30b

MICROPROGRAMMING A PROPOSED 16-BIT STACK MACHINE

by

STEVEN HOWARD CULP

B.A., Mid-America Nazarene College, 1985

A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Electrical and Computer Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1988

Approved by

John J Devore
Major Professor

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont.)

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank Dr. John J. Devore for the role he played as being my Major Professor and for the guidance, insight, and assistance he provided. I would also like to express my appreciation to the Electrical and Computer Engineering Department for its financial support and to the faculty thereof for their influential contributions to my education. My parents Raymond and Avis Culp who provided the foundation of my motivation, are to whom my deepest gratitude belongs. The immense depth of their interest and emotional support is valued so very much.

1 -- Objectives

This thesis presents the implementation of a specific instruction set on the 16-bit microprogrammed stack machine designed by Dr. Don Hush in 1982. The purpose for this research was threefold:

1. to verify that an arbitrary instruction set could indeed be implemented on the predetermined hardware design;
2. to provide a simple, yet illustrative example of microcoding; and
3. to develop an implementation tool in the form of a computer program which would automatically generate the required control bits.

Due to the nature of the research, little concern was given to the optimization and eloquence of the microcode. Instead, a rather straightforward, logical approach was adopted, which should prove to be more conducive to the learning process for students in the future.

Various instruction sets will place different demands on the structure of a machine and because of this, occasional tailoring of given design choices is expected. This tailoring process entails any changes that must be made to the machine's structure in order to implement the desired instruction set. For the instruction set presented in this thesis, these changes took the form of redefining several registers and two status flags. The redefining of these fields results in only minor changes in the original architecture presented by Dr. Hush, and this is favorable since paralleling his design as closely as possible was desired. The changes made to Dr. Hush's design will be noted in the text.

2 -- Abbreviations and Notation

The following is a list of abbreviations and a definition of the notation used throughout the text.

Abbreviations

AC -- ACcumulator

ALU -- Arithmetic Logic Unit

AND -- The logic AND function

CPU -- Central Processing Unit

CS -- Control Store. The memory where the microcode is stored

DCR -- Device Code Register

ea -- effective address. The address where the data is stored

FS -- Fast Stack

ILL -- Intermediate Language Level

IR -- Instruction Register, i.e. the macroinstruction register

IX -- the X Index register

IY -- the Y Index register

macroinstruction -- a single instruction which is a member of the instruction set specified by the machine architecture

MAR -- Memory Address Register

microinstruction -- an instruction which specifies the control bits for a single cycle. Microinstructions are more primitive in nature than macroinstructions, i.e. a single macroinstruction is composed of several microinstructions.

NA -- Next Address field

N/D -- Not Defined

OR -- the logic OR function

PC -- Program Counter

PLR -- PipeLine Register, i.e. the microinstruction register

RAM -- Random Access Memory

ROM -- Read Only Memory

RTN -- Register Transfer Notation

SP -- Stack Pointer

XOR -- the logic eXclusive OR function

Notation

BUS -- this refers to the system BUS and will always be capitalized. Any other bus referred to will be in lower-case letters.

CARRY -- this refers to the CARRY status bit

IMMEDIATE -- the IMMEDIATE field in the pipeline register

$I_{2,1}$ -- the comma indicates bits 2 and 1

I_{5-0} -- the dash indicates bits 5 through 0

<-- -- an arrow symbol indicates that a data transfer will take place. At the end of the given cycle, the data that is located on the BUS or in the register to the right of the arrow will be

copied in the register to the left of the arrow or loaded on the BUS, respectively.

= -- the equals sign indicates that no data transfer takes place. Instead, the specified register, control bit, etc. on the left of the equals sign contains the value on the right of the equals sign prior to the execution of the cycle.

Y <-- M[X] -- Brackets signify an access to memory. At the completion of this transfer, register Y will contain the contents of memory, the location of which is pointed to by register X.

( ) -- parenthesis signify that an option is available; one of the enclosed variables may be selected.

3 -- Overview

The machine presented in Dr. Hush's Master's Thesis was a 16-bit microprogrammed stack machine. The ALU was designed using the Am2901 Four-Bit Bipolar Microprocessor Slice, i.e. a four-bit chip-slice, which has 16 internal registers. The top four registers were utilized as a "Fast Stack" (FS) or small cache memory and four of these chip-slices were connected in order to obtain a full 16-bit ALU.

The 2910 Microprogram Controller was used as a Next Address Generator, which determines the next Control Store (CS) location to be accessed. Each time a next address is generated, the appropriate microinstruction is loaded into the pipeline register (PLR), also known as the microinstruction register. It is the contents of this register that control the execution of each microinstruction.

The machine originated by Dr. Hush had an undefined architecture, but was intended to operate as a stack machine. A stack machine operates by pulling the top two operands off the stack, performing the operation, and then pushing the result back on the stack. For a number of reasons however, the microcode that was written for this research assumed a single-address architecture. The primary disadvantage of adopting a single-address architecture point of view is that a degradation in speed occurs. One operand that is to be sent to the ALU generally lies in main memory which implies that a single-address architecture machine accesses memory much more frequently than a machine with a stack architecture. These additional accesses to memory result in a slower machine.

The decision of opting for the single-address architecture when writing the microcode was twofold. First, the curriculum at Kansas State University favors the single-address architecture approach to computer design, and therefore students studying the microcode would already be familiar with the sequence of events that must be executed. Realizing that the purpose of this machine is to serve as a teaching tool brings the second reason to light; speed is not of critical importance. If some speed is given up in order to have a machine that is easy for students to understand, then the sacrifice is certainly justified.

It is also for the sake of simplicity that the use of four internal registers as a small cache memory has been eliminated. This frees the four registers of the 2901 to be used for other purposes and places the entire stack in main memory, alleviating the need for certain TRAP routines which considerably complicated the machine. (These TRAP routines were necessary if an operation was to be performed on the FS when it was in an inadequate condition, namely too many or too few values on the FS).

The author's original intention was to build the machine and implement the instruction set on actual hardware. The four registers internal to the 2901 which served as a small cache memory complicated the implementation process to such an extent however, that the decision was made to keep the entire stack in main memory. When actually constructing the machine was observed to be a bit too ambitious, the top of the stack (formerly the four 2901 registers) was left in main memory. The author made this design choice due to the fact that the emphasis of his research was on producing an educational tool. No functionality of the instruction set is lost by allowing the top four stack locations to remain in main memory; indeed, instructions are executed at a slower speed, but this is not important for an educational tool.

This thesis presents an implementation of a specific instruction set on Dr. Hush's machine. Each instruction is decomposed into a sequence of microoperations which are expressed in Register Transfer Notation (RTN). The microoperations are then converted into Intermediate Language Level (ILL) mnemonics. Just as macroinstructions compose an instruction set, so these ILL mnemonics also compose their own instruction set. This instruction set will simply be referred to as the ILL. These ILL mnemonics strongly resemble assembly language code, but careful attention should be paid not to confuse the two. The implementable instruction set is given in assembly language code whereas the ILL represents the microcode. Each individual ILL statement represents one microinstruction and generally, several ILL statements are required to complete a single assembly language command. The ILL is one notch closer to machine code (the binary form of the microcode), i.e. the ILL serves as an interface which translates the assembly language commands to their corresponding machine code. Just as a high-level language (such as C, Pascal, Fortran etc.) command is broken down into several assembly language commands, so an assembly language command is decomposed into several ILL commands. The resulting ILL commands in turn specify the 1's and 0's required for machine execution.

The ILL presented within this thesis was created by the author for the purpose of bridging the gap between the instruction set and the microcode for this particular implementation task; it is by no means a universally accepted symbolic code.

Each ILL statement represents a single microinstruction. The control bits for each ILL command were then generated as they would appear in the Control Store and pipeline register, and may be viewed in their entirety in appendix A.

It is important to keep the following distinction in mind while reading this thesis: a <u>macro</u>instruction is one, specific instruction in the instruction set. A macroinstruction is executed by performing a series of ILL commands or <u>micro</u>instructions. Each microinstruction has its own location in the CS memory and is loaded into the pipeline register when it is ready to be executed.

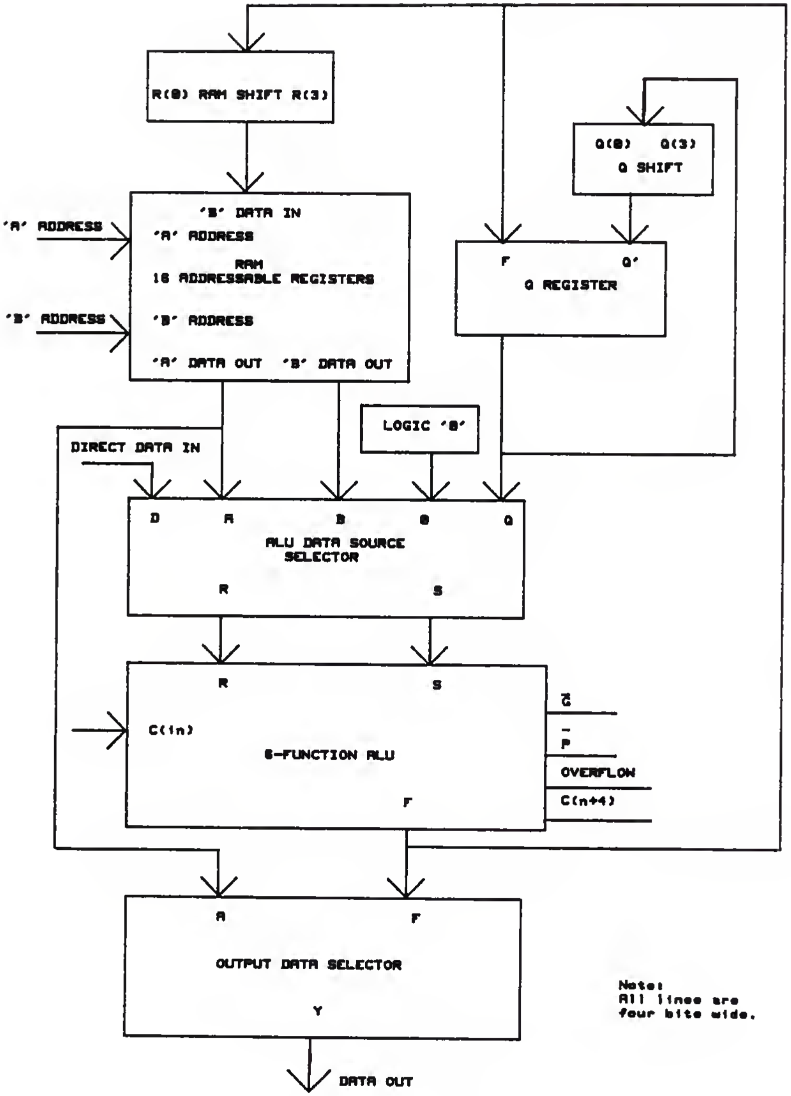
4 -- The ALU Hardware

4.1 Operation of the 2901

The main component in the ALU hardware is the 2901, a block diagram of which is illustrated in Fig. 4.1. This chip features a 16-word by 4-bit two-port RAM and a high-speed ALU.

The ALU has five possible sources which are passed through a selector circuit composed of two multiplexors. Each multiplexor has one output which is connected to an input of the ALU. The first multiplexor is a 2-to-1 multiplexor which selects between the Direct Data or D inputs and the ASEL field (to be defined shortly.) This multiplexor's output becomes the R input to the ALU. The second multiplexor selects one of three input fields: the ASEL field, the BSEL field or the Q register output. The Q register is a feature of the 2901 for enhanced execution of multiplication and division routines that require a double length operand and is not used in the implementation presented in this thesis. The output of this 3-to-1 multiplexor becomes the S input to the ALU. Additionally, both multiplexors have an inhibit capability which in effect, loads a logic "0."

