =    Q0   -> RAMn,    RAM0 -> Qn,   Q0   -> Mc
       E   =  In exor I0vr -> RAMn,     RAM0 -> Qn
       F   =    Q0   -> RAMn,    RAM0 -> Qn
       H to get help with this procedure
       N to back up one frame.
3
```

```
                AM29203 ALU INSTRUCTION AND OUTPUT ENABLE MENU

             Do you want the ALU results to appear on the Y-bus?
                Type an  Y  for YES
                Type a   N  for NO




        y
             Do you want to change the contents of any ALU
        register
             during this ALU operation?

                Type an  Y  for YES
                Type an  N  for NO
        y
```

```
              AM29203 ALU REGISTER ADDRESS MENU

     The default source selection is Source A - pipeline,
     Source B - pipeline, Destination C - pipeline

     Enter the value corresponding to the register address
you desire
              Source A        Source B        Destination C
          0   Pipeline        Pipeline        Pipeline
          1   Instruction     Pipeline        Pipeline
          2   Pipeline        Instruction     Pipeline
          3   Instruction     Instruction     Pipeline
          4   Pipeline        Pipeline        Instruction
          5   Instruction     Pipeline        Instruction
          6   Pipeline        Instruction     Instruction
          7   Instruction     Instruction     Instruction
2
```

```
          AM29203 ALU RAM A REGISTER SELECT

               Enter the value corresponding to the RAM A
Register
     you wish to select

               0  RAMA A Register 0
               1  RAMA A Register 1
               2  RAMA A Register 2
               3  RAMA A Register 3
               4  RAMA A Register 4
               5  RAMA A Register 5
               6  RAMA A Register 6
               7  RAMA A Register 7
               8  RAMA A Register 8
               9  RAMA A Register 9
               A  RAMA A Register A
               B  RAMA A Register B
               C  RAMA A Register C
               D  RAMA A Register D
               E  RAMA A Register E
               F  RAMA A Register F
               H  for HELP with this menu
               R  to RETURN to a higher level
1
```

```
                    AM29203 ALU RAM B REGISTER SELECT

                Enter the value corresponding to the RAM B
Register
     you wish to select

                0  RAM B Register 0
                1  RAM B Register 1
                2  RAM B Register 2
                3  RAM B Register 3
                4  RAM B Register 4
                5  RAM B Register 5
                6  RAM B Register 6
                7  RAM B Register 7
                8  RAM B Register 8
                9  RAM B Register 9
                A  RAM B Register A
                B  RAM B Register B
                C  RAM B Register C
                D  RAM B Register D
                E  RAM B Register E
                F  RAM B Register F
                H  for HELP with this menu
                R  to RETURN to a higher level
e
```

```
               MASTER AM29203 ALU MENU

     0100000000010010  01XXXXXXXXX00011  XXXX00011110XXXX
            4012              7fe3              f1ef
The X s indicate bits which are not yet defined
The defaults for the AM29203 ALU are:
     Register Address Select - bits 47-45 - A,B pipeline =
111
     Instruction Enable - bit 44 - Disable = 1
     Output Enable - bit 43 - Disable = 1
     Source - bits 42-40 - DAQ = 111
     Destination - bits 39-36 - YBUS = 1111
     ALU Function - bits 35-32 - OR = 1111

What do you want to do next?
     type a  B  to choose ALU FUNCTIONS
             S  to choose SPECIAL FUNCTIONS
             H  for HELP with this program
             R  to RETURN to higher level
r
Do you really want to return to mastermenu?
y
% test2
```

APPENDIX B

Program Name: Seqmake
Purpose: The Makefile used to compile the Sequencer module and its submodules. The name Seqmake must be changed to makefile before using.

```
test3: 2910.o utils.o
     cc 2910.o utils.o -o test3
```

Program name: 2910.c
Purpose: Source code for Sequencer module.

```
#include        <stdio.h>
#include        "declare.h"

   /* these defines refer to the logical fields of the
microword they are used to pass field_set the fields --note:
these defines are commented out!! they are provided for
readability only.

#define         regsel 1
#define         ien_fld 2
#define         oey_fld 3
#define         source_fld 4
#define         dest_fld 5
#define         function_fld 6
#define         carryin_fld 7
#define         I5I4_fld 8
#define         I3_I0_fld 9
#define         I5_I0_fld 10
#define         ceu_fld 11
#define         cem_fld 12
#define         cmden_fld 13
#define         shiften_fld 14
#define         command_fld 15
#define         shift_fld 16
#define         breakpoint_fld 17
#define         notused_fld 18
#define         msb_br_fld 19
#define         mid_br_fld 20
#define         lsb_br_fld 21
#define         rega_fld 22
#define         regb_fld 23
#define         seq_fld 24
#define         no_sub 0
*/


int  KEEPgoing,goback,docu_word[24];
char cmd_line[80],*pcmd,*pmwd,micro_word[49];
main()
{
int  i,helpset;
char CONTINUEcommand[4],am2910command[80];

/* Initialize micro_word to 'X' and docu_word to 0 */

micro_word[48] = '\0';
for (i=0;i < 48; i++)
```

```
     micro_word[i] = 'X';

for (i=0;i < 24;i++)
     docu_word[i] = 0;


KEEPgoing = true;
helpset = false;
goback = false;

while (KEEPgoing || helpset)  /* Need both KEEPgoing and   */
  {                           /* helpset = 0 to get out of*/
  helpset = false;            /* main program section.    */
  am2910menu(am2910command);  /* KEEPgoing is global and  *
                              /* can be set from ext proc.*/
  if ( *am2910command == '0')
      {
      COMMANDselect();
      if (goback)
         {
         helpset = true;
         goback = false;
         }
      }
  else
  if (*am2910command == 'H'  || *am2910command == 'h')
     {
         helpset = true;
         puts("Help will be coming soon!.\0");
         puts("Press enter to continue\0");
         gets(CONTINUEcommand);
      }
  else
  if ((*am2910command == 'R' || *am2910command == 'r')
                             || (KEEPgoing == false))
         {
         puts("Do you really want to leave?\0");
         gets(CONTINUEcommand);
         switch(*CONTINUEcommand)
           {
           case 'YES':
           case 'yes':
           case 'Y':
           case 'y':
               KEEPgoing = false;
               break;
           }
         }
  else
      {
      helpset = true;
```

```
    puts("Your input is invalid, enter 0,h,H,r,R  only. 0");
      sleep(3);
      }
    } /* while */
  exit();
} /* procedure am2910master */


/*************************************************************
/*                                                         */
/*                  AM2910 MENU PROCEDURES                 *
/*                                                         */
/*************************************************************


am2910menu(am2910command)

char *am2910command;

{
puts (erase_screen);
puts("\t\t\tMASTER AM2910 SEQUENCER MENU\n\0");
display_word();
display_in_hex();
display_docu();
puts("\tThe X s indicate bits which are not yet
                                          defined.\n\0");
puts("\tWhat do you want to do next?\0");
puts("\t\tEnter a  0  to select SEQUENCER COMMAND\0");
puts("\t\t         H  for HELP with this program\0");
puts("\t\t         R  to RETURN to system\0");
gets(am2910command);

} /* procedure AM2910menu */


SEQUENCERmenu(SEQUENCERcommand)

char *SEQUENCERcommand;

{
  puts(erase_screen);
  puts("\t\t\tAM2910 SEQUENCER COMMAND MENU\n\0");
  puts("\tWhich AM2910 Sequencer Command do you wish to
                                               Chose?\n 0");
  puts("Enter a  0  JUMP ZERO - JZ\0");
  puts("         1  CONDITIONAL JUMP SUBROUTINE - CJS\0");
  puts("         2  JUMP MAP - JMAP\0");
  puts("         3  CONDITIONAL JUMP PIPELINE - CJP\0");
  puts("         4  PUSH/CONDITIONAL LOAD REGISTER/PIPELINE
                                                    - PUSH\0");
      case 12:
      case 13:
      case 14:
      case 17:
      case 18:
      case 19:
      case 20:
            docu_word[field-1] = -1;
            break;
      case 8:   /* The conditional testing function includes
                   physical fields 8,9,13,15.  This will be
                   covered by docu field #8 (case 8 in the
                   docu procedure).  Sub_set will hold an
                   integer representing the function chosen.
                   (i.e. forced pass, forced fail, conditional
                   testing.)  In going backwards, a table or
                   big switch will be needed to translate.  *

            docu_word[7]=sub_set;
            break;
      case 24:
            docu_word[field-1] = sub_set;
            break;
      }  /* end switch */
      display_docu();
   }  /* end docu */


field_set(field_cnt,sub_set,choice)
  int  field_cnt,sub_set;
  char *choice;

  /*There are 25 defined fields in the 29203 eval board
microword, and several of them have multiple definitions. In
this routine, we accept a pointer to the field and to the
definition of the subset, and a pointer to a character which
represents the actual choice. We generate a data structure
which holds the choice and the actual bit pattern in the
microword. */

   {
   char scrap[4];
   switch(field_cnt)
      {
      case 1: /* regsel_fld, register address source    */
         octal_field(0,choice);
         break;

      case 2: /* ien_fld, 29203 instruction enable   */
         binary_field(3,choice);
         break;
```

Program Name: Utils.c
Purpose: Source code for the Utilities module.

```
#include     "extern.h"
#include     "declare.h"
#include     <stdio.h>

bad_choice(choice)
char *choice;
  {
  puts("bad_choice called.\0");
  putchar(*choice);
  sleep (1);
  }

docu (field,sub_set,choice)
int field,sub_set;
char *choice;

  {
  puts("docunew called\0");
/*This procedure sets the elements in an array called
docu_word.  Each element corresponds to a docu field (an
int) which has a code in it indicating whether the docu
field is being used.  Unfortunately the docu fields don't
necessarily match the physical fields used by field_set.
This was necessary since each physical field doesn't always
stand alone.  Ex. the three physical fields for the branch
address are always set together.  So docu has only one field
for that function.  So the size of the docu_word array will
be changing as new modules are added.  At the end, it can be
adjusted to try for some matching of names or numbers or
both. */

/*  The code for each element is:
          a #  means which sub_set function was selected.
          -1   means this element is a selected docu field
               the values can be obtained from the
               micro_word.
          0    means not set.   */
switch(field)
  {
     case 1:
     case 2:
     case 3:
     case 4:
     case 5:
     case 6:
     case 7:
     case 11:
            BRANCHaddress();

 if((*SEQUENCERcommand == '1') || (*SEQUENCERcommand == '3') ||
  (*SEQUENCERcommand == '4') || (*SEQUENCERcommand == '5') ||
  (*SEQUENCERcommand == '6') || (*SEQUENCERcommand == '7') ||
  (*SEQUENCERcommand == 'A') || (*SEQUENCERcommand == 'B') ||
  (*SEQUENCERcommand == 'a') || (*SEQUENCERcommand == 'b') ||
  (*SEQUENCERcommand == 'D') || (*SEQUENCERcommand == 'F') ||
  (*SEQUENCERcommand == 'd') || (*SEQUENCERcommand == 'f'))
                                                     && (!goback))
        CONDITIONsequencer();

  if ((*SEQUENCERcommand == '4') && (!goback))
        PUSHmenu();
  else
  if ((*SEQUENCERcommand == 'C' || *SEQUENCERcommand == 'c')
            && (!goback))
        LDCTmenu();
 else
 if(((*SEQUENCERcommand == '8')||(*SEQUENCERcommand == 'B')||
   (*SEQUENCERcommand == 'd') || (*SEQUENCERcommand == 'f') ||
   (*SEQUENCERcommand == 'D') || (*SEQUENCERcommand == 'F'))
                                                     && (!goback))
        NEEDPUSHmenu();
 else
 if(((*SEQUENCERcommand == '7')||(*SEQUENCERcommand == '9'))
                                                     && (!goback))
        NEEDLDCTmenu();

       if(goback)
         {
         helpset = true;
         goback = false;
         }
   }
 while(helpset);
 } /* procedure COMMAND select */
```

```
                    seq. code can be done cleanly i.e. without
                    leaving branch addr and cond codes from
                    previous settings of the seq_fld.  The
                    docu_word entry can be read and "decoded"
                    to reset any appropriate fields.       *


        case '0':
        case '2':
        case '8':
        case '9':
        case 'e':
        case 'E':
                field_set(seq_fld,1,SEQUENCERcommand);
                break;
        case 'c':
        case 'C':
                field_set(seq_fld,2,SEQUENCERcommand);
                break;
        case '6':
        case 'a':
        case 'A':
        case 'd':
        case 'D':
                field_set(seq_fld,3,SEQUENCERcommand);
                break;
        case '1':
        case '3':
        case '4':
        case '5':
        case '7':
        case 'b':
        case 'B':
        case 'f':
        case 'F':
                field_set(seq_fld,4,SEQUENCERcommand);
                break;
        default:
               helpset = true;
               puts("Invalid input, digits or ALL CAPS or
                                                  r,R,h,H\0");
               puts("Press enter to continue.\0");
               gets(CONTINUECommand);
               break;
        }
    if((*SEQUENCERcommand == '1') :: (*SEQUENCERcommand == '3')
     (*SEQUENCERcommand == '4') :: (*SEQUENCERcommand == '5')
     (*SEQUENCERcommand == '7')    (*SEQUENCERcommand == 'B')
     (*SEQUENCERcommand == 'C') :: (*SEQUENCERcommand == 'F')
     (*SEQUENCERcommand == 'c') :  (*SEQUENCERcommand == 'f')
     (*SEQUENCERcommand == 'b'))
```

```
            while(helpl);
               break;
        default:
            helpset = true;
            puts("Invalid input, wait for menu.\0");
            sleep(1);
            break;
        }
     }
while(helpset);
} /* procedure CONDITION sequencer */


COMMANDselect()
{
char CONTINUEcommand[4],*SEQUENCERcommand,comd_line[80];
int helpset;

SEQUENCERcommand = comd_line;

do
  {
  helpset = false;
  SEQUENCERmenu(SEQUENCERcommand);

  switch(* SEQUENCERcommand )
     {
     case 'r':
     case 'R':
             /*  do nothing, keeps helpset = false  */
             break;
     case 'h':
     case 'H':
             helpset =true;
             puts("Help is coming, Real Soon Now!\0");
             puts("Press enter to continue.\0");
             gets(CONTINUEcommand);
             break;

                 /* The sub_set parameter of the field_set
                    procedure is being used here to indicate
                    four groupings of the choices for the
                    sequencer field:
                    1  means seq. field only is set.
                    2  means branch address field is set.
                    3  means conditional codes are set.
                    4  means all three of above are set.