The RAM is addressed by address lines A_{3-0} (A_3 through A_0), a.k.a. the ASEL field, which select the A inputs to the ALU, and B_{3-0}, a.k.a. the BSEL field, which select the



Note:
All lines are
four bits wide.

Figure 4.1 2901 Block Diagram

B inputs. Any two of the 16 RAM locations may be accessed in parallel and provided as operands to the ALU. A third input to the ALU may originate from the Direct Data In input, through which data coming from a source external to the chip enters the ALU. The Q register, as well as a "logic 0" input may also serve as operands to the ALU. These five sources are then passed through the selector circuit explained above, with the outputs of this circuit determining the R and S inputs to the ALU.

The ALU receives the two operands R and S, performs the currently selected function on them, and stores the result F in the local RAM prescribed by B_{3-0}. The option of transferring data off-chip is also available. The data to be transferred may either be the F output, i.e. the result of the ALU function, or must currently reside in the register addressed by the ASEL field.

The 2901 was designed as a bit-slice element so that expanding it from a four-bit data flow to a 16-bit data flow is accomplished merely by cascading four of the 2901s together. Because of this parallel cascading, the 2902 Carry Look-Ahead Generator is incorporated into the ALU circuit to increase the speed of computations.

4.2 The 2901 Control Bits

The following section goes into considerable detail about the control bits and how they affect the behavior of the 2901. These and all control bits are stored in the CS and are loaded into the pipeline register when they need to be executed, and it is the variance of these control bits that distinguishes one microinstruction from another.

Control bits I_{2-0} determine the ALU source operands, i.e. the R and S inputs to the ALU. These bits are enumerated in Table 4.1. When microprogramming this machine, the microprogrammer must decide which inputs are to be used. If sources A and B are desired, then control bits I_{2-0} would be assigned the values 001_2. Similarly, if the sources D and A were desired, then I_{2-0} would be set to 101_2.

I_{5-3} determine which ALU function is to be performed on the selected R and S inputs. These functions are listed in Table 4.2 and are fairly self-explanatory, with three of the functions performing binary arithmetic operations and five performing logic operations.

Tables 4.1 and 4.2 may be combined into a single matrix, shown in Table 4.3. I_{2-0} are listed horizontally across the top while I_{5-3} are listed vertically along the side. As an aid in understanding the usefulness of this figure, an example is now demonstrated.

> Suppose that an XOR operation to be performed on registers D and A is desired. Since

Table 4.1 ALU Source Operand Control Bits

			ALU Source Operands	
I2	I1	I0	R	S
0	0	0	A	Q
0	0	1	A	B
0	1	0	0	Q
0	1	1	0	B
1	0	0	0	A
1	0	1	D	A
1	1	0	D	Q
1	1	1	D	0

Table 4.2 ALU Function Control Bits

I5	I4	I3	Function
0	0	0	R plus S
0	0	1	S minus R
0	1	0	R minus S
0	1	1	R OR S
1	0	0	R AND S
1	0	1	R AND S
1	1	0	R XOR S
1	1	1	R XNOR S

Table 4.3 ALU Source Operand and Function Matrix

		I_{210}OCTAL							
		0	1	2	3	4	5	6	7
OCTAL I_{543}	ALU Function	ALU Source							
		A,Q	A,B	0,Q	0,B	O,A	D,A	D,Q	D,0
0	$C_n = L$ R Plus S	$A + Q$	$A + B$	Q	B	A	$D + A$	$D + Q$	D
	$C_n = H$	$A + Q + 1$	$A + B + 1$	$Q + 1$	$B + 1$	$A + 1$	$D + A + 1$	$D + Q + 1$	$D + 1$
1	$C_n = L$ S Minus R	$Q - A - 1$	$B - A - 1$	$Q - 1$	$B - 1$	$A - 1$	$A - D - 1$	$Q - D - 1$	$-D - 1$
	$C_n = H$	$Q - A$	$B - A$	Q	B	A	$A - D$	$Q - D$	$-D$
2	$C_n = L$ R Minus S	$A - Q - 1$	$A - B - 1$	$-Q - 1$	$-B - 1$	$-A - 1$	$D - A - 1$	$D - Q - 1$	$D - 1$
	$C_n = H$	$A - Q$	$A - B$	$-Q$	$-B$	$-A$	$D - A$	$D - Q$	D
3	R OR S	$A \vee Q$	$A \vee B$	Q	B	A	$D \vee A$	$D \vee Q$	D
4	R AND S	$A \wedge Q$	$A \wedge B$	0	0	0	$D \wedge A$	$D \wedge Q$	0
5	$\overline{R}$ AND S	$\overline{A} \wedge Q$	$\overline{A} \wedge B$	Q	B	A	$\overline{D} \wedge A$	$\overline{D} \wedge Q$	0
6	R EX-OR S	$A \nabla Q$	$A \nabla B$	Q	B	A	$D \nabla A$	$D \nabla Q$	D
7	R EX-NOR S	$\overline{A \nabla Q}$	$\overline{A \nabla B}$	$\overline{Q}$	$\overline{B}$	$\overline{A}$	$\overline{D \nabla A}$	$\overline{D \nabla Q}$	$\overline{D}$

+ = Plus; − = Minus; v = OR; ∧ = AND; ∇ = EX-OR registers D and A are needed, I_{2-0} are 101_2, and the XOR operation is performed by setting bits I_{5-3} to 110_2. The concatenation of control bits I_{5-0} would then be 110101_2.

The disposition of the result produced by the ALU is a function of control bits I_{8-6}. A summary of allowed operations and destinations is shown in Table 4.4. When microprogramming this machine, the instruction set was implemented in such a way that the I_{8-6}, RAM function and Y output specifications were of primary concern. For example, if the result of a given ALU operation needed to be stored in one of the internal RAM locations, then any value for I_{8-6} except 000_2 and 001_2 could be used. Furthermore, if the result was not to be shifted, then values of 100_2 through 111_2 for I_{8-6} were eliminated. Thus, a choice of 010_2 and 011_2 remain available, and the determination of whether the A select register or the ALU result F should be relayed to the Y output dictated which to use. If the F output of the ALU was desired, then I_{8-6} would be set to 011_2, otherwise I_{8-6} would equal 010_2.

A_{3-0} and B_{3-0} are specified by the ASEL and BSEL control bits, and it is these control bits that select one of the 16 internal RAM locations defined in Table 4.5. A slight deviation from Dr. Hush's design now occurs, i.e. some register redefinitions have taken place. The main

Table 4.4 ALU Destination/Shift Control

I8	I7	I6	RAM Function Shift	RAM Function Load	Q-reg. Function Shift	Q-reg. Function Load	Y Output	RAM Shifter RAMØ	RAM Shifter RAM3
Ø	Ø	Ø	X	none	none	F-->Q	F	X	X
Ø	Ø	1	X	none	X	none	F	X	X
Ø	1	Ø	none	F-->B	X	none	A	X	X
Ø	1	1	none	F-->B	X	none	F	X	X
1	Ø	Ø	down	F/2-->B	down	Q/2-->Q	F	FØ	IN3
1	Ø	1	down	F/2-->B	X	none	F	FØ	IN3
1	1	Ø	up	2F-->B	up	2Q-->Q	F	INØ	F3
1	1	1	up	2F-->B	X	none	F	INØ	F3

reallocations of which the microprogrammer should be aware are locations 0000_2 and 0011_2. These registers have been changed from FS registers to a Device Code Register (DCR) and an Accumulator (AC), respectively. Also, locations 1010_2 and 1011_2 have become the X Index register (IX) and the Y Index register (IY).

The ALU can operate in a total of four modes determined by control bits $M_{1,0}$. Mode 00_2 is the typical mode with registers A and B coming from the ASEL and BSEL fields as specified in the PLR. Mode 11_2 is used for register reference instructions and is also fairly common. In this mode the A and B registers are taken directly from the IR. Mode 10_2 is used for computing the ea when indexed addressing is invoked, and allows the ASEL register (the register to be indexed from) to be defined in the IR. Since mode 01_2 deals with the Bottom Of Stack (BOS) register and was initially designed by Dr. Hush to handle a certain TRAP condition that could occur on the FS, it is never used in this implementation. (These FS registers have been reallocated to other uses and the stack relocated to main memory, which completely eliminates the need for this mode.) These mode functions are summarized in Table 4.6.

The value for the carry-in bit involves programming two control bits: C_{in} and 0/1. If C_{in} equals 1, then the carry in value is determined by the CARRY bit. If C_{in}

Table 4.5 2901 RAM Register Definitions

ASEL or BSEL	2901 Register
0000	Device Code Register
0001	N/D
0010	N/D
0011	Accumulator
0100	TEMP1
0101	TEMP2
0110	N/D
0111	N/D
1000	N/D
1001	PC, Program Counter
1010	IX, X Index Register
1011	IY, Y Index Register
1100	N/D
1101	N/D
1110	SP
1111	N/D

Table 4.6 Modes of ALU Operation

M1	M0	Reg. A Select	Reg. B Select
0	0	ASEL	BSEL
0	1	ASEL	BOS
1	0	A(IR)	BSEL
1	1	A(IR)	B(IR)

equals 0 however, the carry in value is determined from the 0/1 control bit. In this way the microprogrammer can either force or not force a carry in. This is illustrated in Table 4.7.

Whenever a value from the Central Processing Unit (CPU) needs to leave the 2901 to go to main memory, I/O, etc., control bits $BS_{1,0}$ must be set to 01_2. BS stands for Bus Select and it is through the BUS that the CPU communicates with other chips and devices. Additionally, the CPU status bits may be routed to the BUS by setting $BS_{1,0}$ to 10_2. Table 4.8 summarizes the $BS_{1,0}$ control bits.

Conversely, data enters the CPU from an off-chip source through the D inputs by setting the control bit Bus to 1. Bus must equal 0 for Table 4.8 to apply.

The final set of control bits with which the microprogrammer must be concerned for the 2901 is $DS_{1,0}$, listed in Table 4.9. These two control bits determine the source of the external data inputs to the 2901. Recall that the D inputs come from off the chip, and $DS_{1,0}$ determine the source of this input. Perhaps the most common value for $DS_{1,0}$ is 00. This indicates that the IMMEDIATE field of the PLR is fed into the D inputs. This of course, is very useful when forcing an increment, e.g. PC <-- PC plus 2.

In summary, the general architecture of the 2901 has

Table 4.7 ALU Carry In Determination Control Bits

C(in)	"Carry in" to ALU
0	0/1
1	CARRY

Table 4.8 BUS Select Control Bits

BS1	BS0	Data to BUS
0	0	Nothing (safe state)
0	1	Y's from 2901
1	0	CPU status bits
1	1	N/D

Table 4.9 2901 Direct Data Select Control Bits

DS1	DS0	D Inputs to 2901
0	0	IMMEDIATE
0	1	N/D
1	0	IR 2nd byte
1	1	N/D

been presented with special attention given to the control bits and how they determine the operation of the ALU.

5 -- The Control Unit

5.1 General Structure of the Control Unit

There are two main methods of control unit (CU) implementation: hardwiring and microprogramming. When a CU is hardwired, additional logic gates are needed, and processes such as prime implicant identification, control point gathering, etc. are performed. The particular CU implemented for this research however, is microprogrammed. Microprogramming differs from hardwiring in that control bits dictate the flow of data and the determination of the next address. These governing control bits are specified by the microprogrammer, as opposed to control signals being produced by the hardware. It is the 1's and 0's of the control bits that dictate exactly what happens during each microinstruction. The previous chapter illustrated how various control bits determined a specific operation, specified where to store the result, and signaled what data was to enter or exit the CPU. This chapter will parallel chapter four in that the necessary control bits are defined, but this chapter deals with the CU and the 2910 Microprogram Controller.

The CU of this machine is predominately horizontal in nature as may be observed by the rather large, 67-bit control word. The control word could alternatively have been vertical in nature, resulting in a much shorter word, but this would have slowed the execution time. As might be expected, a trade-off is present: faster execution at the expense of a large control word, or degradation of speed with a small control word.

After a macroinstruction is loaded into the IR, it is executed by a series of steps specified by microinstructions. A microinstruction that is ready to be executed is transferred from the Control Store (where the microcode is stored) into the PLR, and it is from the PLR that the required control bits are used.

A microinstruction is usually composed of two major fields which execute simultaneously. The first field provides bits necessary for the ALU to function (discussed previously), for memory and for I/O. The second field is used for next address generation, i.e. determining which address of the CS to access next.

5.2 The 2910

The Next Address (NA) generation is accomplished through the use of the 2910 Microprogram Controller, a block diagram of which is shown in Fig. 5.1. The microprogram counter is used when accessing the next sequential CS location and is probably the block used most frequently. However, other logic is present as well, including a 9-word X 12-bit stack which can be used for

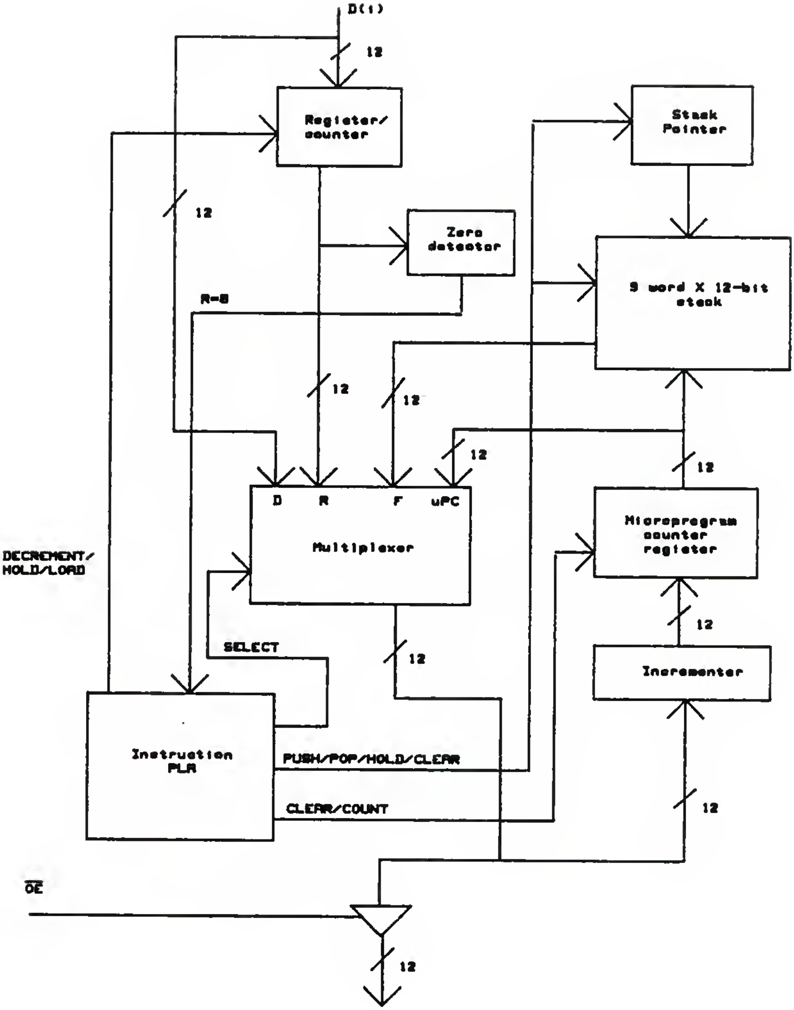


Figure 5.1 2910 Block Diagram

nesting microinstruction subroutines. Any data entering the chip from an external source enters through the D inputs, and once the Next Address has been generated, it exits the chip via the Y output bus.

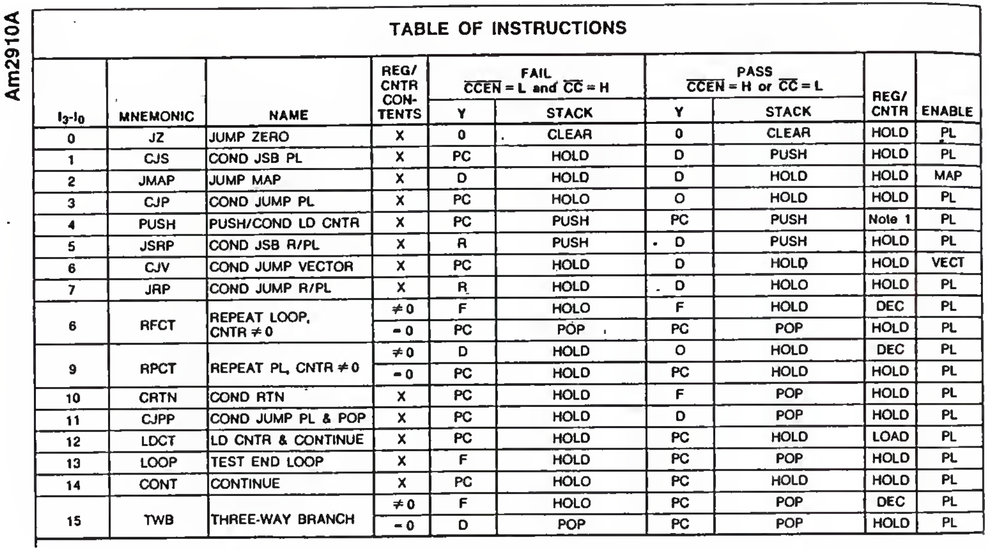
The instruction set for the 2910 appears in Table 5.1, and the vast majority of the microcoding was done using three of the 16 instructions. These three instructions are explained in detail.

Instruction number 2, JMAP, performs an unconditional jump. The flow of such an instruction is illustrated in Fig. 5.2, from which it is apparent that regardless of a previous result or condition, control is routed out of the normal next sequential address to a different address. This new address is located in the NA field of the PLR (microinstruction register) and must be specified by the microprogrammer.

Instruction 3, CJP, is a conditional jump instruction. If a certain condition is met, a status flag being set for example, a jump to another location is executed. Once again, the NA field of the PLR provides this address. If however, the condition is not met, then the jump is aborted and sequential access is continued. Fig. 5.3 pictures the conditional jump flow.

Fig. 5.4 illustrates the flow for the final instruction that was most commonly used. This is the

Table 5.1 2910 Instruction Set



Am2910A

TABLE OF INSTRUCTIONS

I_3-I_0	MNEMONIC	NAME	REG/ CNTR CON-TENTS	FAIL $\overline{CCEN}$ = L and $\overline{CC}$ = H		PASS $\overline{CCEN}$ = H or $\overline{CC}$ = L		REG/ CNTR	ENABLE
				Y	STACK	Y	STACK		
0	JZ	JUMP ZERO	X	0	CLEAR	0	CLEAR	HOLD	PL
1	CJS	COND JSB PL	X	PC	HOLD	D	PUSH	HOLD	PL
2	JMAP	JUMP MAP	X	D	HOLD	D	HOLD	HOLD	MAP
3	CJP	COND JUMP PL	X	PC	HOLD	D	HOLD	HOLD	PL
4	PUSH	PUSH/COND LD CNTR	X	PC	PUSH	PC	PUSH	Note 1	PL
5	JSRP	COND JSB R/PL	X	R	PUSH	D	PUSH	HOLD	PL
6	CJV	COND JUMP VECTOR	X	PC	HOLD	D	HOLD	HOLD	VECT
7	JRP	COND JUMP R/PL	X	R	HOLD	D	HOLD	HOLD	PL
6	RFCT	REPEAT LOOP, CNTR ≠ 0	≠ 0	F	HOLD	F	HOLD	DEC	PL
			= 0	PC	POP	PC	POP	HOLD	PL
9	RPCT	REPEAT PL, CNTR ≠ 0	≠ 0	D	HOLD	D	HOLD	DEC	PL
			= 0	PC	HOLD	PC	HOLD	HOLD	PL
10	CRTN	COND RTN	X	PC	HOLD	F	POP	HOLD	PL
11	CJPP	COND JUMP PL & POP	X	PC	HOLD	D	POP	HOLD	PL
12	LDCT	LD CNTR & CONTINUE	X	PC	HOLD	PC	HOLD	LOAD	PL
13	LOOP	TEST END LOOP	X	F	HOLD	PC	POP	HOLD	PL
14	CONT	CONTINUE	X	PC	HOLD	PC	HOLD	HOLD	PL
15	TWB	THREE-WAY BRANCH	≠ 0	F	HOLD	PC	POP	DEC	PL
			= 0	D	POP	PC	POP	HOLD	PL

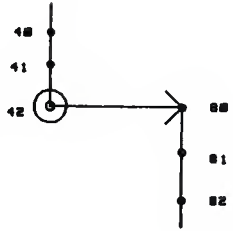


Figure 5.2 JMAP Flow

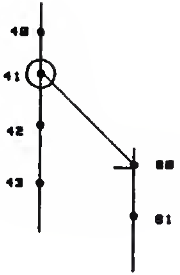


Figure 5.3 CJP Flow

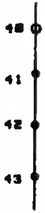


Figure 5.4 CONT Flow

continue (CONT) instruction, and it merely causes the microprogram counter to be incremented and the next sequential address to be accessed.

5.3 The 2910 Control Bits

In addition to I_{3-0}, a number of other control bits are needed for proper operation of the 2910. Control bits T_{3-0} for example, specify which status condition to test for and are listed in Table 5.2. Once again a slight deviation from Dr. Hush's design has been made. TRAP1 and TRAP2 flags, signified by T_{3-0} being 0111_2 and 1000_2, respectively, and were necessary only because of the small cache memory located internal to the 2901, have been replaced by N XOR V and Halt status flags. This redefinition considerably simplified the microcoding.

For every test condition, the microprogrammer may use positive or negative polarity. The control bit which specifies this is POL, and POL equalling 0 implies positive logic; POL equalling 1 implies negative logic. Table 5.3 results.

The final two control bits required by the 2910 are $S_{1,0}$ and they determine the D inputs. Recall that all external sources must enter the 2910 via the D inputs and the possible sources are given in Table 5.4. The definition of these bits is believed to be self-explanatory, and any confusion that may be currently

Table 5.2 Status Condition
Select Control Bits

Test Field				Selected Status
T3	T2	T1	T0	
0	0	0	0	1
0	0	0	1	CARRY
0	0	1	0	OVERFLOW
0	0	1	1	SIGN
0	1	0	0	ZERO
0	1	0	1	INTERRUPT
0	1	1	0	I/O READY
0	1	1	1	N XOR V
1	0	0	0	HALT
1	0	0	1	N/D
1	0	1	0	N/D
1	0	1	1	N/D
1	1	0	0	N/D
1	1	0	1	N/D
1	1	1	0	N/D
1	1	1	1	N/D

Table 5.3 Polarity Definition Control Bits

POL	Test for:
0	TRUE (1)
1	FALSE (0)

Table 5.4 2910 D Input Select Control Bits

S1	S0	2910 D Inputs
0	0	Next Address Field
0	1	IR 2nd byte
1	0	IR opcode
1	1	IR ea/opcode

present should be cleared when examples of actual microinstructions are given.

6 -- Main Memory

6.1 Main Memory Organization

The main memory of this machine contains 32k-words of 16-bits each, resulting in a total of 64k-bytes of storage. The Hitachi 6116 static CMOS RAM was chosen by Dr. Hush to implement the memory, and the 6116 is available with a 2k X 8-bit format. A full 64k-byte memory would then require 32 of these RAMs and they are arranged in 16 blocks as illustrated in Fig. 6.1. Since Dr. Hush's design however, Hitachi has produced a 61256 chip which is 32k X 8-bits. Using the 61256 for main memory, only two chips would be needed!