                  The above codes are put into the docu_word
                  so that subsequent attempts to change the
```

```
case 'F':  /* FORCED FAIL */
        field_set(condtest_fld,2,CONDITIONcommand);
        break;
case 't':
case 'T':  /*  TEST the condition  */
  do
    {
    help1 = false;
    TEST1menu(CONDITIONcommand);
    switch(*CONDITIONcommand)
       {
       case 'r':
       case 'R':
         helpset = true;
         break;
       case 'h':
       case 'H':
         help1 = true;
         puts("Here's where help would be nice!\0");
         puts("Press enter to continue.\0");
         gets(CONTINUEcommand);
         break;
       default:
         if(cond_1_set(CONDITIONcommand))
            {
            do
              {
              help2 = false;
              TEST2menu(CONDITIONcommand);
              switch(*CONDITIONcommand)
              {
              case 'r':
              case 'R':
                 helpset = true;
                 break;
              case 'h':
              case 'H':
                 help2 = true;
                 puts("Help goes here!\0");
                 puts("Press enter to cont.\0");
                 gets(CONTINUEcommand);
                 break;
              default:
                 cond_2_set(CONDITIONcommand);
                 break;
               }
              }
            while(help2);
            }
         break;
        }
```

```
                {
                field_set(msb_br_fld,0,branchselect);
                }
            else
                {
                helpset = true;
                puts("Invalid input, the max hex number is
                                                      3FF.\0");
                puts("Press enter to continue\0");
                gets(CONTINUEcommand);
                }
            break;
        }
    }   /*  while   */
} /* procedure branch address select */



CONDITIONsequencer()

{
char *CONTINUEcommand,*CONDITIONcommand,cmd_line[80],
     cont_line[10];
int helpset,help1,help2;

CONDITIONcommand = cmd_line;
CONTINUEcommand = cont_line;

do
  {
  helpset = false;
  CONDITIONmenu(CONDITIONcommand);
  switch(*CONDITIONcommand)
      {
      case 'h':
      case 'H':
             helpset = true;
             puts("Help is coming Real Soon Now!\0");
             puts("Press enter to continue.\0");
             gets(CONTINUEcommand);
             break;
      case 'r':
      case 'R':
             goback = true;
             break;

      case 'p':
      case 'P':   /* FORCED PASS--unconditional */
             field_set(condtest_fld,1,CONDITIONcommand);
             break;
      case 'f':
```

```
/**********************************************************/
/*                                                        */
/*              AM2910 PROCESSING PROCEDURES              */
/*                                                        */
/**********************************************************/


BRANCHaddress ()

{
char *branchselect,*CONTINUEcommand,cmd_line[80],
     cont_line[10];
int  helpset;

branchselect = cmd_line;
CONTINUEcommand = cont_line;
helpset = true;
while (helpset)
  {
  helpset = false;
  BRANCHmenu(branchselect);
  switch(*branchselect)
     {
     case 'H':
     case 'h':   /*  help   */
             helpset = true;
             puts("The branch address field is 12 bits
                                                long,\0");
             puts("the max hex address is 3FF.\0");
             puts("Enter anything to continue.\0");
             gets(CONTINUEcommand);
             break;

     case 'R':
     case 'r':   /*  Return   */
             goback = true;
             break;

     default:
         printf("This is the address being
                                 used.%s\n",branchselect);
         CONTINUEcommand = branchselect;
       if((*CONTINUEcommand<='3'&&*CONTINUEcommand>='0')
       &&((*(++CONTINUEcommand)>='0'&&*CONTINUEcommand<='7')
           ||(*CONTINUEcommand>='A'&&*CONTINUEcommand<='F')
           ||(*CONTINUEcommand>='a'&&*CONTINUEcommand<='f'))
       &&((*(++CONTINUEcommand)>='0'&&*CONTINUEcommand<='7')
           ||(*CONTINUEcommand>='A'&&*CONTINUEcommand<='F')
          ||(*CONTINUEcommand>='a'&&*CONTINUEcommand<='f')))
} /* procedure TEST2 */


cond_2_set(pchar)
     char (*pchar);
/* This is the second level selection of the cond. test */
     {
     switch (*pchar)
          {
          case '0':  /* SIGN exor OVR or ZERO */
          case '1':  /* SIGN exnor OVR and not ZERO */
          case '2':  /* SIGN exor OVR  */
          case '3':  /* SIGN exnor OVR */
          case '4':  /* ZERO */
          case '5':  /* not ZERO */
          case '6':  /* OVR */
          case '7':  /* not OVR */
          case '8':  /* CARRY or ZERO */
          case '9':  /* not CARRY or not ZERO */
          case 'A':  /* CARRY */
          case 'a':
          case 'B':  /* not CARRY */
          case 'b':
          case 'C':  /* not CARRY or ZERO */
          case 'c':
          case 'D':  /* CARRY or  not ZERO */
          case 'd':
               field_set(condtest_fld,4,pchar);
               if (micro_word[I05_04] =='0')
                   bit_erase(I04_04);
               break;
          case 'E':  /* SIGN */
          case 'e':
          case 'F':  /* not SIGN */
          case 'f':
               field_set(condtest_fld,4,pchar);
               break;
          }
     }   /* end procedure cond_2_set  */
```

```
          case '2':  /* Immediate Inputs. */
               bit_set(I05_04);
               bit_set(I04_04);
               break;

          case '3': /* Imm. sign exor MSR sign */
               *pchar = 'e';
               field_set(condtest_fld,3,pchar);
               next_level = FALSE;
               break;

          case '4': /* Imm. sign exnor MSR sign */
               *pchar = 'f';
               field_set(condtest_fld,3,pchar);
               next_level = FALSE;
               break;
          }
     return (next_level);
     } /* end cond_l_set */


TEST2menu(TEST2select)

char *TEST2select;

{
  puts(erase_screen);
  puts("\t\t\tAM2904 CONDITIONAL TEST MENU\n\0");
  puts("     What condition do you want reflected by the
                                             condition?\n\0");
  puts("     Type a  0  for (SIGN exor OVR) or ZERO\0");
  puts("             1  for (SIGN exnor OVR) and not
                                                     ZERO\0");
  puts("             2  for (SIGN exor OVR)\0");
  puts("             3  for (SIGN exnor OVR)\0");
  puts("             4  for ZERO\0");
  puts("             5  for not ZERO\0");
  puts("             6  for OVR\0");
  puts("             7  for not OVR\0");
  puts("             8  for (CARRY or ZERO)\0");
  puts("             9  for (not CARRY) or (not ZERO)\0");
  puts("             A  for CARRY\0");
  puts("             B  for not CARRY\0");
  puts("             C  for (not CARRY or ZERO)\0");
  puts("             D  for (CARRY or not ZERO)\0");
  puts("             E  for SIGN\0");
  puts("             F  for not SIGN\0");
  puts("             H  for HELP with this menu\0");
  puts("             R  to RETURN to a higher level\0");
  gets(TEST2select);
```

```
TESTlmenu(TESTlselect)

char *TESTlselect;

{
  puts(erase_screen);
  puts("\t\t\tAM2904 CONDITIONAL TEST MENU\n\0");
  puts("     There are two steps to selecting a test
                                                condition\0");
  puts("        1) select a REGISTER to be used\0");
  puts("        2) select a TEST on that register\n\0");
  puts("     This menu selects the register ot two special
                                                    tests\0");
  puts("     which combine two registers\n\0");
  puts("     What do you want to do?\n\0");
  puts("     Type a  0  for the Micro status register\0");
  puts("             1  for the MACRO Status Register\0");
  puts("             2  for the Immediate Status Inputs\0");
  puts("             3  for Immediate Sign EXOR MACRO
                                                     Sign\0");
  puts("             4  for Immediate Sign EXNOR MARCO
                                                     Sign\0");
  puts("             H  for HELP with this menu\0");
  puts("             R  to RETURN to a higher level\0");
  gets(TESTlselect);
} /* procedure TESTlmenu */


cond_l_set(pchar)
 /* This is the first level cond. code select, and matches
                                                TESTlmenu. */
     char *pchar;
     {
     int  next_level;
     char *field,field_line[4];
     next_level = TRUE;

     switch (*pchar)
          {
          case '0':  /* Micro status register selected. */
               bit_clear(I05_04);
               bit_set(I04_04);
/* Note that I04 can be cleared for many cases, see Tbl. 4,
                                                    Pg 5-79 */
               break;

          case '1':  /* Macro status register. */
               bit_set(I05_04);
               bit_clear(I04_04);
               break;
```

```
    puts("Press enter to continue\0");
    gets(CONTINUEcommand);

  } /* procedure need_push menu */


  NEEDLDCTmenu ()


  {
    char CONTINUEcommand[4];
    puts(erase_screen);
    puts("\t\t\tREMINDER INFORMATION\n\0");
    puts("You have chosen one of the following AM2910
                                  Sequencer Commands:\n\0");
    puts("\t\tJRP   JUMP REGISTER OR PIPELINE\0");
    puts("\t\tRPCT  REPEAT PIPELINE, COUNTER NOT EQUAL
                                                    0\n\0");
    puts("These commands MUST be preceded by a\0");
    puts("\t\tLDCT - LOAD COUNTER AND CONTINUE\n\n\n\0");
    puts("Press enter to continue\0");
    gets(CONTINUEcommand);

  } /* need_ldct menu */



  CONDITIONmenu (CONDITIONcommand)

  char *CONDITIONcommand;

  {
    puts(erase_screen);
    puts("\t\tAM2910 SEQUENCER CONDITION SELECT MENU\n\0");
    puts("You have chosen an AM2910 Sequencer Command which
                                               requires a\0");
    puts("\tconditional test\n\0");
    puts("What do you want to do next?\n\0");
    puts("\tType a  P  for FORCED PASS - unconditional\0");
    puts("\t        F  for FORCED FAIL\0");
    puts("\t        T  to TEST the condition\0");
    puts("\t        H  for HELP with this program\0");
    puts("\t        R  to RETURN to higher level\0");
    gets(CONDITIONcommand);

  } /* procedure condition menu */
  puts("\tas the AM2910 Sequencer Command\n\0");
  puts("This command MUST precede the following
                                         commands: n 0" ;
  puts("\t\tRFCT  REPEAT LOOP, COUNTER NOT EQUAL 0 0" ;
  puts("\t\tCJPP  CONDITIONAL JUMP PIPELINE AND POP 0" ;
  puts("\t\tLOOP  TEST FOR END OF LOOP\0");
  puts("\t\tTWB   THREE WAY BRANCH\n\n\n\0");
  puts("Press enter to continue\0");
  gets(CONTINUEcommand);

} /* procedure PUSHmenu */


LDCTmenu ()


{
  char CONTINUEcommand[4];
  puts(erase_screen);
  puts("\t\t\tREMINDER INFORMATION\n\0");
  puts("You have chosen a LOAD COUNTER AND CONTINUE -LDCT-
                                              as the\0" ;
  puts("\tAM2910 Sequencer Command\n\0");
  puts("This command MUST precede the following:\n\0");
  puts("\t\tJRP   CONDITIONAL JUMP REGISTER OR PIPELINE 0" ;
  puts("\t\tRPCT  REPEAT PIPELINE, COUNTER NOT EQUAL
                                             0\n\n\n 0" ;
  puts("Press enter to continue\0");
  gets(CONTINUEcommand);

} /* procedure LDCTmenu */


NEEDPUSHmenu()


{
  char CONTINUEcommand[4];
  puts(erase_screen);
  puts("\t\t\tREMINDER INFORMATION\n\0");
  puts("You have chosen one of the following 2910 Sequencer
                                           Commands\n\0" ;
  puts("\t\tRFCT  REPEAT LOOP, COUNTER NOT EQUAL 0\0");
  puts("\t\tCJPP  CONDITIONAL JUMP PIPELINE AND POP\0");
  puts("\t\tLOOP  TEST FOR END OF LOOP\0");
  puts("\t\tTWB   THREE WAY BRANCH\n\0");
  puts("These commands MUST be preceded by a\n\0");
  puts("\t\tPUSH - PUSH/CONDITIONAL LOAD
                                 REGISTER/COUNTER\n\n\n\0");
  puts("          5  CONDITIONAL JUMP SUB. VIA REG OR
                                       PIPELINE - JSRP 0");
  puts("          6  CONDITIONAL JUMP VECTOR - CJV\0");
  puts("          7  CONDITIONAL JUMP VIA REGISTER OR
                                              PIPELINE\0");
  puts("          8  REPEAT LOOP, COUNTER NOT EQUAL 0 -
                                                  RFCT\0");
  puts("          9  REPEAT PIPELINE, COUNTER NOT EQUAL 0 -
                                                  RPCT\0");
  puts("          A  CONDITIONAL RETURN FROM SUBROUTINE\0");
  puts("          B  CONDITIONAL JUMP PIPELINE AND POP\0");
  puts("          C  LOAD COUNTER AND CONTINUE - LDCT\0");
  puts("          D  TEST FOR END OF LOOP - LOOP\0");
  puts("          E  CONTINUE - CONT\0");
  puts("          F  THREE WAY BRANCH - TWB\0");
  puts("          H  HELP with this program\0");
  puts("          R  RETURN to higher level\0");
  gets(SEQUENCERcommand);

} /* procedure SEQUENCER menu */



BRANCHmenu(branchselect)

char *branchselect;

{
  puts(erase_screen);
  puts("\t\tAM2910 SEQUENCER BRANCH ADDRESS MENU\n\0");
  puts("You have chosen a command which requires a value in
                                                   the\0");
  puts("register/counter\n\0");
  puts("What do you want to do next?\n\0");
  puts("\tENTER YOUR BRANCH ADDRESS FIELD\0");
  puts("\t      H  for HELP with this program\0");
  puts("\t      R  to RETURN to a higher level\0");
  gets(branchselect);
} /* procedure branch menu */



PUSHmenu ()


{
  char CONTINUEcommand[4];
  puts(erase_screen);
  puts("\t\t\tREMINDER INFORMATION\n\0");
  puts("You have chosen a PUSH/CONDITIONAL LOAD
                              REGISTER/COUNTER -PUSH\0");
```

```
case 3:  /* oey_fld, 29203 output enable  *
   binary_field(4,choice);
   break;

case 4:  /* source_fld, source field for the 29203.  *
   octal_field(5,choice);
   break;

case 5:  /* dest_fld, destination field.  */
   hex_field(8,choice);
   break;

case 6:  /* function_fld, function field.  */
   hex_field(12,choice);
   break;

case 7:  /* carryin_fld, carry-in mux control for the
                                                2904. */
   dual_field(16,choice);
   break;

case 8:  /* I5I4_fld, bits I05_04, two MSB's  *
         /* of conditional test codes.         *
   dual_field(18,choice);
   break;

case 9:/* I3_I0, bits I03_04 thru I00_04, four LSB's*
       /* of conditional test codes.               *
   hex_field(20,choice);
   break;

case 10: /* bits I05_04 thru I00_04                *
         /* don't know how to use this field yet.  *
   break;

case 11: /* ceu_fld, micro status enable bit       *
   binary_field(24,choice);
   break;

case 12: /* cem_fld, macro status enable bit.      *
   binary_field(25,choice);
   break;

case 13: /* cmden_fld, command enable field        *
   binary_field(26,choice);
   break;

case 14: /* shiften_fld, shift enable field.       *
   binary_field(27,choice);
   break;
```