6.2 Main Memory Access

Inclusive in the hardware design for main memory is a Memory Address Register (MAR). The MAR is a 16-bit register which holds the address of the memory location to be accessed during both read and write operations. As a design constraint imposed by the author, the MAR may contain only even-numbered addresses, thus forcing an entire 16-bit word to be accessed each time memory is invoked.

A Memory Buffer Register (MBR) is not needed in this system since no handshaking sequence exists; all transfers to and from main memory occur in two microinstruction

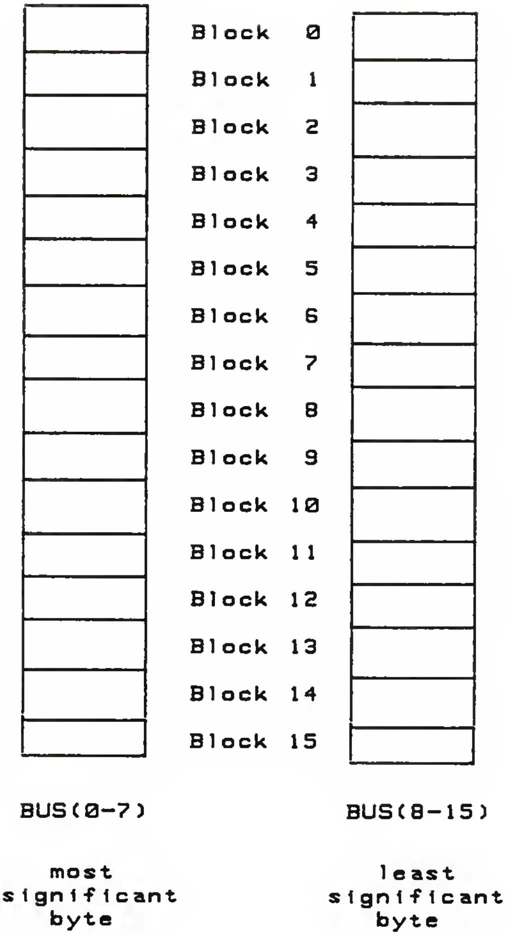


Figure 6.1 Main Memory Format

cycles.

A read instruction consists of two sequential microinstruction cycles. In the first cycle the CPU sends its data, i.e. an address, via the BUS into the MAR, where in the second cycle the data is actually read. This of course, suggests the use of control bits to govern these operations. In the first microinstruction cycle $BS_{1,0}$ must be set to 01_2 in order to pipe the Y output of the 2901 to the BUS. (See Table 4.8). Also LDMAR must be set to 1 to enable the MAR to be loaded. For the second microinstruction cycle, control bits MEMSEL and $R/\overline{W}$ both must be set to 1, indicating that memory is being selected and that a read operation is being performed.

A write operation is practically identical to the read operation. It also requires two sequential microinstruction cycles, the first of which parallels the read operation exaclty and the second of which differs in only one control bit; $R/\overline{W}$ must be cleared to 0, indicating a write operation.

Summarizing, main memory is organized in 32k-words with each word being 16-bits wide. In addition to the 6116s (or the 61256s), the other major component to the memory hardware is a 16-bit MAR. All transfers to and from main memory are performed in two sequential microinstruciton cycles: one to send the address to the

MAR and one to transfer the data.

7 -- Data Flow

A fairly detailed block diagram of the computer system being microprogrammed and the governing control bits is illustrated in Fig. 7.1. The diagram is given at this time to aid the reader in conceptualizing exactly what happens during the execution of an instruction.

7.1 Data Flow for Memory Reference Instructions

Memory reference instructions are four-byte instructions. The first two bytes define the opcode and the addressing scheme and are present in the IR upon completion of the instruction fetch. Before the designated opcode routine can be performed however, an operand located in main memory must be received. In order to obtain this operand the address where it is located must be computed; this process is known as the effective address (ea) calculation since finding the address where the actual data is stored needs to be performed.

Numerous microinstruction routines have been written to facilitate ea calculation and are presented in detail in appendix A. Once the ea has been calculated, the data is fetched and the operation prescribed by the opcode performed. The result is then either stored in the internal RAM registers or transferred to main memory.

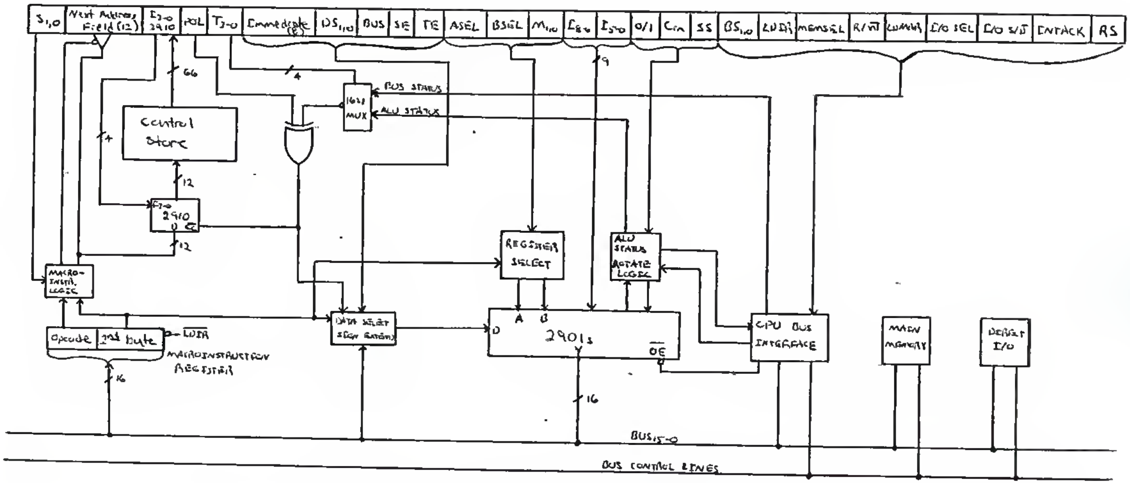


Figure 7.1 Computer Block Diagram with Control Bits

7.2 Data Flow for Register Reference Instructions

Register reference instructions are two-byte instructions, implying that after the instruction fetch, no additional access to memory is required. The first byte defines the opcode and the second byte indicates the source and destination registers. After the instruction fetch, the data flow is confined strictly to the 2901 ALU chip; this is the reason register reference instructions execute so rapidly.

Fig. 7.1 may appear a bit overwhelming at first glance, but after studying and working with it for a period of time the intimidating initial impression disappears. At that point, it is certain the microprogrammer will find that periodically referring to it will prove to be quite beneficial.

8 -- Instruction Set Definition

Up to now, the groundwork upon which the actual research was performed has been laid. The reader should at this point possess a working knowledge of the 2901, the 2910, of Dr. Hush's paper-designed machine and of the few, minor changes that have been made to his machine. If comprehension in any of these areas is weak, reviewing the preceeding text and references [1] and [2] is strongly urged.

The instruction set to be presented consists of 29 instructions, 11 of which are memory reference instructions, eight of which are register reference instructions, eight of which are branching instructions and two of which are I/O instructions.

8.1 Goals of the Instruction Set

When choosing the instructions to be included, several criteria were carefully weighed. First, the instruction set should be confined to a reasonable number of instructions, all of which in of themselves are basic, yet when integrated into a whole, form a powerful programming tool. For this reason, this machine may be considered to have a Reduced Instruction Set Computer (RISC) nature, although that was not its focal point.

Strong branching was also desired. Since control of the flow for a program is dictated directly by branch statements, a large number of branching options seemed to be an inherent part of obtaining a powerful instruction set.

In addition, subroutines were to be facilitated, particularly the capability of nesting subroutines. By allowing the programmer to nest subroutines, the depth and complexity to which he may work has been substantially increased, thus promoting more efficient programs. Additionally, possessing the ability to nest subroutines adds a touch of flare to any instruction set!

Finally, an assortment of addressing modes was sought. The more methods possible in which a programmer may access data relates proportionally to the efficiency and eloquence of a program, and providing the programmer with these tools was a primary goal.

Successfully meeting the four criteria outlined above should result in a highly efficient, workable instruction set, and it is hoped that the instruction set listed in the following section is found to be so.

8.2 The Macroinstructions

The instructions given below compose the instruction set. These instruction are called macroinstructions, are the equivalent of a given microprocessor's assembly language, and are executed by a series of microinstructions. Given with each mnemonic is its expanded form, the Register Transfer Notation(s) (RTNs) required, and a brief English description of what takes place upon execution of the instruction.

Memory reference instructions are listed first, register reference instructions next, branch instructions following with the I/O commands concluding the section. Each group of instructions are listed by increasing opcode number which will be defined shortly. The instructions of the instruction set follow.

Memory reference instructions

LAC -- Load ACcumulator

AC <-- M[ea]

A read from main memory is performed and the data is loaded into the accumulator.

SAC -- Store ACcumulator

M[ea] <-- AC

The data stored in the accumulator is written to main memory.

AND -- logic AND operation

AC <-- AC AND M[ea]

The data retrieved from the access to memory is ANDed with the data stored in the accumulator. The result is then stored back in the accumulator.

OR -- logic OR operation

AC <-- AC OR M[ea]

The data retrieved from the access to memory is ORed with the data stored in the accumulator. The result is then stored back in the accumulator.

ADD -- ADDition operation

AC <-- AC plus M[ea]

The data retrieved from the access to memory is added with the data stored in the accumulator. The result is then stored back in the accumulator.

SUB -- SUBtraction operation

AC <-- AC minus M[ea]

The data retrieved from the access to memory is subtracted from the data stored in the accumulator. The result is then stored back in the accumulator.

PUSH -- PUSH accumulator on stack

SP <-- SP plus 2
M[SP] <-- AC

The Stack Pointer (SP) is adjusted so data will not be overwritten. The data located in the accumulator is written to the memory location pointed to by the stack pointer.

PULL -- PULL off stack into accumulator

AC <-- M[SP]
SP <-- SP minus 2

The top data value on the stack is read from memory and loaded into the accumulator. The stack pointer is then adjusted to point to the top valid data location.

RTI -- ReTurn from Interrupt

PC <-- M[SP]
SP <-- SP minus 2
Restore status registers

The address of the next macroinstruction is loaded from the stack into the program counter. The stack pointer is then adjusted and the status register is restored.

RTS -- ReTurn from Subroutine

PC <-- M[SP]
SP <-- SP minus 2

The address of the next macroinstruction is loaded from the stack into the program counter. The stack pointer is then adjusted.

NOP -- No OPeration

There is no real reason for this instruction; it just seems no instruction set is complete without one!

Register reference instructions

INC -- INCrement
(A 2901 reg.) <-- (A 2901 reg.) plus 1
The selected register will be incremented by 1.

DEC -- DECrement
(A 2901 reg.) <-- (A 2901 reg.) minus 1
The selected register will be decremented by 1.

ROR -- ROtate Right
(A 2901 reg.) <-- ROR(A 2901 reg.)
The selected register will be rotated right 1 bit. Note -- the CARRY bit is included.

ROL -- ROtate Left
(A 2901 reg.) <-- ROL(A 2901 reg.)
The selected register will be rotated left 1 bit. Note -- the CARRY bit is included.

CLR -- CLeaR
(A 2901 reg.) <-- 0
The selected register will be cleared, i.e. loaded with zeros.

COM -- COMplement
(A 2901 reg.) <-- COM(A 2901 reg.)
The selected register will be complemented, i.e. a 0 will become a 1 and vice versa.

TFR -- TransFeR
(A 2901 reg.) <-- (A 2901 reg.)

The data currently in the source register will be transferred to the destination register.

HLT -- HaLT

Halt execution of machine.

A NOP is continually jumped to with the halt flag tested each time.

Branch instructions

BEQ -- Branch if EQual

```
If "Z" = 1
then PC <-- PC plus relative address$_{SE}$
else PC <-- PC plus 0
```

The "Z" status flag is tested. If set, then the relative address is sign extended and added to the program counter. If the "Z" flag is clear, then 0 is added to the program counter. In either case, the result is stored back into the program counter.

BNE -- Branch if Not Equal

```
If "Z" = 0
then PC <-- PC plus relative address$_{SE}$
else PC <-- PC plus 0
```

The "Z" status flag is tested. If clear, then the relative address is sign extended and added to the program counter. If the "Z" flag is set, then 0 is added to the program counter. In either case, the result is stored back into the program counter.

BGT -- Branch if Greater Than

```
If "N XOR V" = 0
then
   if "Z" = 0
   then PC <-- PC plus relative address$_{SE}$
   else PC <-- PC plus 0
else PC <-- PC plus 0
```

The "N XOR V" status flag is tested. If it is set then 0 is added to the PC. If it is clear, then the "Z" status flag is tested. If the "Z" flag is clear then the relative address is sign extended and added to the program counter. If it is set, then 0 is added to the program counter.

BLT -- Branch if Less Than

```
If "N XOR V" = 1
then PC <-- PC plus relative address$_{SE}$
else PC <-- PC plus 0
```

The "N XOR V" status flag is tested. If set, then the relative address is sign extended and added to the program counter. If it is clear, then 0 is added to the program counter.

BGE -- Branch if Greater than or Equal

```
If "N XOR V" = 0
then PC <-- PC plus relative address$_{SE}$
else PC <-- PC plus 0
```

The "N XOR V" status flag is tested. If clear, then the relative addres is sign extended and added to the program counter. If it is set, then 0 is added to the program counter.

BLE -- Branch if Less than or Equal

```
If "N XOR V" = 1
then PC <-- PC plus relative address$_{SE}$
else
   if "Z" = 1
   then PC <-- PC plus relative address$_{SE}$
   else PC <-- PC plus 0
```

The "N XOR V" status flag is tested. If set, then the relative address is sign extended and added to the program counter. If it is clear, then the "Z" status flag is tested. If set, then the relative address is sign extended and added to the program counter. If clear, then 0 is added to the program counter.

BSR -- Branch to SubRoutine

```
SP    <-- SP plus 2
M[SP] <-- PC
PC    <-- PC plus relative address$_{SE}$
```

The stack pointer is adjusted so that data will not be overwritten. The program counter is then pushed onto the stack and

the relative address sign extended and
added to the program counter.

BRA -- BRanch Always
PC <-- PC plus relative address$_{SE}$
The relative address is sign extended and
added to the program counter.

I/O instructions

DO -- Do Output
Device <-- AC
The accumulator will output its data to an
output device.

DI -- Do Input
AC <-- Data
The accumulator will receive data from an
input device.

8.3 Bit Formats

Instructions are either two or four bytes in length with the opcode always occupying the first byte. The second byte fulfills a variety of roles and the third and fourth bytes provide either an address or data. Details are now given.

All memory reference instructions must occupy four bytes and Fig. 8.1 illustrates the prescribed bit format. As previously mentioned, the first byte specifies the opcode. Please note that the first four bits are all set to 1's. This signals to the CPU that a memory reference

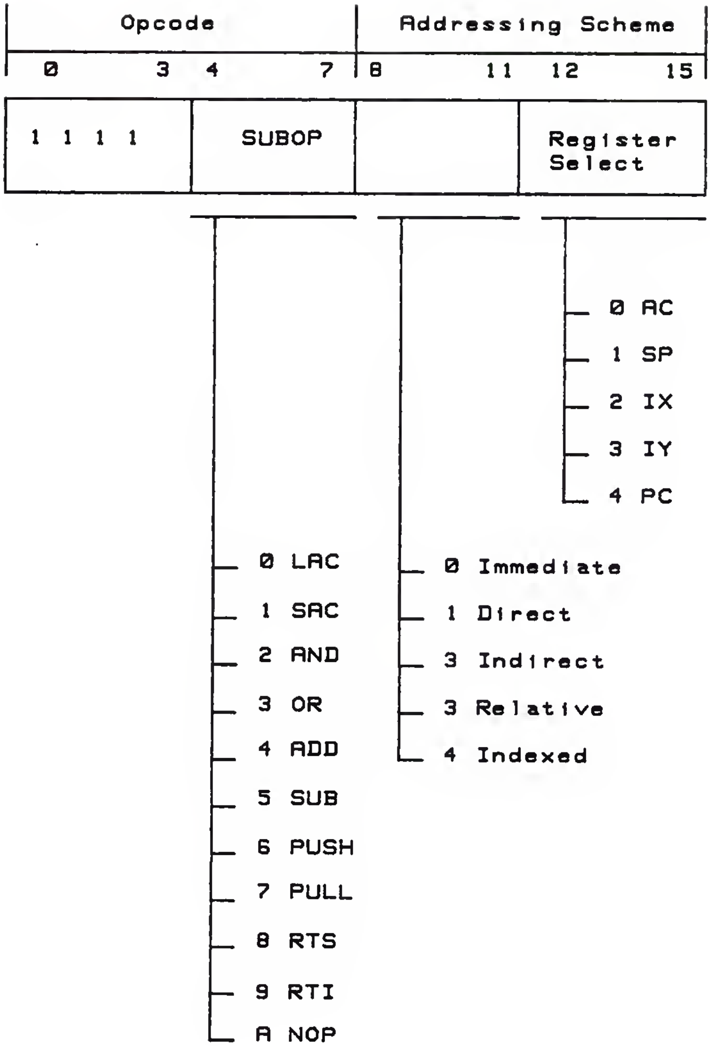


Figure 8.1 Bit Format for Memory Reference Instructions

instruction is being executed and that regardless of the opcode, an ea calculation needs to be computed prior to the execution of the instruction. This leaves four bits in which 16 memory reference instructions may be defined.

The second byte specifies the addressing scheme and the register to be used while the third and fourth bytes provide either a data value or a relative address. *Data* is provided by bytes three and four for immediate addressing, the *address of the data* for direct addressing, the *address of the address of the data* for indirect addressing, the *data to be added to the PC* for relative addressing and the *register to be indexed off of* for indexed addressing. Examples of ea calculation for all of these addressing schemes are given in appendix A.

Register reference instructions are two bytes in length and their format is pictured in Fig. 8.2. A register reference instruction is specified by setting the first four bits of the opcode to 0001_2. The last four bits in the opcode indicate which specific operation to perform.

The second byte is also divided into two fields of four bits each, and they determine the source register (bits 8 - 11) and the destination register (bits 12 - 15) for the instruction. These are loaded directly from the IR into the ASEL and BSEL fields of the 2901 via mode 11_2 as described previously in chapter four.

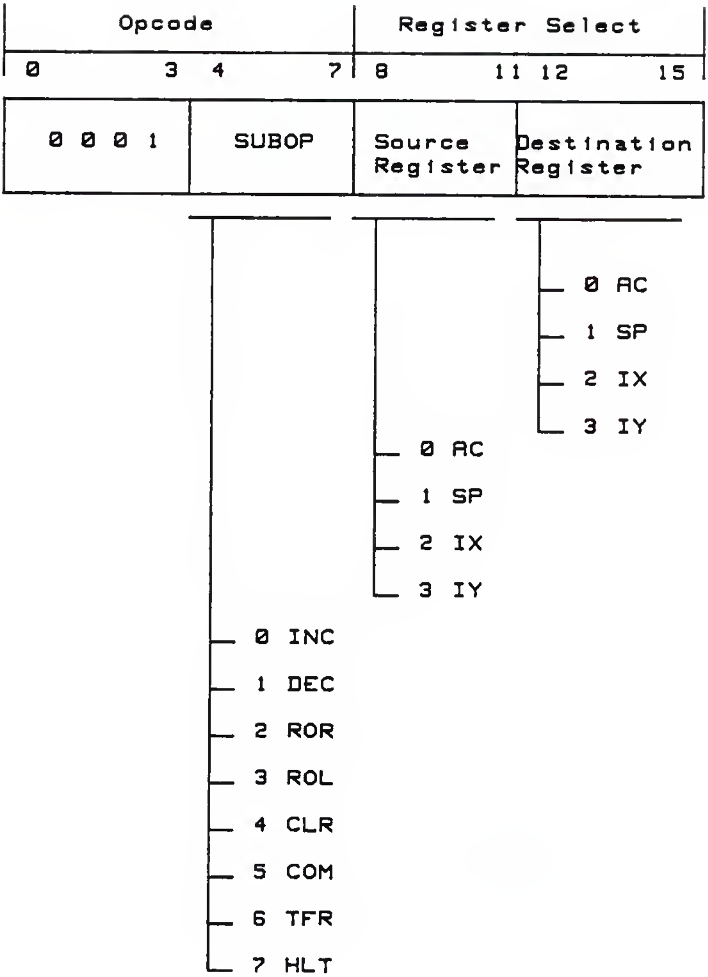


Figure 8.2 Bit Format for Register Reference Instructions

The third format, illustrated in Fig. 8.3 is for branching instructions. A branch instruction is indicated by setting the first four bits of the opcode to 0010_2 and allowing bits 4 - 7 to define the specific branch desired.

The second byte of a branch instruction provides a relative address which must be sign extended and added to the PC. The sign extension capability is performed via the hardware designed by Dr. Hush and simply converts a byte into a full 16-bit word. This is accomplished by copying the most significant bit of the 8-bit byte into the most significant byte of the newly formed 16-bit word!!

The final instruction format, viewed in Fig. 8.4 is for the I/O instructions. The first seven bits must be cleared to 0 with the least significant bit indicating whether an input or output command is being issued.

The second byte contains the device code to which the data is to be either written to in the case of an output command, or read from in the case of an input command. 2^8 combinations allow 256 I/O devices to be connected to the system.

At this point a few words of warning seem appropriate. The instruction set allows the programmer many capabilities and careful attention must be paid to the detail in specifying each task. For example, the instruction set permits the programmer to clear the SP. Doing this in the

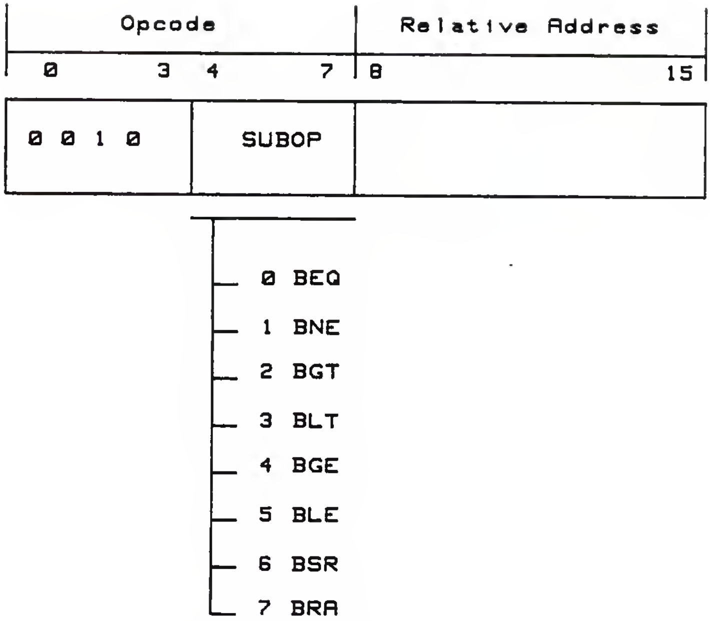


Figure 8.3 Bit Format for Branch Instructions

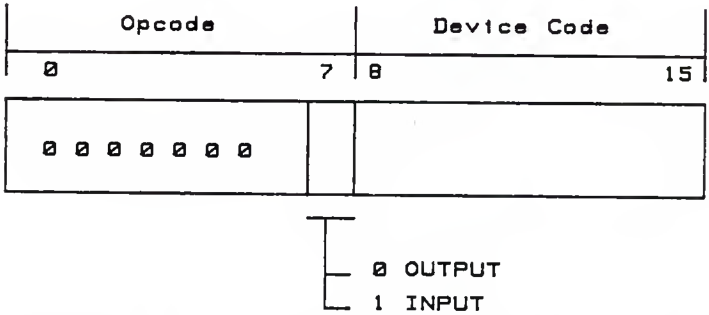


Figure 8.4 Bit Format for I/O Instructions

vast majority of cases would prove detrimental to further execution and successful completion of the program, yet allowing this to occur is facilitated just in case the need ever should arise. Also, rotating a given register and then storing the result in a different register is possible. The occasions where this is necessary are probably more common than clearing the SP, but a frequent use of this capability certainly isn't anticipated.