```
case 15: /* comand_fld, command field.             *
   hex_field(28,choice);
   break;

case 16: /* shift_fld, shift field.                */
   hex_field(28,choice);
   break;

case 17: /* breakpt_fld, breakpoint field.         */
   binary_field(32,choice);
   break;

case 18:  /* notused_fld, this field not used.     */
   binary_field(33,choice);
   break;

case 19:/*msb_br_fld,2 MSB's of branch addressfield */
        /*First test for conflicts by testing      */
        /*docu_word. If no conflicts, finish setting*/
        /*branch address fields with recursive calls*
        /*to field_set with cases 20 and 21.       *

 if (docu_word[19]==0)
    {
    if (docu_word[18]==0)
       {
       docu(field_cnt,no_sub,choice);
       dual_field(34,choice);
       field_set(mid_br_fld,no_sub,++choice);
       field_set(lsb_br_fld,no_sub,++choice);
       }
    else
    if (docu_word[18] == -1)
       {
       puts("Branch Address is already set. \0");
       puts("Do you want to change it?\0");
       gets(scrap);
       switch (*scrap)
             {
        case 'YES':
        case 'yes':
        case 'y':
        case 'Y':
                docu(19,no_sub,choice);
                dual_field(34,choice);
                field_set(mid_br_fld,no_sub,++choice);
                field_set(lsb_br_fld,no_sub,++choice);
             break;
              default:
```

```
                              printf("OKAY--it hasn't been
                                                  changed!\n");
                              break;
                   }
                  }
           else
              puts("Garbage in the docu_word for #19, br.
                                                     add.\0");
           }
        else
           {
puts("Can't use this field for both register desig\0");
puts("nation AND branch address in the same micro-\0");
puts("word.  Right now it's being used to select \0");
puts("register A and register B.\0");
           }
           break;

 case 20: /* mid_br_fld, 4 middle bits of branch
                                            address field */
           hex_field(36,choice);
           break;

 case 21: /* lsb_br_fld, 4 LSB's of branch address
                                                 field.. *
           hex_field(40,choice);
           break;

 case 22: /* rega_fld, specify register A as source */
           hex_field(36,choice);
           break;

 case 23: /* regb_fld, specify register B as source */
           hex_field(40,choice);
           break;

 case 24: /* seq_fld, sequencer code                   */
  /*  This case has been modified to allow changing of
        the seq code after it has already been set.   */

           if(docu_word[23] != 0)
              {
              puts("The sequencer code is already set.\0");
              puts("Do you want to change it?\0");
              gets(scrap);
              switch(*scrap)
                 {
                 case 'YES':
                 case 'yes':
                 case 'Y':
                 case 'y':
```

```
                switch(docu_word[23])
                  {
                  case 2:  /* clear previous branch address */
                     string_erase(34,43);
                     docu_word[18] = 0;
                     string_erase(44,47);
                     docu_word[23] = 0;
                     break;
                  case 4: /* clear br.addr. and cond. test */
                     string_erase(34,43);
                     docu_word[18] =0;
                  case 3: /* clear conditional test codes*/
                   switch(docu_word[7])
                    {
                    case 4:
                    case 3:
                       string_erase(18,23);
                    case 2:
                       string_erase(28,31);
                    case 1:
                       bit_erase(Cmd_En);
                       docu_word[7] = 0;
                       break;
                    }
                  case 1: /* clear sequencer code */
                    string_erase(44,47);
                    docu_word[23] = 0;
                    break;
                  }
               docu(field_cnt,sub_set,choice);
               hex_field(44,choice);
               break;
             default:
                 puts("It hasn't been changed. \0");
                 break;
             } /* end switch */
           } /* end if */
           else
              {
              docu(field_cnt,sub_set,choice);
              hex_field(44,choice);
              }
           break;

      case 25: /* Conditional Tests field--still not
coordinated with shift codes. Check to see if conditional
testing already set.  If yes, erase previous micro_word
entries.  If no, go on to set proper code.     */

        if (docu_word[7] != 0)
          {
```

```
 puts("We got to docu_word[7] not = 0.\0" ;
 switch(docu_word[7])
   {
   case 3: /* erase logical fields 8 & 9 */
   case 4:
      string_erase(18,23);
   case 2: /* erase logical field 15, command_fld */
      string_erase(28,31);
   case 1: /* erase logical field 13, Command_en_fl*/
      bit_erase(Cmd_En);
      docu_word[7] = 0;
      display_docu();
      display_word();
      display_in_hex();
      break;
   default:
      puts("Garbage in sub_set of case 25 in
                                     field_set.\0" ;
      break;
      }   /* end switch  */
   }   /* end if */

      /* Set proper conditional testing bits. */
switch(sub_set)
   {
   case 1:  /* Forced Pass */
      docu(8,1,choice);
      bit_set(Cmd_En);
      break;
   case 2:  /* Forced Fail */
      docu(8,2,choice);
      bit_clear(Cmd_En);
      *scrap = '8';
      field_set(command_fld,no_sub,scrap);
      break;
   case 3:  /* Single level testing */
      bit_clear(Cmd_En);
      *scrap = '9';
      field_set(command_fld,no_sub,scrap);
      docu(8,3,choice);
      *scrap = '0';
      field_set(I5I4_fld,no_sub,scrap);
      field_set(I3_I0_fld,no_sub,choice);
      break;
   case 4:  /* Second Level Testing */
      bit_clear(Cmd_En);
      *scrap = '9';
      field_set(command_fld,no_sub,scrap);
      docu(8,4,choice);
      field_set(I3_I0_fld,no_sub,choice);
      break;
```

```
            default:
            puts("Garbage in sub_set for field_set case
                                                        25.\0");
            break;
            } /*  end switch */
        } /* end switch */
    }  /*  end field_set  */


binary_field(bit_num,choice)

    int  bit_num;
    char *choice;

    {
    if (*choice == '0')
        bit_clear(bit_num);

    else if (*choice == '1')
        bit_set(bit_num);

    else
        bad_choice(choice);
    }

dual_field(bit_num,choice)

    int  bit_num;
    char *choice;

    {
    switch (*choice)

       {
       case '0':
         bit_clear(bit_num);
         bit_clear(bit_num+1);
         break;

       case '1':
         bit_clear(bit_num);
         bit_set(bit_num+1);
         break;

       case '2':
         bit_set(bit_num);
         bit_clear(bit_num+1);
         break;

       case '3':
         bit_set(bit_num);
```

```
          bit_set(bit_num+1);
          break;

        default:
          bad_choice(choice);
          break;
        }
    }

octal_field(bit_num,choice)

    int  bit_num;
    char *choice;

    {
    switch (*choice)
        {
        case '0':
           bit_clear(bit_num);
           bit_clear(bit_num+1);
           bit_clear(bit_num+2);
           break;
        case '1':
           bit_clear(bit_num);
           bit_clear(bit_num+1);
           bit_set(bit_num+2);
           break;
        case '2':
           bit_clear(bit_num);
           bit_set(bit_num+1);
           bit_clear(bit_num+2);
           break;
        case '3':
           bit_clear(bit_num);
           bit_set(bit_num+1);
           bit_set(bit_num+2);
           break;
        case '4':
           bit_set(bit_num);
           bit_clear(bit_num+1);
           bit_clear(bit_num+2);
           break;
        case '5':
           bit_set(bit_num);
           bit_clear(bit_num+1);
           bit_set(bit_num+2);
           break;
        case '6':
           bit_set(bit_num);
           bit_set(bit_num+1);
           bit_clear(bit_num+2);
          break;
       case '7':
          bit_set(bit_num);
          bit_set(bit_num+1);
          bit_set(bit_num+2);
          break;
       default:
          bad_choice(choice);
          break;
       }
    }

hex_field(bit_num,choice)

   int bit_num;
   char *choice;

   {
   switch (*choice)
        {
        case '0':
           bit_clear(bit_num);
           bit_clear(bit_num+1);
           bit_clear(bit_num+2);
           bit_clear(bit_num+3);
           break;

        case '1':
           bit_clear(bit_num);
           bit_clear(bit_num+1);
           bit_clear(bit_num+2);
           bit_set(bit_num+3);
           break;

        case '2':
           bit_clear(bit_num);
           bit_clear(bit_num+1);
           bit_set(bit_num+2);
           bit_clear(bit_num+3);
           break;

        case '3':
           bit_clear(bit_num);
           bit_clear(bit_num+1);
           bit_set(bit_num+2);
           bit_set(bit_num+3);
           break;

        case '4':
           bit_clear(bit_num);
           bit_set(bit_num+1);
```

```
      bit_clear(bit_num+2);
      bit_clear(bit_num+3);
      break;

   case '5':
      bit_clear(bit_num);
      bit_set(bit_num+1);
      bit_clear(bit_num+2);
      bit_set(bit_num+3);
      break;

   case '6':
      bit_clear(bit_num);
      bit_set(bit_num+1);
      bit_set(bit_num+2);
      bit_clear(bit_num+3);
      break;

   case '7':
      bit_clear(bit_num);
      bit_set(bit_num+1);
      bit_set(bit_num+2);
      bit_set(bit_num+3);
      break;

   case '8':
      bit_set(bit_num);
      bit_clear(bit_num+1);
      bit_clear(bit_num+2);
      bit_clear(bit_num+3);
      break;

   case '9':
      bit_set(bit_num);
      bit_clear(bit_num+1);
      bit_clear(bit_num+2);
      bit_set(bit_num+3);
      break;

   case 'a':
   case 'A':
      bit_set(bit_num);
      bit_clear(bit_num+1);
      bit_set(bit_num+2);
      bit_clear(bit_num+3);
      break;

   case 'b':
   case 'B':
      bit_set(bit_num);
      bit_clear(bit_num+1);
      bit_set(bit_num+2);
      bit_set(bit_num+3);
      break;

    case 'c':
    case 'C':
      bit_set(bit_num);
      bit_set(bit_num+1);
      bit_clear(bit_num+2);
      bit_clear(bit_num+3);
      break;

    case 'd':
    case 'D':
      bit_set(bit_num);
      bit_set(bit_num+1);
      bit_clear(bit_num+2);
      bit_set(bit_num+3);
      break;

    case 'e':
    case 'E':
      bit_set(bit_num);
      bit_set(bit_num+1);
      bit_set(bit_num+2);
      bit_clear(bit_num+3);
      break;

    case 'f':
    case 'F':
      bit_set(bit_num);
      bit_set(bit_num+1);
      bit_set(bit_num+2);
      bit_set(bit_num+3);
      break;

    default:
      bad_choice(choice);
      break;

    }
  }



display_word()
    {
    int  i,j;
    printf("     ");
    for ( j=0 ; j<47 ; j=j+16 )
```

```
          for ( i=j ; i < 16+j ; i++ )
               putchar(micro_word[i]);
          putchar(' ');
          putchar(' ');
          }
     putchar('\n');
     }

display_in_hex()
     {
     int  i,j;
     printf("              ");
     for ( j=0; j<47; j=j+16)
          {
          for ( i=j; i < 16+j; i += 4)
               hex_display( &micro_word[i]);
          printf("                    ");
          }
     putchar('\n');
     }

hex_display(pchar)
     char *pchar;
     {
     int  i,value;
     value = 0;
     for ( i=0; i < 4; i++ )
          {
          switch (*(pchar+i))
               {
               case '0':
                    value = 2*value;
                    break;

               case 'X':
               case '1':
                    value = 1 + 2*value;
                    break;


               case '?':
                    putchar('?');
                    return;
                    break;
               }
          }
     printf("%.1x",value);
     }

display_docu()
```

```
    int i;
    putchar('\n');
    for (i=0;i<24;i++)
        printf("%d ",docu_word[i]);
    putchar('\n');
    }

bit_set(i)
     int  i;
     {
                int error;
             error = 0;
          micro_word[i] = '1';
                return(error);
     }


bit_clear(i)
     int  i;
     {
          int  error;
          error = 0;
          micro_word[i] = '0';
          return (error);
     }

bit_erase(i)
     int  i;
     {
          micro_word[i] = 'X';
     }

string_erase(i,j)
     int i,j;
     {
     for(; i<=j;i++ )
        micro_word[i] = 'X';
     }
```

Program Name: Alumake
Purpose: The Makefile used to compile the ALU module and
its submodules. Tne name alumake must be changed to
makefile to be used.

```
test: ALU.o 203.menus.o 2904.supp.o alutils.o
     cc ALU.o 203.menus.o 2904.supp.o alutils.o -o test2
```

Program Name: ALU.c
Purpose: Source code for the ALU module.

```
/* This is the draft of the 29203 section of the microcode
   generation system as of 27 Dec 1984.     */

#include  <stdio.h>
#include  "declare.h"



char cmd_line[80],*pcmd,micro_word[49],*pmwd;
int  KEEPgoing,goback,docu_word[24];

main()

char contin[10];
int i,lim_src,spc_src,rt_shift,left_shift,rama,ramb,helpset;

pcmd = cmd_line;
for (i=0; i<48;i++) micro_word[i] = 'X';
for (i=0; i<24;i++) docu_word[i] = 0;

goback = false;

do
  {
  KEEPgoing = true;
  helpset = false;

  rt_shift = false;
  left_shift = false;
  rama = false;
  ramb = false;

  am29203menu(pcmd);
  switch (*pcmd)
     {
     case 'h':
     case 'H':
       helpset = true;
       KEEPgoing = false;
       puts("The 29203 alu is documented in chapter 5 of the
                                                    AMD\0");
       puts("data book.  There are two types of functions it
                                                    can\0");
       puts("perform, regular functions and special
                                              functions. \0");
       puts("The rest of the decisions you must make are
                                                based on\0");
```

```
        break;

    case 12: /* cem_fld, macro status enable bit */
        binary_field(25,choice);
        docu(12,0,choice);
     break;

    case 13:  /* cmden_fld, command enable field */
        binary_field(26,choice);
        docu(13,0,choice);
        break;

    case 14: /* shiften_fld, shift enable field */
        binary_field(27,choice);
        docu(14,0,choice);
        break;

    case 15: /* command_fld, command field */
       hex_field(28,choice);
       docu(15,0,choice);
       break;

    case 16:  /* shift_fld, shift field */
        hex_field(28,choice);
        docu(16,0,choice);
        break;

    case 17:  /* breakpt_fld, [illegible] */
        binary_field(32,choice);
        docu(17,0,choice);
        break;

    case 18:  /* notused_fld, this field is not used */
        binary_field(33,choice);
        docu(18,0,choice);
        break;

    case 19:  /* msb_br_fld, 2 MSB's of branch address
                                                    field */
        dual_field(34,choice);
        docu(19,0,choice);
        break;

    case 20:  /* mid_br_fld, 4 middle bits of branch
                                            address field */
        hex_field(36,choice);
        docu(20,0,choice);
        break;

    case 21:  /* lsb_br_fld, 4 LSB's of branch address
                                                    field */
```

```
case 2:  /* ien_fld, 29203 instruction enable */
   binary_field(3,choice);
   docu(2,0,choice);
   break;

case 3: /* oey_fld, 29203 output enable */
   binary_field(4,choice);
   docu(3,0,choice);
   break;

case 4: /* source_fld, source field for the 29203.  */
   octal_field(5,choice);
   docu(4,0,choice);
   break;

case 5: /* dest_fld, destination field. */
   hex_field(8,choice);
   docu(5,0,choice);
   break;

case 6: /* function_fld, function field   */
   hex_field(12,choice);
   docu(6,0,choice);
   break;

case 7: /* carryin_fld, carry-in mux control for the
                                             2904  */
   dual_field(16,choice);
   docu(7,0,choice);
   break;

case 8:/*I5I4_fld, bits I05_04 and I04_04, two MSB's*/
       /* of conditional test codes.                */
    dual_field(18,choice);
    docu(8,0,choice);
    break;

case 9:/*I3_I0, bits I03_04 thru I00_04, four LSB's */
       /* of conditional test codes.                */
    hex_field(20,choice);
    docu(9,0,choice);
    break;

 case 10: /* bits I05_04 thru I00_04  */
/*  don't know how to use this field yet */
    docu(10,0,choice);
    break;

case 11: /* ceu_fld, micro status enable bit */
    binary_field(24,choice);
    docu(11,0,choice);
```

Program Name: Alutils.c
Purpose: Source code for Utilities module which is compiled
with the ALU module.