In summary, the instruction set has been given along with a breakdown of the various bit formats. The function of each byte of the four types of instructions, memory reference, register reference, branch and I/O has been defined and subsequently clarified.

9 -- The ILL

9.1 Purpose of the ILL

To accommodate the generation of the control bits, an Intermediate Language Level (ILL) instruction set was developed, simply referred to from now on as "ILL." The utility of the ILL lies in the realization that it aids in bridging the conceptual gap present between the instruction set and the 1's and 0's. The ILL lends the microprogrammer an additional tool for microprogramming the machine and should facilitate breaking down the macroinstructions into their various microinstructions.

The ILL is a set of 42 commands which are used to decompose the macroinstructions into their respective control bits and the particular ILL developed was named "Later Daze!" Once these control bits are generated, they require only to be stored in the CS to obtain a complete microprogrammed machine. The power of an ILL is manifested once the microprogrammer becomes familiar enough with it that he can think and program using the ILL. Without an ILL, he must jump directly from the macroinstruction level to the 1's and 0's which define the microinstruction. The existence of an ILL will fill this conceptual deficiency and thus act as an interagent between these two levels of thought.

The particular ILL developed assumes a pseudo-assembly language format, which should prove to be fairly easy for the microprogrammer to become accustomed to. A thorough understanding of each ILL command is essential for fully utilizing the convenience and power of such a tool.

As one last reminder before the ILL commands are introduced, it is stated that they strongly resemble assembly language commands; in fact, some of them are the exact same. However, care must be taken to distinguish between assembly language commands and ILL commands; they are in nowise identical! An assembly language command is a member of the instruction set, a.k.a. a macroinstruction. A macroinstruction is performed by execution of one or more microinstructions, a.k.a. ILL commands. Each microinstruction is represented by a unique ILL command.

9.2 ILL Description

ACTOBUS -- BUS <-- AC
The accumulator is transferred to the BUS. This instruction is used for I/O purposes.

ADD -- AC <-- AC plus D plus C_{in}
The accumulator receives the sum of what is currently in the accumulator plus the D inputs of the 2901 (this will usually be the result of an off-chip access to memory) plus the value of the CARRY bit.

AND -- AC <-- AC AND D
The accumulator receives the result of a logical AND operation performed on

the contents of the accumulator with the D inputs of the 2901.

CLR -- (A 2901 reg.) <-- 0
The specified register will be cleared, i.e. loaded with zeros.

COM -- (A 2901 reg.) <-- COM(A 2901 reg.)
The destination register receives the complement of the source register.

DATATOAC -- AC <-- data
Data is read from memory and loaded into the accumulator. This instruction is used for I/O purposes.

DCRTOBUS -- BUS <-- DCR
The device code register is transferred to the BUS.

DEC -- (A 2901 reg.) <-- (A 2901 reg.) minus 1
The destination register receives the result of decrementing the source register by 1.

FORPC -- PC <-- forced data
The program counter is loaded with a forced data value. For this particular implementation, the beginning address of the macroinstruction interrupt servicing routine is assumed to be at location 100_{10}. This instruction is used when an interrupt occurs.

HLT -- ----------------
The current instruction is a no-operation instruction and execution is halted.

INC -- (A 2901 reg.) <-- (A 2901 reg.) plus 1
The destination register is loaded with the result of incrementing the source register by 1.

INCIASM -- SP <-- SP plus 2
MAR <-- SP_{new}
Interrupt Acknowledge
The stack pointer is incremented by 2 and this result is transferred to the memory address register. The interrupt

is then acknowledged.

INCPCMAR -- PC <-- PC plus 2
MAR <-- PC_{new}
The program counter is incremented by 2 and this result is transferred to the memory address register.

INCSPMAR -- SP <-- SP plus 2
MAR <-- SP_{new}
The stack pointer is incremented by 2 and this result is transferred to the memory address register.

INIT -- PC <-- 0
The program counter is loaded with zeros.

LAC -- AC <-- M[ea]
The accumulator is loaded with the data read from memory.

LIRPCINC -- IR <-- M[ea]
PC <-- PC plus 2
The instruction register is loaded with the data read from memory and the program counter is incremented by 2.

LPC -- PC <-- M[ea]
The program counter is loaded with the data read from memory.

LPCRS -- PC <-- M[ea]
Restore status bits
The program counter is loaded with the data read from memory and the status bits are restored.

LTEMP1 -- TEMP1 <-- M[ea]
The TEMP1 register is loaded with the data read from memory.

NOP -- -------------
No operation is performed.

OR -- AC <-- AC OR D
The accumulator receives the result of

a logical OR operation performed on the contents of the accumulator with the D inputs of the 2901.

PCREL -- PC <-- PC plus rel. addr.$_{SE}$
The program counter receives the result of adding the program counter with the sign extended relative address.

PCTEMP1 -- TEMP1 <-- PC
The TEMP1 register is loaded with the data currently in the program counter.

ROL -- (A 2901 reg.) <-- ROL(A 2901 reg.)
The destination register receives the result of the source register being rotated left 1 bit. The CARRY bit is included.

ROR -- (A 2901 reg.) <-- ROR(A 2901 reg.)
The destination register receives the result of the source register being rotated right 1 bit. The CARRY bit is included.

SAC -- M[ea] <-- AC
The accumulator is written to memory.

SPC -- M[SP] <-- PC
The program counter is written to the location of memory where the stack pointer is pointing.

SPMARDEC -- MAR <-- SP
SP <-- SP minus 2
The stackpointer is transferred to the memory address register and then decremented.

SUB -- AC <-- AC minus D
The accumulator receives the result of subtracting D from the accumulator. The CARRY is included.

TEMAR -- MAR <-- TEMP1
The memory address register is loaded with the data in the TEMP1 register.

TEMPADD -- TEMP1 <-- TEMP1 plus

(A 2901 reg.)
The TEMP1 register receives the result of adding the TEMP1 register with the specified register.

TEMPC -- TEMP1 <-- TEMP1 plus PC
The TEMP1 register receives the result of adding the TEMP1 register with the program counter.

TFRIA -- (A 2901 reg.) <-- (A 2901 reg.)
The destination register receives the contents of the source register. The source register is specified in the IR and the destination register by the BSEL field in the PLR. This instruction is used when indexed addressing is being used.

TFRRR -- (A 2901 reg.) <-- (A 2901 reg.)
The destination register receives the contents of the source register. Both the source and destination registers are specified in the IR.

TINT -- MAR <-- PC plus 2
Test for interrupt
The program counter is incremented by 2 and transferred to the memory address register. The interrupt status flag is tested.

TNV0 -- PC <-- (PC plus rel. addr.$_{SE}$ /PC plus 0)
Test for "N XOR V" = 0
Depending on the result of the test condition, the program counter receives the result of one of two sums: either the program counter plus the sign extended relative address or the program counter plus zero.

TNV0NC -- Test for "N XOR V" = 0
The "N XOR V" status flag is tested to see if it is clear. No computations are performed.

TNV1 -- PC <-- (PC plus rel. addr.$_{SE}$ /PC plus 0)
Test for "N XOR V" = 1
Depending on the result of the test

condition, the program counter receives one of two sums: either the program counter plus the sign extended relative address, or the program counter plus zero.

TNV1NC -- Test for "N XOR V" = 1
The "N XOR V" status flag is tested to see if it is set. No computations are performed.

TZ0 -- PC <-- (PC plus rel. addr.$_{SE}$
/PC plus 0)
Test for "Z" = 0
Depending on the result of the test condition, the program counter receives one of two sums: either the program counter plus the sign extended relative address, or the program counter plus zero.

TZ1 -- PC <-- (PC plus rel. addr.$_{SE}$
/PC plus 0)
Test for "Z" = 1
Depending on the result of the test condition, the program counter receives one of two sums: either the program counter plus the sign extended relative address, or the program counter plus zero.

The same warnings mentioned at the end of chapter eight also apply here. The ILL gives the microprogrammer much power when coding microroutines, but this strength also introduces the possibility of severe errors. For instance, it is possible to increment a certain register but store it in a different register! Likewise, a register may be transferred to itself. Suffice it to say that a solid understanding of each ILL command and its exact function are vital to effective microprogramming.

9.3 ILL Control Bit Specifications

Each individual ILL command requires a specific set of control bits to be set or cleared. These control bits are related to the CPU, the BUS, and sometimes memory and I/O devices depending on the particular command.

With the exception of the seven test commands, the control bits driving the 2910 are independent of the ILL commands. That is, the control bits associated with the next address generation part of the microinstruction are completely independent from the "action" part. As a result, a complete microinstruction is a concatenation of two types of control bits: those generated by the particular ILL and those necessary for the next address generation. This should be intuitively obvious since an operation is performed with no prior knowledge as to the location of the succeeding instruction.

With this in mind, the control bits for each ILL command are now listed. Unless specified otherwise, all value are given in binary. Clarifications are given in section 9.4 for various operations that differ in their method of transferring data. Note that these are not complete microinstructions. Complete microinstructions, ILL commands with the addition of the next address control bits, are presented in appendix A.

ACTOBUS

2901		
I_{5-0}	= 000100	A plus 0
C_{in}	= 0	"Carry in" = 0
0/1	= 0	
$M_{1,0}$	= 00	Normal mode
ASEL	= 0011	AC
I_{8-6}	= 000	Don't store result
BUS		
$BS_{1,0}$	= 01	Result to I/O
I/O'SEL	= 1	as data
I/O S/$\overline{D}$	= 0	

ADD

2901		
I_{5-0}	= 000101	D plus A plus C_{in}
C_{in}	= 1	C_{in} from CARRY bit
SS	= 1	Set status bits
BUS	= 1	D inputs of 2901 come from BUS
$M_{1,0}$	= 00	Normal mode
ASEL	= 0011	AC
I_{8-6}	= 010	Result to AC
BSEL	= 0011	
BUS		
MEMSEL	= 1	Read from memory
R/$\overline{W}$	= 1	

AND

2901		
I_{5-0}	= 100101	D AND A
C_{in}	= 0	"Carry in" = 0
0/1	= 0	
SS	= 1	Set status bits
BUS	= 1	D inputs of 2901 come from BUS
$M_{1,0}$	= 00	Normal mode
ASEL	= 0011	AC
I_{8-6}	= 010	Result to AC
BSEL	= 0011	
BUS		
MEMSEL	= 1	Read from memory
R/$\overline{W}$	= 1	

CLR

2901		
I_{5-0}	= 000111	D plus 0
C_{in}	= 0	"Carry in" = 0
0/1	= 0	
SS	= 1	Set status bits
$M_{1,0}$	= 11	Reg. ref. mode
ASEL	= 0011 1110 1010 1011	AC SP IX IY
BUS	= 0	0 to D inputs
$DS_{1,0}$	= 00	
Immed.	= 00000000	
I_{8-6}	= 010	Result to
BSEL	= 0011 1110 1010 1011	AC SP IX IY
BUS		
$BS_{1,0}$	= 00	Nothing to BUS

COM

2901		
I_{5-0}	= 110101	D XOR A
C_{in}	= 0	"Carry in" = 0
0/1	= 0	
SS	= 1	Set status bits
$M_{1,0}$	= 11	Reg. ref. mode
ASEL	= 0011	AC
	1110	SP
	1010	IX
	1011	IY
BUS	= 0	FF_{16} to D inputs
$DS_{1,0}$	= 00	
Immed.	= 11111111	
I_{8-6}	= 010	Result to
BSEL	= 0011	AC
	1110	SP
	1010	IX
	1011	IY
BUS		
$BS_{1,0}$	= 00	Nothing to BUS

DATATOAC

2901		
I_{5-0}	= 000111	D plus 0
C_{in}	= 0	"Carry in" = 0
0/1	= 0	
$M_{1,0}$	= 00	Normal mode
BUS	= 1	D inputs from BUS
I_{8-6}	= 010	Result to AC
BSEL	= 0011	
BUS		
I/O SEL	= 1	Data from input device onto BUS
I/O S/$\overline{D}$	= 0	

DCRTOBUS

2901		
I_{5-0}	= 000100	A plus 0
C_{in}	= 0	"Carry in" = 0
0/1	= 0	
$M_{1,0}$	= 00	Normal mode
ASEL	= 0000	DCR
I_{8-6}	= 000	Don't store result
BUS		
$BS_{1,0}$	= 01	Result to I/O
I/O'SEL	= 1	device as a
		device code
I/O S/$\overline{D}$	= 0	

DEC

2901		
I_{5-0}	= 001101	A minus D
C_{in}	= 0	"Carry in" = 1
0/1	= 1	
SS	= 1	Set status bits
$M_{1,0}$	= 11	Reg. ref. mode
ASEL	= 0011	AC
	1110	SP
	1010	IX
	1011	IY
BUS	= 0	1 to D inputs
$DS_{1,0}$	= 00	
Immed.	= 00000001	
I_{8-6}	= 010	Result to
BSEL	= 0011	AC
	1110	SP
	1010	IX
	1011	IY
BUS		
$BS_{1,0}$	= 00	Nothing to BUS

FORPC

2901		
I_{5-0}	= 000111	D plus 0
C_{in}	= 0	"Carry in" = 0
0/1	= 0	
$M_{1,0}$	= 00	Normal mode
BUS	= 0	100_{10} to D inputs
$DS_{1,0}$	= 00	
Immed.	= 01100100	
I_{8-6}	= 011	Result to PC
BSEL	= 1001	
BUS		
$BS_{1,0}$	= 10	Write CPU status to memory
MEMSEL	= 1	
R/$\overline{W}$	= 0	

HLT

All control bits set to 0	

INC

<u>2901</u>		
I_{5-0}	= 000101	D plus A
C_{in}	= 0	"Carry in" = 0
0/1	= 0	
SS	= 1	Set status bits
$M_{1,0}$	= 11	Reg. ref. mode
ASEL	= 0011	AC
	1110	SP
	1010	IX
	1011	IY
BUS	= 0	1 to D inputs
$DS_{1,0}$	= 00	
Immed.	= 00000001	
I_{8-6}	= 010	Result to
BSEL	= 0011	AC
	1110	SP
	1010	IX
	1011	IY
<u>BUS</u>		
$BS_{1,0}$	= 00	Nothing to BUS

```
INCIASM
----------------------------------------------------------------
              |  2901                        |
              |  I5-0       = 000101         |  D plus A
              |  Cin        = 0              |  "Carry in" = 0
              |  0/1        = 0              |
              |                              |
              |  M1,0       = 00             |  Normal mode
              |  ASEL       = 1110           |  SP
              |                              |
              |  BUS        = 0              |  2 to D inputs
              |  DS1,0      = 00             |
              |  Immed      = 00000010       |
              |                              |
              |  I8-6       = 011            |  Y <-- F output
              |  BSEL       = 1110           |  SP
              |                              |
              |  BUS                         |
              |  BS1,0      = 01             |  Y output to BUS
              |  LDMAR      = 1              |
              |  INTACK     = 1              |
```

INCPCMAR

<u>2901</u>		
I_{5-0}	= 000101	D plus A
C_{in}	= 0	"Carry in" = 0
0/1	= 0	
$M_{1,0}$	= 00	Normal mode
ASEL	= 1001	PC
BUS	= 0	2 to D inputs
$DS_{1,0}$	= 00	
Immed.	= 00000010	
I_{8-6}	= 011	Result to PC
BSEL	= 1001	
<u>BUS</u>		
$BS_{1,0}$	= 01	Y output to BUS
LDMAR	= 1	

INCSPMAR

<u>2901</u>		
I_{5-0}	= 000101	D plus A
C_{in}	= 0	"Carry in" = 0
0/1	= 0	
$M_{1,0}$	= 00	Normal mode
ASEL	= 1110	SP
BUS	= 0	2 to D inputs
$DS_{1,0}$	= 00	
Immed.	= 00000010	
I_{8-6}	= 011	Y <-- F output
BSEL	= 1110	SP
<u>BUS</u>		
$BS_{1,0}$	= 01	Y output to BUS
LDMAR	= 1	

INIT

2901		
I_{5-0}	= 000111	D plus 0
C_{in}	= 0	"Carry in" = 0
0/1	= 0	
$M_{1,0}$	= 00	Normal mode
BUS	= 0	0 to D inputs
$DS_{1,0}$	= 00	
Immed.	= 00000000	
I_{8-6}	= 010	Result to PC
BSEL	= 1001	
BUS		
$BS_{1,0}$	= 00	Nothing to BUS

LAC

2901		
I_{5-0}	= 000111	D plus 0
C_{in}	= 0	"Carry in" = 0
0/1	= 0	
SS	= 1	Set status bits
BUS	= 1	D inputs of 2901 come from BUS.
$M_{1,0}$	= 00	Normal mode
I_{8-6}	= 010	Result to AC
BSEL	= 0011	
BUS		
MEMSEL	= 1	Read from memory
$R/\overline{W}$	= 1	

LIRPCINC

2901		
I_{5-0}	= 000101	A plus D
C_{in}	= 0	"Carry in" = 0
0/1	= 0	
$M_{1,0}$	= 00	Normal mode
ASEL	= 1001	PC
BUS	= 0	2 to D inputs
$DS_{1,0}$	= 00	
Immed.	= 00000010	
I_{8-6}	= 010	Result to PC
BSEL	= 1001	
BUS		
MEMSEL	= 1	Read value from memory into IR
$R/\overline{W}$	= 1	
LDIR	= 1	

LPC

2901		
I_{5-0}	= 000111	D plus 0
C_{in}	= 0	"Carry in" = 0
0/1	= 0	
BUS	= 1	D inputs of 2901 come from BUS
$M_{1,0}$	= 00	Normal mode
I_{8-6}	= 010	Result to PC
BSEL	= 1001	
BUS		
MEMSEL	= 1	Read from memory
$R/\overline{W}$	= 1	

LPCRS

2901		
I_{5-0}	= 000111	D plus 0
C_{in}	= 0	"Carry in" = 0
0/1	= 0	
BUS	= 1	D inputs of 2901 come from BUS
$M_{1,0}$	= 00	Normal mode
I_{8-6}	= 010	Result to PC
BSEL	= 1001	
BUS		
MEMSEL	= 1	Read from memory
$R/\overline{W}$	= 1	
RS	= 1	Restore status bits

LTEMP

2901		
I_{5-0}	= 000111	D plus 0
C_{in}	= 0	"Carry in" = 0
0/1	= 0	
$M_{1,0}$	= 00	Normal mode
BUS	= 1	D inputs from BUS
I_{8-6}	= 010	Result to TEMP1
BSEL	= 0100	
BUS		
MEMSEL	= 1	Read from memory
$R/\overline{W}$	= 1	

NOP

All control bits = 0	

OR

2901		
I_{5-0}	= 011101	D OR A
C_{in}	= 0	"Carry in" = 0
0/1	= 0	
SS	= 1	Set status bits
BUS	= 1	D inputs of 2901 come from BUS
$M_{1,0}$	= 00	Normal mode
ASEL	= 0011	AC
I_{8-6}	= 010	Result to AC
BSEL	= 0011	
BUS		
MEMSEL	= 1	Read from memory
R/$\overline{W}$	= 1	

PCREL

2901		
I_{5-0}	= 000101	D plus A
C_{in}	= 0	"Carry in" = 0
0/1	= 0	
$M_{1,0}$	= 00	Normal mode
ASEL	= 1001	PC
BUS	= 0	2nd byte of IR
$DS_{1,0}$	= 10	to D inputs
SE	= 1	Sign extend
I_{8-6}	= 010	Result to PC
BSEL	= 1001	
BUS		
$BS_{1,0}$	= 00	Nothing to BUS

PCTEMP1

2901		
I_{5-0}	= 000100	A plus 0
C_{in}	= 0	"Carry in" = 0
0/1	= 0	
$M_{1,0}$	= 00	Normal mode
ASEL	= 1001	PC
I_{8-6}	= 010	Result to TEMP1
BSEL	= 0100	
BUS		
$BS_{1,0}$	= 00	Nothing to BUS

ROL

2901		
I_{5-0}	= 000100	A plus 0
C_{in}	= 1	Include CARRY
SS	= 1	Set status bits
$M_{1,0}$	= 11	Reg. ref. mode
ASEL	= 0011	AC
	1110	SP
	1010	IX
	1011	IY
I_{8-6}	= 110	ROL - Result to
BSEL	= 0011	AC
	1110	SP
	1010	IX
	1011	IY
BUS		
$BS_{1,0}$	= 00	Nothing to BUS

ROR

<u>2901</u>		
I_{5-0}	= 000100	A plus 0
C_{in}	= 1	Include CARRY
SS	= 1	Set status bits
$M_{1,0}$	= 11	Reg. ref. mode
ASEL	= 0011	AC
	1110	SP
	1010	IX
	1011	IY
I_{8-6}	= 100	ROR -- Result to
BSEL	= 0011	AC
	1110	SP
	1010	IX
	1011	IY
<u>BUS</u>		
$BS_{1,0}$	= 00	Nothing to BUS

SAC

<u>2901</u>		
I_{5-0}	= 000100	A plus 0
C_{in}	= 0	"Carry in" = 0
0/1	= 0	
SS	= 1	Set status bits
$M_{1,0}$	= 00	Normal mode
ASEL	= 0011	AC
I_{8-6}	= 000	Y <-- F output
		Don't store result
<u>BUS</u>		
MEMSEL	= 1	Write to memory
$R/\overline{W}$	= 0	

SPC

2901		
I_{5-0}	= 000100	A plus 0
C_{in}	= 0	"Carry in" = 0
0/1	= 0	
$M_{1,0}$	= 00	Normal mode
ASEL	= 1001	PC
I_{8-6}	= 000	Don't store result
BUS		
$BS_{1,0}$	= 01	Write PC to memory
MEMSEL	= 1	
R/$\overline{W}$	= 0	

SPMARDEC

2901		
I_{5-0}	= 001101	A minus D
C_{in}	= 0	"Carry in" = 1
0/1	= 1	
$M_{1,0}$	= 00	Normal mode
ASEL	= 1110	SP
BUS	= 0	2 to D inputs
$DS_{1,0}$	= 00	
Immed.	= 00000010	
I_{8-6}	= 010	Y <-- A select
BSEL	= 1110	Result to SP
BUS		
$BS_{1,0}$	= 01	A select to BUS
LDMAR	= 1	

SUB

2901		
I_{5-0}	= 001101	A minus D
C_{in}	= 1	Consider C_{in}
SS	= 1	Set status bits
BUS	= 1	D inputs of 2901 come from BUS
$M_{1,0}$	= 00	Normal mode
ASEL	= 0011	AC
I_{8-6}	= 010	Result to AC
BSEL	= 0011	
BUS		
MEMSEL	= 1	Read from memory
$R/\overline{W}$	= 1	

TEMAR

2901		
I_{5-0}	= 000100	A plus 0
C_{in}	= 0	"Carry in" = 0
0/1	= 0	
$M_{1,0}$	= 00	Normal mode
ASEL	= 0100	TEMP1
I_{8-6}	= 000	Y <-- F output Don't store result
BUS		
$BS_{1,0}$	= 01	Y output to BUS
LDMAR	= 1	