```
#include      <stdio.h>
#include      "declare.h"
#include      "extern.h"


docu(field,sub_set,choice)

int  field,sub_set;
char *choice;
 {
 puts("docu called\0");
 sleep(2);
 }


bad_choice(choice)

char *choice;
  {
  puts("bad_choice called.\0");
  putchar(*choice);
  sleep (2);
  }



field_set(field_cnt,sub_set,choice)

  int  field_cnt,sub_set;
  char *choice;

   *   There are 13 defined fields in the 29203 eval board
microword, and several of them have multiple definitions.
In this routine, we accept a pointer to the field and to the
definition of the subset, and a pointer to a character which
represents the actual choice. We generate a data structure
which holds the choice and the actual bit pattern in the
microword.      */

  {
  switch (field_cnt)
      {
      case 1: /* regsel_fld, register address source   */
         octal_field(0,choice);
         docu(1,0,choice);
         break;
      RAMBmenu(cmd_line);
      switch (*cmd_line)
        {
          case 'H':
          case 'h':
              puts("This menu describes the register
                                         selections for\0");
              puts("the ALU.  They are documented on\0");
              puts("page 5-XXX of the AMD data book.\0");
              puts("type a C to continue.\0");
              gets(contin);
              break;

          case 'R':
          case 'r':
              break;

          case '0':
          case '1':
          case '2':
          case '3':
          case '4':
          case '5':
          case '6':
          case '7':
          case '8':
          case '9':
          case 'a':
          case 'A':
          case 'b':
          case 'B':
          case 'c':
          case 'C':
          case 'd':
          case 'D':
          case 'e':
          case 'E':
          case 'f':
          case 'F':
              field_set(regb_fld,0,cmd_line);
              docu(regb_fld,0,cmd_line);
              break;
          }
      }
     }
     while(helpset || KEEPgoing);
     exit();
 }  /* the end of 29203master  */
```

```
    }

if (rama == true && (KEEPgoing))
  {
  RAMAmenu(cmd_line);
  switch (*cmd_line)
    {
      case 'H':
      case 'h':
          puts("This menu describes the register
                                     selections for\0");
          puts("the ALU.  They are documented on\0");
          puts("page 5-XXX of the AMD data book.\0");
          puts("type a C to continue.\0");
          gets(contin);
          break;

      case 'R':
      case 'r':
          break;

      case '0':
      case '1':
      case '2':
      case '3':
      case '4':
      case '5':
      case '6':
      case '7':
      case '8':
      case '9':
      case 'a':
      case 'A':
      case 'b':
      case 'B':
      case 'c':
      case 'C':
      case 'd':
      case 'D':
      case 'e':
      case 'E':
      case 'f':
      case 'F':
          field_set(rega_fld,0,cmd_line);
          docu(rega_fld,0,cmd_line);
          break;
    }
  }

if (ramb == true && (KEEPgoing))
  {
```

```
                    field_set(dest_fld,0,cmd_line);
                    docu(dest_fld,0,cmd_line);
                    /* Need to add things here!!!*
                    break;
                default:
                    helpset = true;
                    puts("Illegal entry, type a C to
                                                repeat.\0");
                    gets(contin);
                    break;
                }
            } /* end while  */
            if(!KEEPgoing)  helpset = true;
          }

      if (rt_shift == true && (KEEPgoing))
        {
        shift2_menu();
        gets(cmd_line);
        if (shift_set(cmd_line)) goto dest_start;
        bit_clear(27);
        }

     if (left_shift == true && (KEEPgoing))
        {
        shift3_menu();
        gets(cmd_line);
        if (shift_set(cmd_line)) goto dest_start;
        bit_clear(27); /* the shift enable for the 2904. *
        }
   if (KEEPgoing)
     {
     ENABLEmenu(cmd_line,contin);
     if (*cmd_line == 'Y' || *cmd_line == 'y')
      *cmd_line = '0';
     else
      *cmd_line = '1';
     field_set(ien_fld,0,cmd_line);

     if (*contin == 'Y' || *contin == 'y')
      *contin = '0';
     else
      *contin = '1';
     field_set(oey_fld,0,contin);
     }

     if (rama == true || ramb == true && (KEEPgoing))
       {
       REGISTERmenu(cmd_line);
       field_set(reg_src,0,cmd_line);
       docu(reg_src,0,cmd_line);
```

```
        puts("the ALU result.  They are documented
                                              on\0");
        puts("page 5-XXX of the AMD data
                                         book.\0");
        puts("type a C to continue.\0");
        gets(contin);
        break;

    case 'R':
    case 'r':
        KEEPgoing = false;
        break;

    case '0':
    case '1':
    case '2':
    case '3':
    case '5':
        rt_shift = true;
        field_set(dest_fld,0,cmd_line);
        docu(dest_fld,0,cmd_line);
        break;
    case '8':
    case '9':
    case 'a':
    case 'A':
    case 'b':
    case 'B':
    case 'd':
    case 'D':
        left_shift = true;
        field_set(dest_fld,0,cmd_line);
        docu(dest_fld,0,cmd_line);
        break;
    case '4':
    case '6':
    case '7':
    case 'c':
    case 'C':
    case 'e':
    case 'E':
    case 'f':
    case 'F':
        field_set(dest_fld,0,cmd_line);
        docu(dest_fld,0,cmd_line);
        break;
    case 'M':
    case 'm':
    case 'I':
    case 'i':
        *cmd_line = '0';
```

```
            puts("The special functions require that
                                              IO=0\0");
            puts("Therefore the sources are limited to
                                                the\0");
            puts("set on this menu.  The sources are
                                          described\0");
            puts("on page 5-XXX of the AMD data
                                              book.\0");
            puts("type a C to continue.\0");
            gets(contin);
            break;

         case 'R':
         case 'r':
            break;

         case '0':
            ramb = true;
         case '1':
            rama = true;
            field_set(src_fld,0,cmd_line);
            docu(src_fld,0,cmd_line);
            break;
         case '4':
            ramb = true;
         case '5':
            field_set(src_fld,0,cmd_line);
            docu(src_fld,0,cmd_line);
            break;
         default:
           puts("Illegal entry, type a C to repeat.\0");
           gets(contin);
           break;
      }
    }

if (spc_src != true && (KEEPgoing))
      {
      helpset = true;
      while(helpset)
          {
          helpset = false;
          dest_start:  DESTINATIONmenu(cmd_line);
          switch(*cmd_line)
            {
            case 'H':
            case 'h':
              helpset = true;
              puts("This menu describes the destinations
                                                for\0");
```

```
        while(helpset)
         {
         helpset = false;
         srcl_start:  SOURCElmenu(cmd_line);
         switch (*cmd_line)
          {
          case 'H':
          case 'h':
              helpset = true;
           puts("Because of the use of the IO bit to
                                          indicate\0");
           puts("special functions, there are limits to
                                              the \0");
           puts("sources for ALU operands.  These limits
                                               are\0");
           puts("described on page 5-XXX of the AMD data
                                             book.\0");
           puts("type a C to continue.\0");
           gets(contin);
           break;

    case 'R':
    case 'r':
           KEEPgoing = false;
           break;

    case '2':
           rama = true;
    case '6':
           field_set(src_fld,0,cmd_line);
           docu(src_fld,0,cmd_line);
           break;

    default:
           helpset = true;
           puts("Illegal entry, type a C to repeat.\0");
           gets(contin);
           break;

     }
   } /* end While  */
   if(!KEEPgoing) helpset = true;
  }

 else if (KEEPgoing)
    {
    src2_start:  SOURCE2menu(cmd_line);
    switch(*cmd_line)
       {
       case 'H':
       case 'h':
```

```
    helpset = true;
    while(helpset)
      {
      helpset = false;
      srcstart:  SOURCEmenu(cmd_line);
  switch (*cmd_line)
    {
    case 'R':
    case 'r':
       KEEPgoing = false;
       break;

    case 'H':
    case 'h':
             helpset = true;
       puts ("Sources for ALU operations are
                                         described\0");
       puts ("on page 5-XXX of the AMD data book.
                                                \0");
       puts ("Type a C to continue.\0");
       gets (contin);
       break;

    case '0':
       ramb = true;
    case '1':
    case '2':
       rama = true;
       field_set(src_fld,0,cmd_line);
       docu(src_fld,0,cmd_line);
       break;
    case '4':
       ramb = true;
    case '5':
    case '6':
       field_set(src_fld,0,cmd_line);
       docu(src_fld,0,cmd_line);
       break;
    default:
       helpset = true;
       puts("Illegal entry, type a C to repeat.\0");
       gets(contin);
       break;
    }
  } /* end while */
  if (!KEEPgoing)  helpset = true;
}

else if (lim_src == true && (KEEPgoing))
{
    helpset = true;
```

```
         field_set(dest_fld,0,pcmd);
         *pcmd = '0';
         field_set(func_fld,0,pcmd);
         break;
      case 'M':
      case 'm':
         spc_src = true;
         docu(dest_fld,0,pcmd);
         *pcmd = '1';
         field_set(dest_fld,0,pcmd);
         *pcmd = '8';
         field_set(func_fld,0,pcmd);
         break;
      case 'Z':
      case 'z':
         spc_src = true;
         docu(dest_fld,0,pcmd);
         *pcmd = '9';
         field_set(dest_fld,0,pcmd);
         *pcmd = '8';
         field_set(func_fld,0,pcmd);
         break;

      default:
           helpset = true;
           puts("Illegal entry.\0");
           puts("Press enter to continue.\0");
           gets(contin);
           break;
      }
   } /* end while */
   while(helpset && KEEPgoing);
   if(!KEEPgoing)
      {
      helpset = true;
      }
      break;

   default:
        helpset = true;
        KEEPgoing = false;
        puts("Illegal entry.\0");
        puts("Press enter to continue.\0");
        gets(contin);
        break;
   } /* end main switch */


   if (lim_src != true && spc_src != true
                                      && (KEEPgoing))
```

```
          }
        break;

        case 'S':
        case 's':
          do
            {
            spc_src = false;
            helpset = false;
            SPECIALmenu(pcmd);
            switch(*pcmd)
            {
            case 'H':
            case 'h':
               helpset = true;
               puts("These special functions are described on
                                                page 0");
               puts("5-XXX of the AMD data book.\0");
               puts("Press enter to continue. \0");
               gets(contin);
               break;

            case 'R':
            case 'r':
               KEEPgoing = false;
               break;

            case '0':
            case '1':
            case '2':
            case '3':
            case '4':
            case '5':
            case '6':
            case '7':
            case '8':
            case '9':
            case 'a':
            case 'A':
            case 'b':
            case 'B':
            case 'c':
            case 'C':
            case 'd':
            case 'D':
            case 'e':
            case 'E':
            case 'f':
            case 'F':
               spc_src = true;
               docu(dest_fld,0,pcmd);
        case '6':  /* codes 6 and 7 require limits on the
                                source field.*/
        case '7':
          lim_src = true;
        case '1':
        case '2':
        case '3':
        case '4':
        case '5':
          field_set(func_fld,0,pcmd);
          docu(func_fld,0,pcmd);
          carryin_menu();
          gets(cmd_line);
          carry_set(cmd_line);
          break;

        case '0':  /*F=high  */
        case '8':  /*F=low   */
             /* codes 0 and 8 require limits on the source
                field.*/
          lim_src = true;

        case '9':
        case 'A':
        case 'a':
        case 'B':
        case 'b':
        case 'C':
        case 'c':
        case 'D':
        case 'd':
        case 'E':
        case 'e':
        case 'F':
        case 'f':
            field_set(func_fld,0,pcmd);
            docu(func_fld,0,pcmd);
            break;

        default:
            helpset = true;
            puts("Invalid input.\0");
            puts("Press enter to continue.\0");
            gets(contin);
            break;
        }
      } /* end while */
    while(helpset && KEEPgoing);
    if(!KEEPgoing)
       {
       helpset = true;
    puts("which of these you chose.\0");
    puts("Press enter to continue.\0");
    gets(contin);
    break;

  case 'r':
  case 'R':
   puts("Do you really want to return to mastermenu? 0");
    gets(contin);
    switch(*contin)
      {
      case 'YES':
      case 'yes':
      case 'Y':
      case 'y':
          KEEPgoing = false;
          break;
      default:
          KEEPgoing = false;
          helpset = true;
          break;
      }
    break;

  case 'b':
  case 'B':
    do

      helpset = false;
      FUNCTIONmenu(pcmd);
      lim_src = false;
      spc_src = false;
      switch(*pcmd)

       case 'h':
       case 'H':
         helpset = true;
         puts("The 29203 ALU functions are described on
                                                  page 5-XXX\0");
         puts("of the AMD data book. \0");
         puts("Press enter to enter.\0");
         gets(contin);
         break;

       case 'r':
       case 'R':
         KEEPgoing = false;
         break;
```

```
                hex_field(40,choice);
                docu(21,0,choice);
                break;

            case 22:  /* rega_fld, specify register A as source */
                hex_field(36,choice);
                docu(22,0,choice);
                break;

            case 23:  /* regb_fld, specify register B as source */
                hex_field(40,choice);
                docu(23,0,choice);
                break;

            case 24:  /* seq_fld, sequencer code */
                hex_field(44,choice);
                docu(24,0,choice);
                break;
            }

        }


    binary_field(bit_num,choice)

        int bit_num;
        char *choice;

        {
        if (*choice == '0')
            bit_clear(bit_num);

        else if (*choice == '1')
            bit_set(bit_num);

        else
            bad_choice(choice);
        }

    dual_field(bit_num,choice)

        int bit_num;
        char *choice;

        {
        switch (*choice)

            {
            case '0':
              bit_clear(bit_num);
              bit_clear(bit_num+1);
```

```
       break;

     case '1':
       bit_clear(bit_num);
       bit_set(bit_num+1);
       break;

     case '2':
       bit_set(bit_num);
       bit_clear(bit_num+1);
       break;

     case '3':
       bit_set(bit_num);
       bit_set(bit_num+1);
       break;

     default:
       bad_choice(choice);
       break;
     }
  }

octal_field(bit_num,choice)

  int bit_num;
  char *choice;

  {
  switch (*choice)
      {
      case '0':
        bit_clear(bit_num);
        bit_clear(bit_num+1);
        bit_clear(bit_num+2);
        break;
      case '1':
        bit_clear(bit_num);
        bit_clear(bit_num+1);
        bit_set(bit_num+2);
        break;
      case '2':
        bit_clear(bit_num);
        bit_set(bit_num+1);
        bit_clear(bit_num+2);
        break;
      case '3':
        bit_clear(bit_num);
        bit_set(bit_num+1);
        bit_set(bit_num+2);
        break;
```

```
         case '4':
           bit_set(bit_num);
           bit_clear(bit_num+1);
           bit_clear(bit_num+2);
           break;
         case '5':
           bit_set(bit_num);
           bit_clear(bit_num+1);
           bit_set(bit_num+2);
           break;
         case '6':
           bit_set(bit_num);
           bit_set(bit_num+1);
           bit_clear(bit_num+2);
           break;
         case '7':
           bit_set(bit_num);
           bit_set(bit_num+1);
           bit_set(bit_num+2);
           break;
         default:
           bad_choice(choice);
           break;
         }
       }

    hex_field(bit_num,choice)

      int bit_num;
      char *choice;

      {
      switch (*choice)
         {
         case '0':
           bit_clear(bit_num);
           bit_clear(bit_num+1);
           bit_clear(bit_num+2);
           bit_clear(bit_num+3);
           break;

         case '1':
           bit_clear(bit_num);
           bit_clear(bit_num+1);
           bit_clear(bit_num+2);
           bit_set(bit_num+3);
           break;

         case '2':
           bit_clear(bit_num);
           bit_clear(bit_num+1);
    bit_set(bit_num+2);
    bit_clear(bit_num+3);
    break;

  case '3':
    bit_clear(bit_num);
    bit_clear(bit_num+1);
    bit_set(bit_num+2);
    bit_set(bit_num+3);
    break;

  case '4':
    bit_clear(bit_num);
    bit_set(bit_num+1);
    bit_clear(bit_num+2);
    bit_clear(bit_num+3);
  break;

  case '5':
    bit_clear(bit_num);
    bit_set(bit_num+1);
    bit_clear(bit_num+2);
    bit_set(bit_num+3);
    break;

  case '6':
    bit_clear(bit_num);
    bit_set(bit_num+1);
    bit_set(bit_num+2);
    bit_clear(bit_num+3);
    break;

  case '7':
    bit_clear(bit_num);
    bit_set(bit_num+1);
    bit_set(bit_num+2);
    bit_set(bit_num+3);
    break;

  case '8':
    bit_set(bit_num);
    bit_clear(bit_num+1);
    bit_clear(bit_num+2);
    bit_clear(bit_num+3);
    break;

  case '9':
    bit_set(bit_num);
    bit_clear(bit_num+1);
    bit_clear(bit_num+2);
    bit_set(bit_num+3);
    break;

  case 'A':
  case 'a':
    bit_set(bit_num);
    bit_clear(bit_num+1);
    bit_set(bit_num+2);
    bit_clear(bit_num+3);
    break;

  case 'B':
  case 'b':
    bit_set(bit_num);
    bit_clear(bit_num+1);
    bit_set(bit_num+2);
    bit_set(bit_num+3);
    break;

  case 'C':
  case 'c':
    bit_set(bit_num);
    bit_set(bit_num+1);
    bit_clear(bit_num+2);
    bit_clear(bit_num+3);
    break;

  case 'D':
  case 'd':
    bit_set(bit_num);
    bit_set(bit_num+1);
    bit_clear(bit_num+2);
    bit_set(bit_num+3);
    break;

  case 'E':
  case 'e':
    bit_set(bit_num);
    bit_set(bit_num+1);
    bit_set(bit_num+2);
    bit_clear(bit_num+3);
    break;

  case 'F':
  case 'f':
    bit_set(bit_num);
    bit_set(bit_num+1);
    bit_set(bit_num+2);
    bit_set(bit_num+3);
    break;

  default:
          bad_choice(choice);
          break;

        }
    }

display_word()
      {
      int  i,j;
      printf("     ");
      for ( j=0 ; j<47 ; j=j+16 )
            {
            for ( i=j ; i < 16+j ; i++)
                  putchar(micro_word[i]);
            putchar(' ');
            putchar(' ');
            }
      putchar('\n');
      }

display_in_hex()
      {
      int  i,j;
      printf("                ");
      for ( j=0; j<47; j=j+16)
            {
            for ( i=j; i < 16+j; i += 4)
                  hex_display( &micro_word[i]);
            printf("                ");
            }
      putchar('\n');
      }

hex_display(pchar)
      char *pchar;
      {
      int  i,value;
      value = 0;
      for ( i=0; i < 4; i++ )
            {
            switch (*(pchar+i))
                  {
                  case '0':
                        value = 2*value;
                        break;

                  case 'X':
                  case '1':
                        value = 1 + 2*value;
                        break;


                    case '?':
                        putchar('?');
                        return;
                        break;
                    }
                }
            printf("%.lx",value);
            }


        bit_set(i)
            int  i;
            {
                int  error;
                error = 0;
                switch (micro_word[i])
                    {
                    case 'X':  /* Don't care, therefore set it'*
                        micro_word[i] = '1';
                        break;

                    case '1': /* Already set, therefore OK. *
                        break;

                    case '0': /* Cleared, therefore an error.  */
                        micro_word[i] = '?';
                        error = 1;
                        break;

                    case '?': /* Already in the error state.  */
                        break;

                    default:  /* Garbage in the microword! */
                        puts ("Garbage in the
                                            microword!!!!\n\0");
                        sleep(2);
                        break;

                    }
                return (error);
            }


        bit_clear(i)
            int  i;
            {
                int  error;
```

```
          error = 0;
          switch (micro_word[i])
               {
               case 'X':/* Don't care, therefore clear it!*/
                    micro_word[i] = '0';
                    break;

               case '0': /* Already clear, therefore OK. */
                    break;

               case '1': /* set, therefore an error.  */
                    micro_word[i] = '?';
                    error = 1;
                    break;

               case '?': /* Already in the error state.  */
                    break;

               default:  /* Garbage in the microword! */
                    puts ("Garbage in the
                                        microword!!!!\n\0");
                    sleep(2);
                    break;

               }
          return (error);
     }

bit_erase(i)
     int  i;
     {
          micro_word[i] = 'X';
     }
```

Program Name: 203.menus.c
Purpose: Source code for the menus used in the ALU module.

```
#define         erase_screen    "\033[2J\033[0:0H"
#include  <stdio.h>


/*******************************************************/
/*                                                     */
/*              AM29203 MENU PROCEDURES                */
/*                                                     */
/*******************************************************/

am29203menu(am29203select)

char *am29203select;

{
  puts(erase_screen);
  puts("\t\t\tMASTER AM29203 ALU MENU\n\n\0");
  display_word();
  display_in_hex();
  puts("The X s indicate bits which are not yet defined\0");
  puts("The defaults for the AM29203 ALU are:\0");
  puts("\tRegister Address Select - bits 47-45 - A,B
                                     pipeline = 111\0");
  puts("\tInstruction Enable - bit 44 - Disable = 1\0");
  puts("\tOutput Enable - bit 43 - Disable = 1\0");
  puts("\tSource - bits 42-40 - DAQ = 111\0");
  puts("\tDestination - bits 39-36 - YBUS = 1111\0");
  puts("\tALU Function - bits 35-32 - OR = 1111\n\0");
  puts("What do you want to do next?\0");
  puts("\ttype a  B  to choose ALU FUNCTIONS\0");
  puts("\t\tS  to choose SPECIAL FUNCTIONS\0");
  puts("\t\tH  for HELP with this program\0");
  puts("\t\tR  to RETURN to higher level\0");
  gets(am29203select);
} /* procedure am29203menu */



FUNCTIONmenu(FUNCTIONselect)

char *FUNCTIONselect;

{
  puts(erase_screen);
  puts("\t\tAM29203 ALU BASIC FUNCTION SELECT\n\0");
  puts("Enter the value corresponding to the function you
                                     wish to perform\0");
  puts("\t\t0  F = High\0");
  puts("\t\t1  F = S - R - 1 + Carry In\0");
  puts("\t\t2  F = R - S - 1 + Carry In\0");
  puts("\t\t3  R + S + Carry In\0");
  puts("\t\t4  S + Carry In\0");
  puts("\t\t5  (NOT S) + Carry In\0");
  puts("\t\t6  R + Carry In\0");
  puts("\t\t7  F = (NOT R) + Carry In\0");
  puts("\t\t8  F = Low\0");
  puts("\t\t9  F = (NOT R) AND S\0");
  puts("\t\tA  F = R EXCLUSIVE OR S\0");
  puts("\t\tB  F = R EXCLUSIVE OR S\0");
  puts("\t\tC  F = R AND S\0");
  puts("\t\tD  F = R NOR S\0");
  puts("\t\tE  F = R NAND S\0");
  puts("\t\tF  F = R OR S\0");
  puts("\t\tH  for HELP with this program\0");
  puts("\t\tR  to RETURN to higher level\0");
  gets(FUNCTIONselect);
} /* procedure FUNCTIONselect */



SPECIALmenu(SPECIALselect)

{
puts(erase_screen);
puts("\t\tAM29203 ALU SPECIAL FUNCTION MENU\n\0");
puts("Enter the value corresponding to the function you wish
                                                  to perform\0");
puts("        0  Unsigned multiply\0");
puts("        1  BCD to Binary Conversion\0");
puts("        M  Multiprecision BCD to Binary Conversion\0");
puts("        2  Two's Complement Multiply\0");
puts("        3  Decrement by 1 or 2\0");
puts("        4  Increment by 1 or 2\0");
puts("        5  Sign/Magnitude to Two's Complement
                                                  Conversion\0");
puts("        6  Two's Complement Multiply\0");
puts("        7  BCD Divide by 2\0");
puts("        8  Single Length Mormalize\0");
puts("        9  Binary to BCD Conversion\0");
puts("        Z  Multiprecision Binary to BCD Conversion\0");
puts("        A  Double Length Normalize; First Division\0");
puts("        B  BCD  Add\0");
puts("        C  Two's Complement Divide\0");
puts("        D  BCD Subtract  F = R - S - 1 + Carry In
                                                        BCD\0");
puts("        E  Two's Complement Divide Correction and
                                                   Remainder\0");
```

```
puts("        F  BCD Subtract  F = S - R - 1 + Carry In
                                                  BCD\0");
puts("        H  for HELP with this menu\0");
puts("        R  to RETURN to higher level\0");
gets(SPECIALselect);

} /* procedure SPECIALselect */



SOURCEmenu(SOURCEselect)

char *SOURCEselect;

{
  puts(erase_screen);
  puts("\t\t\tAM29203 ALU SOURCE MENU\n\0");
  puts("The source control default is DAQ\n\0");
  puts("             Operand R        Operand S    Mnemonic\0");
  puts("Enter a  0  RAMA            RAMB           RAMAB\0");
  puts("         1  RAMA            Direct B       RAMADB\0");
  puts("         2  RAMA            Q Register     RAMAQ\0");
  puts("         4  Direct A        RAMB           DARAMB\0");
  puts("         5  Direct A        DirectB        DADB\0");
  puts("         6  Direct A        Q Register     DAQ\0");
  puts("         I      Instruction Register\0");
  puts("         P      Pipeline Register\0");
  puts("         H      for H with this program\0");
  puts("         R      to RETURN to higher level\0");
  gets(SOURCEselect);
} /* procedure SOURCEmenu */


SOURCE1menu(SOURCE1select)

char *SOURCE1select;

{
  puts(erase_screen);
  puts("\t\t\tAM29203 ALU SOURCE MENU\n\0");
  puts("You have chosen one of the following AM29203 ALU
                                              functions:\0");
  puts("\tF = High\0");
  puts("\tF = R + Carry In\0");
  puts("\tF + (NOT R) + Carry In\0");
  puts("\tF = LOW\n\0");
  puts("For these functions, the only allowed AM29203 ALU
                                          Sources are:\n\0");
  puts("\tOperand R       Operand S       Mnemonic\n\0");
  puts("\tRAMA            Q Register      RAMAQ\0");
  puts("\tDirect A        Q Register      DAQ\n\0");
```

```
   puts("Type a  2  for RAMAQ\0");
   puts(" \t6  for DAQ\0");
   puts(" \tH  for HELP with this program\0");
   puts(" \tR  to RETURN to a higher level\0");
   gets(SOURCE1select);
} /* procedure SOURCE1menu */


SOURCE2menu(SOURCE2select)

char *SOURCE2select;
{

   puts(erase_screen);
   puts("\t\t\tAM29203 ALU SOURCE SELECT\n\0");
   puts("        You have chosen an AM29203 ALU Special
                                                  Function\n\0");
   puts("        What sources do you want to use\n\0");
   puts("                Operand R       Operand S   Mnemonic\0");
   puts("    Enter a  0  RAMA A            RAM B       RAMAB\0");
   puts("             1  RAM A             DIRECT B    RAMADB\0");
   puts("             4  DIRECT A          RAM A       DARAMB\0");
   puts("             5  DIRECT A          DIRECT B     DADB\0");
   puts("             H  for HELP with this menu\0");
   puts("             R  to RETURN to a higher level\0");
   gets(SOURCE2select);

} /* procedure SOURCE2menu */


DESTINATIONmenu(DESTINATIONselect)

char *DESTINATIONselect;
{

   puts(erase_screen);
   puts("\t\t\tAM29203 ALU DESTINATION MENU\n\0");
   puts("      Enter the value corresponding to the
                                    destination you desire\0");
   puts("    0  RAMDA -  F to RAM, Arithmetic Down Shift \0");
   puts("    1  RAMDL -  F to RAM, Logical Down Shift\0");
   puts("    2  RAMQDA - Double Precision Arithmetic Down
                                                       Shift\0");
   puts("    3  RAMQDL - Double Precision Logical Down
                                                       Shift\0");
   puts("    4  RAM - F to RAM with parity\0");
   puts("    5  QD - F to Y, Down Shift Q\0");
   puts("    6  LOADQ - F to Q with parity\0");
   puts("    7  RAMQ - F to RAM with parity\0");
```

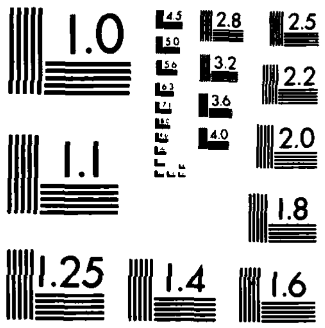
```
 puts("   8  RAMUPA - F to RAM, Arithmetic Up Shift\0");
 puts("   9  RAMUPL - F to RAM, Logical Up Shift\0");
 puts("   A  RAMQUPA - Double Precision Arithmetic Up
                                                Shift\0");
 puts("   B  RAMQUPL - Double Procision Logical Up
                                                Shift\0");
 puts("   C             -  F to Y only\0");
 puts("   D              - F to Y, Up SHift Q\0");
 puts("   E  SIGNEXT - SIO0 to Y(i)\0");
 puts("   F  RAMEXT - F to Y, Sign extend LSB\0");
 puts("   I  Instruction Register\0");
 puts("   M  Main Memory\0");
 puts("   H  for HELP with this program\0");
 puts("   R  to RETURN to higher level\0");
 gets(DESTINATIONselect);
  * procedure DESTINATIONmenu */


EGISTERmenu(REGISTERselect)

har *REGISTERselect;


 puts(erase_screen);
 puts("\t\t\tAM29203 ALU REGISTER ADDRESS MENU\n\0");
 puts("       The default source selection is Source A -
                                          pipeline,\0");
 puts("       Source B - pipeline, Destination C -
                                          pipeline\n\0");
 puts("       Enter the value corresponding to the register
                                             address\0");
 puts("you desire\0");
 puts("    Source A        Source B        Destination C\0");
 puts(" 0  Pipeline         Pipeline        Pipeline\0");
 puts(" 1  Instruction      Pipeline        Pipeline\0");
 puts(" 2  Pipeline         Instruction     Pipeline\0");
 puts(" 3  Instruction      Instruction     Pipeline\0");
 puts(" 4  Pipeline         Pipeline        Instruction\0");
 puts(" 5  Instruction      Pipeline        Instruction\0");
 puts(" 6  Pipeline         Instruction     Instruction\0");
 puts(" 7  Instruction      Instruction     Instruction\0");
 gets(REGISTERselect);
  * procedure REGISTER menu */


IRECTmenu(constant1,constant2)

har *constant1,*constant2;


 puts (erase_screen);
```

```
  puts("\t\t\tAM29203 ALU DIRECT SOURCE MENU\n\0");
  puts("You have chosen an ALU function, source, or
                                               destination which \0");
  puts("        requires a constant or Ra and Rb to be
                                                 entered into the \0");
  puts("        the branch address field of the
                                             microinstruction\n\0");
  puts("        This constant or the RAM register
                                               designations is 1 \0");
  puts(" byte in length\0");
  puts("      Please enter the constant or the RAM register
                                                     designation \0");
  puts("      Enter a  H  for HELP with this program \0");
  puts("      Enter a  R  to RETURN to a higher level\0");
  gets(constant1,constant2);
  /* procedure DIRECT menu */


:NABLEmenu(INSTRUCTIONselect,OUTPUTselect)

:har *INSTRUCTIONselect,*OUTPUTselect;


  puts(erase_screen);
  puts("\t\tAM29203 ALU INSTRUCTION AND OUTPUT ENABLE
                                                         MENU\n\0");
  puts("        Do you want the ALU results to appear on the
                                                        Y bus? \0");
  puts("           Type an  Y  for YES\0");
  puts("           Type a   N  for NO\n\n\n\0");
  gets(OUTPUTselect);
  puts("        Do you want to change the contents of any ALU
                                                     register\0");
  puts("        during this ALU operation?\n\0");
  puts("           Type an  Y  for YES\0");
  puts("           Type an  N  for NO\0");
  gets(INSTRUCTIONselect);
  /* procedure ENABLEmenu */



RAMAmenu(RAMAselect)

:har *RAMAselect;


  puts(erase_screen);
  puts("\t\tAM29203 ALU RAM A REGISTER SELECT\n\0");
  puts("\t         Enter the value corresponding to the RAM A
                                                    Register \0");
  puts("\tyou wish to select\n\0");
```

AD-A155 164   EVALUATION AND IMPLEMENTATION OF A FUNCTIONAL   3/3
MICROPROGRAM GENERATOR(U) NAVAL POSTGRADUATE SCHOOL
MONTEREY CA D B STILTNER DEC 84

UNCLASSIFIED   F/G 9/2   NL

END
FILMED
DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

```
  puts("\t        0  RAMA A Register 0\0");
  puts("\t        1  RAMA A Register 1\0");
  puts("\t        2  RAMA A Register 2\0");
  puts("\t        3  RAMA A Register 3\0");
  puts("\t        4  RAMA A Register 4\0");
  puts("\t        5  RAMA A Register 5\0");
  puts("\t        6  RAMA A Register 6\0");
  puts("\t        7  RAMA A Register 7\0");
  puts("\t        8  RAMA A Register 8\0");
  puts("\t        9  RAMA A Register 9\0");
  puts("\t        A  RAMA A Register A\0");
  puts("\t        B  RAMA A Register B\0");
  puts("\t        C  RAMA A Register C\0");
  puts("\t        D  RAMA A Register D\0");
  puts("\t        E  RAMA A Register E\0");
  puts("\t        F  RAMA A Register F\0");
  puts("\t        H  for HELP with this menu\0");
  puts("\t        R  to RETURN to a higher level\0");
  gets(RAMAselect);
} /* procedure RAMAmenu */


RAMBmenu(RAMBselect)

char *RAMBselect;

{
  puts(erase_screen);
  puts("\t\t\tAM29203 ALU RAM B REGISTER SELECT\n\0");
  puts("\t        Enter the value corresponding to the RAM B
                                                 Register \0");
  puts("\tyou wish to select\n\0");
  puts("\t        0  RAM B Register 0\0");
  puts("\t        1  RAM B Register 1\0");
  puts("\t        2  RAM B Register 2\0");
  puts("\t        3  RAM B Register 3\0");
  puts("\t        4  RAM B Register 4\0");
  puts("\t        5  RAM B Register 5\0");
  puts("\t        6  RAM B Register 6\0");
  puts("\t        7  RAM B Register 7\0");
  puts("\t        8  RAM B Register 8\0");
  puts("\t        9  RAM B Register 9\0");
  puts("\t        A  RAM B Register A\0");
  puts("\t        B  RAM B Register B\0");
  puts("\t        C  RAM C Register C\0");
  puts("\t        D  RAM B Register D\0");
  puts("\t        E  RAM B Register E\0");
  puts("\t        F  RAM B Register F\0");
  puts("\t        H  for HELP with this menu\0");
  puts("\t        R  to RETURN to a higher level\0");
```

```
    gets(RAMBselect);
} /* procedure RAMBmenu */
```

Program Name: 2904.supp.c
Purpose: Source code for the utilities used with the Alu module.

```
/*      This version of the 2904 code is included to allow
the testing of other modules.  The utilities have been
removed, as has the main program portion.  13 July 1984.  */

/*   This program will aid the 2900 system designer in the
programming of a 2904 "glue" chip.    */

/*   This version is not as current as the utilities used
with the Sequencer module.  27 Dec. 1984    */

 /*  This source is set up to work with the vt100 from
VAX/unix.  */

#include  <stdio.h>
#define   erase_screen   "\033[2J\033[0;0H"  /* vt100 erase
                                      screen and home cursor  */
#define   TRUE      1
#define   FALSE     0

     /* These defines relate to the 29203ET board, and
should be contained in a routine to initialize them, rather
than as defines. */

#define        I12_04    16
#define        I11_04    17
#define        I10_04    8
#define        I09_04    28
#define        I08_04    29
#define        I07_04    30
#define        I06_04    31
#define        I05_04    18
#define        I04_04    19
#define        I03_04    20
#define        I02_04    21
#define        I01_04    22
#define        I00_04    23
#define        Ceu_04    24
#define        CeM_04    25
#define        Se_04     27
#define        Cmd_En    26
#define        Cmd_3     28
#define        Cmd_2     29
#define        Cmd_1     30
#define        Cmd_0     31
```

```
/* Variable Declarations */

extern    char cmd_line[80],*pcmd,micro_word[49],*pmwd;


/*  We need a static data structure which holds the
different choices available for bits I5 to I0 of the 2904.*/

char *choices_04(n) /* return a pointer to the n-th choice*/
     int  n;
     {
     static char *choice [] = {
     "0X0XXX", /* carry in = u carry, first choice -0 */
     "0XX1XX", /* carry in = u carry, second choice -1  */
     "0XXX1X", /* carry in = u carry, third choice -2 */
     "1X0XXX", /* carry in = Macro carry, first choice -3*/
     "1XX1XX", /* carry in = Macro carry, second choice -4*/
     "1XXX1X", /* carry in = Macro carry, third choice -5 */
     "00011X", /* Load u register, retain overflow bit -6 */
     "X1100X", /* Load u reg,invert carry, first choice -7*/
     "1X100X", /* Load u reg,invert carry,second choice -8*/
     "XX010X", /* Load u reg, immed., first choice. -9 */
     "X10XXX", /* Load u reg, immed., second choice. -10 */
     "X1XX1X", /* Load u reg, immed., third choice. -11 */
     "X1X1XX", /* Load u reg, immed., fourth choice. -12 */
     "1X0XXX", /* Load u reg, immed., fifth choice. -13 */
     "1XXX1X", /* Load u reg, immed., sixth choice. -14 */
     "1XX1XX", /* Load u reg, immed., seventh choice. -15 */
     "XX100X", /* Load M reg, invert carry -16 */
     "XXX11X", /* Load M reg, immed, first choice -17 */
     "XX1X1X", /* Load M reg, immed, second choice -18 */
     "XX11XX", /* Load M reg, immed, third choice -19 */
     "X10XXX", /* Load M reg, immed, fourth choice -20 */
     "1X0XXX"  /* Load M reg, immed, fifth choice -21 */
     };

               return (choice [n]);
     }


     /* Now we need a structure to manipulate this data.
Each time we   invoke a function with a choice, a pointer to
that choice gets added to the list of such pointers.  When
we display or save a microword, we will search this list to
find an entry which matches all of the chosen functions.  */

dummy()
     {
     puts ("Dummy called\n\n\0");
     }
```

```
shift2_menu()
     {
     puts(erase_screen);
     puts("You have chosen a down shift for this microword.
                                                There are\0"):
     puts("16 possible shift patterns, coded 0 thru F in
                                                  bits I9\0");
     puts("thru I6. Choose the shift pattern you desire from
                                                      the\0");
     puts("following set:\0");
     puts("\tzero  =    0    -> RAMn,        0  -> Qn\0");
     puts("\t one  =    1    -> RAMn,        1  -> Qn\0");
     puts("\t two  =    0    -> RAMn,     RAM0 -> Mc,   Mn ->
                                                      Qn\0"):
     puts("\tthree =    1    -> RAMn,     RAM0 -> Qn\0"):
     puts("\tfour  =    Mc   -> RAMn,     RAM0 -> Qn\0");
     puts("\tfive  =    Mn   -> RAMn,     RAM0 -> Qn\0");
     puts("\t six  =    0    -> RAMn,     RAM0 -> Qn\0");
     puts("\tseven =    0    -> RAMn,     RAM0 -> Qn,   Q0 -
                                                      Mc\0");
     puts("\teight =  RAM0 -> RAMn,        Q0  -> Qn,  RAM0 ->
                                                      Mc\0");
     puts("\tnine  =    Mc   -> RAMn,       Q0  -> Qn,  RAM0 -
                                                      Mc\0");
     puts("\t  A   =  RAM0 -> RAMn,        Q0  -> Qn\0");
     puts("\t  B   =    Ic   -> RAMn,     RAM0 -> Qn\0"):
     puts("\t  C   =    Mc   -> RAMn,     RAM0 -> Qn,   Q0 ->
                                                      Mc\0");
     puts("\t  D   =    Q0   -> RAMn,     RAM0 -> Qn,   Q0  ->
                                                      Mc\0");
     puts("\t  E   =  In exor IOvr -> RAMn,  RAM0 -> Qn\0");
     puts("\t  F   =    Q0   -> RAMn,     RAM0 -> Qn\0");
     puts("\t  H to get help with this procedure\0");
     puts("\t  N to back up one frame.\0");
     }


shift3_menu()
     {
     puts(erase_screen);
     puts("You have chosen an up shift for this microword.
                                                There are\0"):
     puts("16 possible shift patterns, coded 0 thru F in
                                                  bits I9\0"):
     puts("thru I6. Choose the shift pattern you desire from
                                                      the\0");
     puts("following set:\0");
     puts("\tzero  =    0    -> RAM0,        0  -> Q0,  RAMn ->
                                                      Mc\0");
     puts("\t one  =    1    -> RAM0,        1  -> Q0,  RAMn ->
                                                      Mc\0");
     puts("\t two   =    0    -> RAM0,        0   -> Q0 \0");
     puts("\tthree  =    1    -> RAM0,        1   -> Q0\0");
     puts("\tfour   =    Qn   -> RAM0,        0   -> Q0,   RAMn ->
                                                           Mc\0");
     puts("\tfive   =    Qn   -> RAM0,        1   -> Q0,   RAMn ->
                                                           Mc\0");
     puts("\t six   =    Qn   -> RAM0,        0   -> Q0\0");
     puts("\tseven  =    Qn   -> RAM0,        1   -> Q0\0");
     puts("\teight  =  RAMn   -> RAM0,       Qn   -> Q0,   RAMn ->
                                                           Mc\0");
     puts("\tnine   =    Mc   -> RAM0,       Qn   -> Q0,   RAMn ->
                                                           Mc\0");
     puts("\t  A    =  RAMn   -> RAM0,       Qn   -> Q0\0");
     puts("\t  B    =    Mc   -> RAM0,        0   -> Q0\0");
     puts("\t  C    =    Qn   -> RAM0,       Mc   -> Q0,   RAMn ->
                                                           Mc\0");
     puts("\t  D    =    Qn   -> RAM0,     RAMn   -> Q0,   RAMn -
                                                           Mc\0");
     puts("\t  E    =    Qn   -> RAM0,       Mc   -> Q0\0");
     puts("\t  F    =    Qn   -> RAM0,     RAMn   -> Q0\0");
     puts("\t  H to get help with this procedure\0");
     puts("\t  N to back up one frame.\0");
     }

carryin_menu()
     {
     puts(erase_screen);
     puts("\tThe Carry into the least significant stage of
                                                   the ALU\0");
     puts("is controlled by bits I12 and I11, and sometimes
                                                        bits\0");
     puts("I5, I3, I2, and I1. There are seven possible
                                                    choices:\0");
     puts("\tType a zero to select ZERO as the carry
                                                        -in.\0");
     puts("\tType a one to select ONE as the carry-in.\0");
     puts("\tType a two to select Cx, the Z output of the
                                                      29203.\0");
     puts("\tType a three to select the carry bit from the
                                                  micro reg\0");
     puts("\tType a four to select the micro carry bit
                                              complemented\0");
     puts("\tType a five to select the MACRO carry bit\0");
     puts("\tType a six to select the MACRO carry bit
                                             complemented\0");
     puts("\tType an H for help.\0");
     }

status1_menu()
     {
     puts(erase_screen);
     puts("\tBits I0 through I5 control the two different
                                                    status\0");
     puts("registers which may be selected.  There are three
                                                      main\0");
     puts("choices to be made, and you can change either or
                                                      both\0");
     puts("of the registers:\n\0");
     puts("\tType a '0' to make no changes to the status
                                                 registers\0");
     puts("\tType a '1' to change the micro status
                                                 register.\0");
     puts("\tType a '2' to change the MACRO status
                                                 register.\0");
     puts("\tType a 'D' to exit from this section.\0");
     }

status2_menu()
     {
     puts(erase_screen);
     puts("\tYou have chosen to modify the micro status
                                                 register \0");
     puts("(abbreviated uSR).  There are 15 different
                                                  choices:\0");
     puts("\tType a zero to load the MSR into the uSR.\0");
     puts("\tType a one to set all bits in the uSR\0");
     puts("\tType a two to swap the MSR and the uSR\0");
     puts("\tType a three to reset all bits to 0 in the
                                                       uSR\0");
     puts("\tType a four to load the uSR from the immed.
                                                    inputs\0");
     puts("\tType a five to load all uSR from I except
                                                  overflow\0");
     puts("\tType a six to load all uSR from I, invert carry
                                                       bit\0");
     puts("\tType a seven to reset only the zero flag in the
                                                       uSR\0");
     puts("\tType an eight to set only the zero flag in the
                                                       uSR\0");
     puts("\tType a nine to reset only the carry flag in the
                                                       uSR\0");
     puts("\tType an A to set only the carry flag in the
                                                       uSR\0");
     puts("\tType a B to reset only the sign flag in the
                                                       uSR\0");
     puts("\tType a C to set only the sign flag in the
                                                       uSR\0");
     puts("\tType a D to reset only the overflow flag in the
                                                       uSR\0");
     puts("\tType an E to set only the overflow flag in the
                                                       uSR\0");
     puts("\tType an H to get help\0");
```

```
    }

mic_stat_set(pchar)
     /* This routine sets up the microstatus register, in
        agreement   with the status2_menu. */

     char *pchar;
     {
     switch (*pchar)
          {
          case '0': /* load the microstatus from the MACRO
                                                    status */
               bit_clear (I05_04);
               bit_clear (I04_04);
               bit_clear (I03_04);
               bit_clear (I02_04);
               bit_clear (I01_04);
               bit_clear (I00_04);
               break;

          case '1': /* set the microstatus register. */
               bit_clear (I05_04);
               bit_clear (I04_04);
               bit_clear (I03_04);
               bit_clear (I02_04);
               bit_clear (I01_04);
               bit_set (I00_04);
               break;

          case '2': /* Swap the micro and MACRO status
                                                    words. */
               bit_clear (I05_04);
               bit_clear (I04_04);
               bit_clear (I03_04);
               bit_clear (I02_04);
               bit_set (I01_04);
               bit_clear (I00_04);
               break;

          case '3': /* zero the microstatus register. */
               bit_clear (I05_04);
               bit_clear (I04_04);
               bit_clear (I03_04);
               bit_clear (I02_04);
               bit_set (I01_04);
               bit_set (I00_04);
               break;

          case '4': /* Load the microstatus register from
the I pins.  Many possible inputs here, need an auxillary
data structure. */
```

```
          bit_set (I05_04);
          bit_set (I04_04);
          bit_set (I03_04);
          bit_set (I02_04);
          bit_set (I01_04);
          bit_set (I00_04);
          break;

     case '5': /* Load from I, with overflow retain. */
          bit_clear (I05_04);
          bit_clear (I04_04);
          bit_clear (I03_04);
          bit_set (I02_04);
          bit_set (I01_04);
          break;

     case '6': /* Load from I, with carry invert.  Need
                         to turn on I5 or I4 or both.  */
          bit_set (I03_04);
          bit_clear (I02_04);
          bit_clear (I01_04);
          break;

     case '7': /* Reset the zero flag. */
          bit_clear (I05_04);
          bit_clear (I04_04);
          bit_set (I03_04);
          bit_clear (I02_04);
          bit_clear (I01_04);
          bit_clear (I00_04);
          break;

     case '8': /* Set the zero flag. */
          bit_clear (I05_04);
          bit_clear (I04_04);
          bit_set (I03_04);
          bit_clear (I02_04);
          bit_clear (I01_04);
          bit_set (I00_04);
          break;

     case '9': /* Reset the carry flag. */
          bit_clear (I05_04);
          bit_clear (I04_04);
          bit_set (I03_04);
          bit_clear (I02_04);
          bit_set (I01_04);
          bit_clear (I00_04);
          break;
```

```
case 'a':
case 'A': /* Set the carry flag. */
     bit_clear (I05_04):
     bit_clear (I04_04);
     bit_set (I03_04);
     bit_clear (I02_04);
     bit_set (I01_04);
     bit_set (I00_04);
     break;

case 'b':
case 'B': /*  Reset the sign bit. */
     bit_clear (I05_04);
     bit_clear (I04_04);
     bit_set (I03_04);
     bit_set (I02_04);
     bit_clear (I01_04);
     bit_clear (I00_04);
     break;

case 'c':
case 'C': /*  Set the sign bit. */
     bit_clear (I05_04);
     bit_clear (I04_04);
     bit_set (I03_04);
     bit_set (I02_04);
     bit_clear (I01_04);
     bit_set (I00_04);
     break;

case 'd':
case 'D': /* Reset the overflow bit. */
     bit_clear (I05_04);
     bit_clear (I04_04);
     bit_set (I03_04);
     bit_set (I02_04);
     bit_set (I01_04);
     bit_clear (I00_04);
     break;

case 'e':
case 'E': /* Set the overflow bit. */
     bit_clear (I05_04);
     bit_clear (I04_04);
     bit_set (I03_04);
     bit_set (I02_04);
     bit_set (I01_04);
     bit_set (I00_04);
     break;

default:
```

```
                puts (" No help implemented yet. \n\0");
                sleep (2);
                break;

            }
        }

status3_menu()
        {
        puts(erase_screen);
        puts("\tYou have chosen to modify the MACRO status
                                                    register\0");
        puts("(abbreviated MSR).  The MSR is also controlled by
                                                        five\0");
        puts("enable bits, which are set on the next menu.\0");
        puts("There are 8 different choices in this menu:\0");
        puts("\tType a zero to load the Y inputs into the
                                                        MSR\0");
        puts("\tType a one to set all bits (if enabled)\0");
        puts("\tType a two to swap the MSR and the uSR\0");
        puts("\tType a three to reset all bits (if
                                                    enabled)\0");
        puts("\tType a four to swap the Mc and the Movr\0");
        puts("\tType a five to complement all bits\0");
        puts("\tType a six to load all MSR from I, invert
                                                      carry\0");
        puts("\tType a seven to load all MSR from I\0");
        }

macro_stat_set(pchar)
            /* This routine deals with the loading of the
                                        Macro status  register.*/
        char *pchar;
        {
        switch (*pchar)
            {
            case '0':  /* Load Y inputs into the MSR  */
                bit_clear (I05_04);
                bit_clear (I04_04);
                bit_clear (I03_04);
                bit_clear (I02_04);
                bit_clear (I01_04);
                bit_clear (I00_04);
                break;

            case '1':  /* Set all bits in the MSR  */
                bit_clear (I05_04);
                bit_clear (I04_04);
                bit_clear (I03_04);
                bit_clear (I02_04);
                bit_clear (I01_04);
```

```
               bit_set (I00_04);
               break;

          case '2':  /* Swap the MSR and the uSR */
               bit_clear (I05_04);
               bit_clear (I04_04);
               bit_clear (I03_04);
               bit_clear (I02_04);
               bit_set (I01_04);
               bit_clear (I00_04);
               break;

          case '3':  /* Reset all bits in the MSR */
               bit_clear (I05_04);
               bit_clear (I04_04);
               bit_clear (I03_04);
               bit_clear (I02_04);
               bit_set (I01_04);
               bit_set (I00_04);
               break;

          case '4':  /* Swap the Mc and the Movr  */
               bit_clear (I05_04);
               bit_clear (I04_04);
               bit_clear (I03_04);
               bit_set (I02_04);
               bit_clear (I01_04);
               bit_clear (I00_04);
               break;

          case '5':  /* Complement all bits in the MSR */
               bit_clear (I05_04);
               bit_clear (I04_04);
               bit_clear (I03_04);
               bit_set (I02_04);
               bit_clear (I01_04);
               bit_set (I00_04);
               break;

          case '6':  /* Load the MSR, with the carry
inverted.  Several choices here, need to implement the
decision process.  Turn on I04 or I05, or both.  I00 is a
don't care. */
               bit_set (I03_04);
               bit_clear (I02_04);
               bit_clear (I01_04);
               break;

          case '7':  /* Load the MSR from the I inputs.
                     /* Many choices here.            */
               bit_set (I05_04);
```

Program Name: extern.h
Purpose: This file is included in all modules which are not "main" programs. It must reside in the same directory in which all compilations are made.

```
/*  This header file used to assign external definitions to
all files except the main program.  The original definitions
should be contained in the main program. */

/*  LAST UPDATE: 5 Sep 1984 */

extern int docu_word[24];
extern char cmd_line[80],*pcmd,micro_word[49],*pmwd;
```

```
   /* these defines refer to the physical fields of the
microword.  They are used to pass field_set the fields */

#define         reg_src 1
#define         ien_fld 2
#define         oey_fld 3
#define         src_fld 4
#define         dest_fld 5
#define         func_fld 6
#define         carryin_fld 7
#define         I5I4_fld 8
#define         I3_I0_fld 9
#define         I5_I0_fld 10
#define         ceu_fld 11
#define         cem_fld 12
#define         cmden_fld 13
#define         shiften_fld 14
#define         command_fld 15
#define         shift_fld 16
#define         breakpoint_fld 17
#define         notused_fld 18
#define         msb_br_fld 19
#define         mid_br_fld 20
#define         lsb_br_fld 21
#define         rega_fld 22
#define         regb_fld 23
#define         seq_fld 24
#define        condtest_fld 25
```

Program Name: declare.h
Purpose: This file is included with all other modules.
         When compiling the modules, this file must be
         in the same directory.

```
/*  declare.h is the latest header file for declarations for
use with the 2900 system functional microprograming effort.
This source is set up to work with the vt100 from
VAX/unix.*/

/*  This header file should be "#included" with all
    modules.*/

/* LAST UPDATE: 5 sep 1984  */

#define   erase_screen "\033[2J\033[0;0H"  /* vt100 erase
                                    screen and home cursor  */
#define   TRUE       1
#define   FALSE      0
#define   false      0
#define   true       1
#define   no_sub     0


 /* These defines relate to the 29203ET board, and should be
contained in a routine to initialize them, rather than as
defines. */

#define        I12_04    16
#define        I11_04    17
#define        I10_04    8
#define        I09_04    28
#define        I08_04    29
#define        I07_04    30
#define        I06_04    31
#define        I05_04    18
#define        I04_04    19
#define        I03_04    20
#define        I02_04    21
#define        I01_04    22
#define        I00_04    23
#define        Ceu_04    24
#define        CeM_04    25
#define        Se_04     27
#define        Cmd_En    26
#define        Cmd_3     28
#define        Cmd_2     29
#define        Cmd_1     30
#define        Cmd_0     31
```

```
          bit_set(I11_04);
          bit_set(I05_04);
          bit_set(I03_04);
          bit_clear(I02_04);
          bit_clear(I01_04);
          break;

     default:  /* Help message on default */
          puts("No help available yet!!\n\0");
          sleep(2);
          break;
     }
}
```

```
carry_set(pchar)
     char *pchar;
     {
     switch (*pchar)
          {
          case '0':  /* carryin of zero */
               bit_clear(I12_04);
               bit_clear(I11_04);
               break;


          case '1':  /* carryin of one */
               bit_clear(I12_04);
               bit_set(I11_04);
               break;

          case '2':  /* carryin of Cx */
               bit_set(I12_04);
               bit_clear(I11_04);
               break;

          case '3':  /* carryin of micro carry */
     /* Three possible choices here, how do we record and
                                              decide? */
               bit_set(I12_04);
               bit_set(I11_04);
               bit_clear(I05_04);
               bit_clear(I03_04);
               break;

          case '4':  /* carryin of micro carry not */
               bit_set(I12_04);
               bit_set(I11_04);
               bit_clear(I05_04);
               bit_set(I03_04);
               bit_clear(I02_04);
               bit_clear(I01_04);
               break;

          case '5':  /* carryin of MACRO carry */
     /* Two other choices here, same problem as case 3 */
               bit_set(I12_04);
               bit_set(I11_04);
               bit_set(I05_04);
               bit_clear(I03_04);
               break;

          case '6':  /* carryin of MACRO carry not */
               bit_set(I12_04);
```

```
                bit_set(I09_04);
                bit_clear(I08_04);
                bit_set(I07_04);
                bit_set(I06_04);
                break;

           case 'c':
           case 'C':
                bit_set(I09_04);
                bit_set(I08_04);
                bit_clear(I07_04);
                bit_clear(I06_04);
                break;

           case 'd':
           case 'D':
                bit_set(I09_04);
                bit_set(I08_04);
                bit_clear(I07_04);
                bit_set(I06_04);
                break;

           case 'e':
           case 'E':
                bit_set(I09_04);
                bit_set(I08_04);
                bit_set(I07_04);
                bit_clear(I06_04);
                break;

           case 'f':
           case 'F':
                bit_set(I09_04);
                bit_set(I08_04);
                bit_set(I07_04);
                bit_set(I06_04);
                break;

           case 'n':
           case 'N':
                backout =1;
                break;

           default:
                puts("Sorry, No help yet - you're on your
                                                  own.\n\0");
                sleep(2);
                break;
           }
      return (backout);
      }
```

```
          bit_clear(I09_04);
          bit_set(I08_04);
          bit_clear(I07_04);
          bit_clear(I06_04);
          break;

     case '5':
          bit_clear(I09_04);
          bit_set(I08_04);
          bit_clear(I07_04);
          bit_set(I06_04);
          break;

     case '6':
          bit_clear(I09_04);
          bit_set(I08_04);
          bit_set(I07_04);
          bit_clear(I06_04);
          break;

     case '7':
          bit_clear(I09_04);
          bit_set(I08_04);
          bit_set(I07_04);
          bit_set(I06_04);
          break;

     case '8':
          bit_set(I09_04);
          bit_clear(I08_04);
          bit_clear(I07_04);
          bit_clear(I06_04);
          break;

     case '9':
          bit_set(I09_04);
          bit_clear(I08_04);
          bit_clear(I07_04);
          bit_set(I06_04);
          break;

     case 'a':
     case 'A':
          bit_set(I09_04);
          bit_clear(I08_04);
          bit_set(I07_04);
          bit_clear(I06_04);
          break;

     case 'b':
     case 'B':
```

```
                    break;

               default:
                    puts(" No help available yet.  n 0" ;
                    puts(" Start this process from the beginning
                                                      again. n 0" ;
                    next_level = FALSE;
                    sleep(2);
                    break;

               }
          return (next_level);
          }


     shift_set(pchar)
          char *pchar;
          {
          int backout;
          backout = 0;
          switch (*pchar)
               {
               case '0':
                    bit_clear(I09_04);
                    bit_clear(I08_04);
                    bit_clear(I07_04);
                    bit_clear(I06_04);
                    break;

               case '1':
                    bit_clear(I09_04);
                    bit_clear(I08_04);
                    bit_clear(I07_04);
                    bit_set(I06_04);
                    break;

               case '2':
                    bit_clear(I09_04);
                    bit_clear(I08_04);
                    bit_set(I07_04);
                    bit_clear(I06_04);
                    break;

               case '3':
                    bit_clear(I09_04);
                    bit_clear(I08_04);
                    bit_set(I07_04);
                    bit_set(I06_04);
                    break;

               case '4':
```

```
     puts("\tType a one for the MACRO status register\0");
     puts("\tType a two for the Immediate status inputs\0");
     puts("\tType a three for Immediate SIGN exor Macro
                                                  SIGN\0");
     puts("\tType a four for Imm. SIGN exnor MACRO SIGN\0");
     }

cond_l_set(pchar)
          /* This is the first level cond. code select, and
                                 matches status7_menu.    */
     char *pchar;
     {
     int  next_level;
     next_level = TRUE;
     switch (*pchar)
          {
          case '0':  /* Micro status register selected. */
               bit_clear(I05_04);
               bit_set(I04_04);
/* Note that I04 can be cleared for many cases, see Tbl. 4,
Pg 5-79 */
               break;

          case '1':  /* Macro status register. */
               bit_set(I05_04);
               bit_clear(I04_04);
               break;

          case '2':  /* Immediate Inputs. */
               bit_set(I05_04);
               bit_set(I04_04);
               break;

          case '3': /* Imm. sign exor MSR sign */
               bit_clear(I05_04);
               bit_clear(I04_04);
               bit_set(I03_04);
               bit_set(I02_04);
               bit_set(I01_04);
               bit_clear(I00_04);
               next_level = FALSE;
               break;

          case '4': /* Imm. sign exnor MSR sign */
               bit_clear(I05_04);
               bit_clear(I04_04);
               bit_set(I03_04);
               bit_set(I02_04);
               bit_set(I01_04);
               bit_set(I00_04);
               next_level = FALSE;
```

```
                    bit_clear(I00_04);
                    if (micro_word[I05_04] == '0')
                        bit_erase(I04_04);
                    break;

                case 'd':
                case 'D':  /* CARRY or  not ZERO */
                    bit_set(I03_04);
                    bit_set(I02_04);
                    bit_clear(I01_04);
                    bit_set(I00_04);
                    if (micro_word[I05_04] == '0')
                        bit_erase(I04_04);
                    break;

                case 'e':
                case 'E':  /* SIGN */
                    bit_set(I03_04);
                    bit_set(I02_04);
                    bit_set(I01_04);
                    bit_clear(I00_04);
                    break;

                case 'f':
                case 'F':  /* not SIGN */
                    bit_set(I03_04);
                    bit_set(I02_04);
                    bit_set(I01_04);
                    bit_set(I00_04);
                    break;

                default:
                    puts("No help yet.\n\0");
                    sleep (2);
                    break;
                }
            }

        status7_menu()
            {
            puts(erase_screen);
            puts("\tThere are two steps to selecting a test
                                                    condition.  The\0");
            puts("first is to select a register to be used, and the
                                                          second \0");
            puts("is to select a test on that register.  This menu
                                                         selects \0");
            puts("the register, or two special tests which combine
                                                              two\0");
            puts("registers.\n\0");
            puts("\tType a zero for the micro status register\0");
            bit_clear(I03_04);
            bit_set(I02_04);
            bit_set(I01_04);
            bit_set(I00_04);
            if (micro_word[I05_04] == '0')
                bit_erase(I04_04);
            break;

        case '8':  /* CARRY or ZERO */
            bit_set(I03_04);
            bit_clear(I02_04);
            bit_clear(I01_04);
            bit_clear(I00_04);
            if (micro_word[I05_04] == '0')
                bit_erase(I04_04);
            break;

        case '9':  /* not CARRY or not ZERO */
            bit_set(I03_04);
            bit_clear(I02_04);
            bit_clear(I01_04);
            bit_set(I00_04);
            if (micro_word[I05_04] == '0')
                bit_erase(I04_04);
            break;

        case 'a':
        case 'A':  /* CARRY */
            bit_set(I03_04);
            bit_clear(I02_04);
            bit_set(I01_04);
            bit_clear(I00_04);
            if (micro_word[I05_04] == '0')
                bit_erase(I04_04);
            break;

        case 'b':
        case 'B':  /* not CARRY */
            bit_set(I03_04);
            bit_clear(I02_04);
            bit_set(I01_04);
            bit_set(I00_04);
            if (micro_word[I05_04] == '0')
                bit_erase(I04_04);
            break;

        case 'c':
        case 'C':  /* not CARRY or ZERO */
            bit_set(I03_04);
            bit_set(I02_04);
            bit_clear(I01_04);
        bit_set(I00_04);
        if (micro_word[I05_04] == '0')
            bit_erase(I04_04);
        break;

   case '2':  /* SIGN exor OVR  */
        bit_clear(I03_04);
        bit_clear(I02_04);
        bit_set(I01_04);
        bit_clear(I00_04);
        if (micro_word[I05_04] == '0')
            bit_erase(I04_04);
        break;

   case '3':  /* SIGN exnor OVR */
        bit_clear(I03_04);
        bit_clear(I02_04);
        bit_set(I01_04);
        bit_set(I00_04);
        if (micro_word[I05_04] == '0')
            bit_erase(I04_04);
        break;

   case '4':  /* ZERO */
        bit_clear(I03_04);
        bit_set(I02_04);
        bit_clear(I01_04);
        bit_clear(I00_04);
        if (micro_word[I05_04] == '0')
            bit_erase(I04_04);
        break;

   case '5':  /* not ZERO */
        bit_clear(I03_04);
        bit_set(I02_04);
        bit_clear(I01_04);
        bit_set(I00_04);
        if (micro_word[I05_04] == '0')
            bit_erase(I04_04);
        break;

   case '6':  /* OVR */
        bit_clear(I03_04);
        bit_set(I02_04);
        bit_set(I01_04);
        bit_clear(I00_04);
        if (micro_word[I05_04] == '0')
            bit_erase(I04_04);
        break;

   case '7':  /* not OVR */
```

```
     puts("\tType a two to output the immediate inputs
                                                 from 0" :
     puts("\t\tthe ALU\0");
     puts("\tType a three for no output\0");
     }

status6_menu()
     {
     puts(erase_screen);
     puts("\tWhat condition do you want reflected by the
                                              condition 0" :
     puts("code output?\0");
     puts("\tType a zero for (SIGN exor OVR) or ZERO\0");
     puts("\tType a one for (SIGN exnor OVR) and not
                                                  ZERO\0");
     puts("\tType a two for (SIGN exor OVR)\0");
     puts("\tType a three for (SIGN exnor OVR\0");
     puts("\tType a four for ZERO\0");
     puts("\tType a five for not ZERO\0");
     puts("\tType a six for OVR\0");
     puts("\tType a seven for not OVR\0");
     puts("\tType an eight for (CARRY or ZERO)\0");
     puts("\tType a nine for (not CARRY) or (not ZERO)\0" :
     puts("\tType an A for CARRY\0");
     puts("\tType a B for not CARRY\0");
     puts("\tType a C for (not CARRY or ZERO)\0");
     puts("\tType a D for (CARRY or not ZERO)\0");
     puts("\tType an E for SIGN\0");
     puts("\tType an F for not SIGN\0");
     }

cond_2_set(pchar)
          /* This is the second level selection of the cond.
                                                     test *
     char (*pchar);
     {
     switch (*pchar)
          {
          case '0':  /* SIGN exor OVR or ZERO */
               bit_clear(I03_04);
               bit_clear(I02_04);
               bit_clear(I01_04);
               bit_clear(I00_04);
               if (micro_word[I05_04] == '0')
                   bit_erase(I04_04);
               break;

          case '1':  /* SIGN exnor OVR and not ZERO */
               bit_clear(I03_04);
               bit_clear(I02_04);
               bit_clear(I01_04);
```

```
                    bit_set (I04_04);
                    bit_set (I03_04);
                    bit_set (I02_04);
                    bit_set (I01_04);
                    bit_set (I00_04);
                    break;
               }
          }

 /*  The status4_menu is not used with the eval board, since
/* the individual status enables are not in the microword.*
status4_menu()
     {
     puts(erase_screen);
     puts("\tThere are six enable inputs to the status
                                               registers\0");
     puts("on the 2904.  They are a master enable for the
                                                    uSR,\0");
     puts("a master enable for the MSR, and individual
                                              enable for\0");
     puts("the four bits of the MSR (zero, carry, sign,
                                              overflow).\0");
     puts("You must chose which of these enables to
                                               activate.\0");
     puts("\tType a zero to activate the micro status
                                                register\0");
     puts("\tType a one to activate the MACRO status
                                                register\0");
     puts("\tType a two to activate the zero flag in the
                                                     MSR\0");
     puts("\tType a three to activate the carry flag in the
                                                     MSR\0");
     puts("\tType a four to activate the sign flag in the
                                                     MSR\0");
     puts("\tType a five to activate the overflow flag in
                                                 the MSR\0");
     puts("\tType a six if you want the rest of the flags
                                                disabled\0");
     puts("\tType an H for help.\0");
     }

status5_menu()
     {
     puts(erase_screen);
     puts("\tYou can output something from the 2904
                                                    onto\0");
     puts("the Y-bus. What do you want on the bus?\0");
     puts("\tType a zero to output the micro-status
                                                register\0");
     puts("\tType a one to output the macro-status
                                                register\0");
```

APPENDIX C

The following is the Compatibility Test program. It demonstrates the algorithm for finding compatible bit patterns when conflicts occur in shared microword fields.

```
#include <stdio.h>


/* We need a static data structure which holds the different
choices available for bits I5 to I0 of the 2904.    */

char *choices_04(n) /* return a pointer to the nth choice.*/
     int  n;
     {
     static char *choice [] = {
     "0X0XXX",/* carry in = u carry, first choice -0 */
     "0XX1XX",/* carry in = u carry, second choice -1  */
     "0XXX1X",/* carry in = u carry, third choice -2 */
     "1X0XXX",/* carry in=Macro carry,first choice -3 */
     "1XX1XX",/* carry in=Macro carry, second choice -4 */
     "1XXX1X",/* carry in=Macro carry, third choice -5 */
     "00011X",/* Load u register,retain overflow bit -6 */
     "X1100X",/* Load u reg,invert carry,first choice -7 */
     "1X100X",/* Load u reg,invert carry,second choice -8 */
     "XX010X",/* Load u reg, immed., first choice. -9 */
     "X10XXX",/* Load u reg, immed., second choice. -10 */
     "X1XX1X",/* Load u reg, immed., third choice. -11 */
     "X1X1XX",/* Load u reg, immed., fourth choice. -12 */
     "1X0XXX",/* Load u reg, immed., fifth choice. -13 */
     "1XXX1X",/* Load u reg, immed., sixth choice. -14 */
     "1XX1XX",/* Load u reg, immed., seventh choice. -15 */
     "XX100X",/* Load M reg, invert carry -16 */
     "XXX11X",/* Load M reg, immed, first choice -17 */
     "XX1X1X",/* Load M reg, immed, second choice -18 */
     "XX11XX",/* Load M reg, immed, third choice -19 */
     "X10XXX",/* Load M reg, immed, fourth choice -20 */
     "1X0XXX"/* Load M reg, immed, fifth choice -21 */
     };

               return (choice [n]);
     }
main()
{
int n,i,conflict;
char  *result;
```

```
static char *ptrarray[];
printf("Pick your first choice for bits I5-I0, 0-21\n");
scanf("%d",&n);
result = choices_04(n);
ptrarray[0]=choices_04(n);
printf("%d\n",result);
printf("The nth choice picked is, %d, the bits are,
                                              %s\n",n,result);
printf("Pick your second choice for bits I5-I0, 0-21\n");
scanf("%d",&n);
result = choices_04(n);
ptrarray[1]=choices_04(n);
printf("The nth choice picked is, %d, the bits are,
                                              %s\n",n,result);

printf("Ptrarray[0]= %c\n",*ptrarray[0]);
printf("Ptrarray[1]= %c\n",*ptrarray[1]);
printf("the value which starts at ptrarray[0] is
                                           %s\n",ptrarray[0]);
printf("the value which starts at ptrarray[1] is
                                           %s\n",ptrarray[1]);

conflict=0;


for (i=0;i<6;i++)

  if((*(ptrarray[0] + i)==*(ptrarray[1] + i))||
     (*(ptrarray[0] + i)=='X'||*(ptrarray[1] +i)=='X'))
     {
     conflict=0;
     printf("conflict=0\n");
     }
  else {
     conflict=1;
     printf("conflict=1\n");
     break;
     }
printf("conflict = %d\n",conflict);
printf("The index, i= %d\n",i);
if (conflict == 1)
   printf("Had a conflict!\n");
else
   printf("No conflicts!\n");

/**********************************************/


*ptrarray[0]='0';
*ptrarray[1]='1';
if(*ptrarray[0]==*ptrarray[1])
```

```
          printf("They were equal");
     else
          printf("They were not equal");
     }
```

BIBLIOGRAPHY

Brooks, F. P., The Mythical Man-Month, Addison-Wesley, 1975.

Kraft, G. D. and Toy, W. N., Microprogrammed Control and Reliable Design of Small Computers, Prentice-Hall, 1981.

Purdum, J., C Programming Guide, Que, 1983.

Siewiorek, D. B., Bell, C. G., and Newell, A., Computer Structures: Principles and Examples, McGraw-Hill, 1982.

White, D. E., Bit-Slice Design: Controllers and ALUs, Garland, 1981.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943	2
3.	LtCol. Alan A. Ross, USAF Code 52Rs Department of Computer Science Naval Postgraduate School Monterey, California 93943	3
4.	Herschel H. Loomis Code 62Lm Department of Electrical Engineering Naval Postgraduate School Monterey, California 93943	1
5.	LT Deborah R. Stiltner, USN Long Beach Naval Shipyard Long Beach, California 90822	2