TEMPADD

<u>2901</u>		
I_{5-0}	= 000001	A plus B
C_{in}	= 0	"Carry in" = 0
$0/\overline{1}$	= 0	
$M_{1,0}$	= 10	Index mode
ASEL	= 0011	AC
	1110	SP
	1010	IX
	1011	IY
BSEL	= 0100	TEMP1
I_{8-6}	= 010	Store result
<u>BUS</u>		
$BS_{1,0}$	= 00	Nothing to BUS

TEMPC

<u>2901</u>		
I_{5-0}	= 000001	A plus B
C_{in}	= 0	"Carry in" = 0
$0/\overline{1}$	= 0	
$M_{1,0}$	= 00	Normal mode
ASEL	= 1001	PC
BSEL	= 0100	TEMP1
I_{8-6}	= 010	Store result
<u>BUS</u>		
$BS_{1,0}$	= 00	Nothing to BUS

TFRIA

2901		
I_{5-0}	= 000100	A plus 0
C_{in}	= 0	"Carry in" = 0
0/1	= 0	
SS	= 1	Set status bits
$M_{1,0}$	= 10	Ind. addr. mode
ASEL	= 0011	AC
I_{8-6}	= 010	Result to
BSEL	= 0011	AC
	1110	SP
	1010	IX
	1011	IY
BUS		
$BS_{1,0}$	= 00	Nothing to BUS

TFRRR

2901		
I_{5-0}	= 000100	A plus 0
C_{in}	= 0	"Carry in" = 0
0/1	= 0	
SS	= 1	Set status bits
$M_{1,0}$	= 11	Reg. ref. mode
ASEL	= 0011	AC
	1110	SP
	1010	IX
	1011	IY
I_{8-6}	= 010	Result to
BSEL	= 0011	AC
	1110	SP
	1010	IX
	1011	IY
BUS		
$BS_{1,0}$	= 00	Nothing to BUS

TINT

2910		
T_{3-0}	= 0101	Test for interrupt
2901		
I_{5-0}	= 000101	A plus D
C_{in}	= 0	"Carry in" = 0
0/1	= 0	
$M_{1,0}$	= 00	Normal mode
ASEL	= 1001	PC
BUS	= 0	2 to D inputs
$DS_{1,0}$	= 00	
Immed.	= 00000010	
I_{8-6}	= 000	Don't store result
BUS		
$BS_{1,0}$	= 01	Y output to BUS
LDMAR	= 1	

TNVO

2910		
T_{3-0}	= 0111	Test for "N XOR V"
2901		
I_{5-0}	= 000101	D plus A
C_{in}	= 0	"Carry in" = 0
0/1	= 0	
$M_{1,0}$	= 00	Normal mode
ASEL	= 1001	PC
BUS	= 0	2nd byte of IR
$DS_{1,0}$	= 10	to D inputs
SE	= 1	Sign extend
TE	= 1	Enable test to
		force D inputs to 0
I_{8-6}	= 010	Result to PC
BSEL	= 1001	
BUS		
$BS_{1,0}$	= 00	Nothing to BUS

TNV0NC

2910		
T_{3-0}	= 0111	Test for "N XOR V"
All other control bits = 0		

TNV1

2910		
T_{3-0}	= 0111	Test for "N XOR V"
2901		
I_{5-0}	= 000101	D plus A
C_{in}	= 0	"Carry in" = 0
0/1	= 0	
$M_{1,0}$	= 00	Normal mode
ASEL	= 1001	PC
BUS	= 0	2nd byte of IR
$DS_{1,0}$	= 10	to D inputs
SE	= 1	Sign extend
TE	= 1	Enable test to force D inputs to 0
I_{8-6}	= 010	Result to PC
BSEL	= 1001	
BUS		
$BS_{1,0}$	= 00	Nothing to BUS

TNV1NC

2910		
T_{3-0}	= 0111	Test for "N XOR V"
All other control bits = 0		

TZO

2910		
T_{3-0}	= 0100	Test for "Z"
2901		
I_{5-0}	= 000101	D plus A
C_{in}	= 0	"Carry in" = 0
0/1	= 0	
$M_{1,0}$	= 00	Normal mode
ASEL	= 1001	PC
BUS	= 0	2nd byte of IR
$DS_{1,0}$	= 10	to D inputs
SE	= 1	Sign extend
TE	= 1	Enable test to
		force D inputs to 0
I_{8-6}	= 010	Result to PC
BSEL	= 1001	
BUS		
$BS_{1,0}$	= 00	Nothing to BUS

TZ1

2910		
T_{3-0}	= 0100	Test for "Z"
2901		
I_{5-0}	= 000101	D plus A
C_{in}	= 0	"Carry in" = 0
0/1	= 0	
$M_{1,0}$	= 00	Normal mode
ASEL	= 1001	PC
BUS	= 0	2nd byte of IR
$DS_{1,0}$	= 10	to D inputs
SE	= 1	Sign extend
TE	= 1	Enable test to force D inputs to 0
I_{8-6}	= 010	Result to PC
BSEL	= 1001	
BUS		
$BS_{1,0}$	= 00	Nothing to BUS

9.4 ILL Examples

When writing the microcode and decomposing the macroinstructions into their respective ILL commands, a certain thought process must be adopted, and five examples of this thought process are now given. This identical procedure must be completed for every instruction in the instruction set, and the examples were intentionally chosen to illustrate the control bit generation approach for instructions with diverse data flows. Note -- looking at

Figs. 4.1 and 5.1 may be helpful while stepping through these examples.

LAC -- In order to load the AC, an access to memory must be made. It is assumed that the ea has already been computed and is currently in the TEMP1 register. The first thing that needs to be done is to send the data in TEMP1, i.e. the ea to the MAR. This is the ILL command TEMAR. Once this is accomplished, the AC needs to be loaded with the data to which the MAR is referring. This is the LAC ILL command.

ADD -- This instruction also requires an access to memory and as before, the ea is assumed to be located in TEMP1. Transferring the ea to the MAR is necessary, so TEMAR is called. The next microinstruction must add the data read to the AC and store the result in the AC. This is the ADD ILL command.

PUSH -- The value in the AC is to be pushed on the stack. The SP is adjusted so that it always points to valid data, therefore implying that it must be incremented before pushing new data onto it. This incremented SP value is then sent to the MAR, and this is done by the ILL command INCSPMAR. Now the data in the AC must be stored to where the MAR is pointing. This is simply the SAC ILL command.

INC -- This example is practically trivial. The given register is incremented. Obviously, the INC ILL command is requested.

BEQ -- This instruction requests a branch if the result of a previous operation was equal to zero. The "Z" status flag must therefore be tested to see if it is set. The resulting ILL command is TZ1.

Several of these examples required two ILL commands to complete their execution. These commands need to be executed sequentially, hence placing them sequentially in the CS is the logical arrangement. The next address part of all the "last" microinstructions would invoke a jump to the microinstruction routine that fetches the next macroinstruction. For this particular implementation project, this was CS location 1000_8.

10 -- Control Bit Summary

Remembering and attempting to keep the numerous control bits from avalanching into a hopeless state of meaningless 1's and 0's certainly may seem to be a sizeable task. For this reason, a summary of all the control bits is now given with an explanation of their function.

10.1 Next Address Generation (2910) Control Bits

I_{3-0} -- These four bits determine which instruction the 2910 will execute. They are summarized in Table 5.1, page 28.

T_{3-0} -- If a status flag needs to be tested for a conditional jump instruction, these four bits indicate which status flag to test. They are summarized in Table 5.2, page 31.

POL -- This bit allows a test condition to be performed for either positive or negative polarity.
0 ==> Positive polarity
1 ==> Negative polarity

$S_{1,0}$ -- These bits determine where the D inputs to the 2910 will come from. See Table 5.4, page 32.

NA -- This is the Next Address field. This is a 12-bit field which supplies a potential address for the CS.

10.2 Arithmetic Logic Unit (2901) Control Bits

I_{2-0} -- These three bits determine eight possible combinations for the ALU source operands. These bits are summarized in Table 4.1, page 15.

I_{5-3} -- These three bits determine eight possible combinations for the ALU function. These bits are summarized in Table 4.2, page 15.

C_{in} -- This bit indicates whether the carry in value comes from the CARRY bit or from 0/1.
0 ==> Carry comes from 0/1
1 ==> Carry comes from CARRY bit

0/1 -- This bit functions as a forced set or clear for C_{in}.

$M_{1,0}$ -- These two bits determine four modes in which the ALU can operate. See Table 4.6, page 20.

ASEL -- This four-bit field selects one of the 16 RAM locations to be fed into the A inputs of the 2901 ALU. See Table 4.5, page 20.

BSEL -- This four-bit field selects one of the 16 RAM locations to be fed into the B inputs of the 2901 ALU. BSEL also functions as the address of the destination register. See Table 4.5, page 20.

SE -- This bit indicates if the D inputs to the 2901 should be sign extended.
0 ==> Do not sign extend.
1 ==> Sign extend.

TE -- This bit allows the test result to force the D inputs to zero.
0 ==> Do not allow force to occur
1 ==> Allow force to occur

SS -- This bit allows the status flags to be set from the resulting ALU operation.
0 ==> Do not allow status flags to be set
1 ==> Allow status flags to be set

BUS -- If set, this bit indicates that the D inputs of the 2901 will come from the BUS. Otherwise, the D inputs are determined by $DS_{1,0}$.

$DS_{1,0}$ -- Assuming BUS is clear, these two bits determine the source of the D inputs to the 2901. See Table 4.9, page 22.

Immed. -- If $DS_{1,0}$ is 00, Immed. supplies an immediate value for the D inputs to the 2901.

I_{8-6} -- These bits determine the destination of the result of the ALU operation as well as whether it should be shifted or not. Finally, the source of the Y output is determined. The function of these bits is summarized in Table 4.4, page 18.

10.3 BUS Control Bits

$BS_{1,0}$ -- These two bits determine what data is transferred to the BUS. Table 4.8, page 22 summarizes these control bits.

LDMAR -- This bit indicates that the MAR is to be loaded with the value currently on the BUS.
0 ==> Do not load MAR
1 ==> Load MAR

LDIR -- This bit indicates that the IR is to be loaded with the value currently on the BUS.
0 ==> Do not load IR
1 ==> Load IR

MEMSEL -- This bit indicates that a transfer to or from memory is to take place.
0 ==> Memory is not selected
1 ==> Memory is selected

R/$\overline{W}$ -- This bit specifies whether a read or a write operation is to occur.
0 ==> Write
1 ==> Read

I/O SEL -- This bit determines if a transfer to or from an I/O device is to take place.
0 ==> No I/O transfer
1 ==> I/O transfer

I/O S/$\overline{D}$ -- When I/O SEL is set, this bit determines whether the status of a device is requested or if data is being sent.
0 ==> Data is being sent
1 ==> Device status is being requested

INTACK -- This bit acknowledges that an I/O device is prompting to be serviced.
0 ==> No acknowledgment
1 ==> Acknowledge interrupt

10.4 Control Bit Layout

Sections 10.1 through 10.3 listed the control bits. These bits are arranged as illustrated in Fig. 10.1, and this particular ordering was chosen to parallel Dr. Hush's machine as much as possible.

Figure 10.1 Control Bit Organization

0 1	2-13	14-17	18	19-22	23-30	31 32	33	34	35	36-39	40-43
S(1,0)	NA	I(3-0)	POL	$(3-0)	IMMEDIATE	DS(1,0)	BUS	SE	TE	ASEL	BSEL

44 45	46-48	49-54	55	56	57	58 59	60	61	62	63	64	65	66	67
M(1,0)	I(8-6)	I(5-0)	0/1	C(in)	SS	BS(1,0)	LDIR	MEMSEL	R/W	LDMAR	I/O SEL	I/O B/D	INTACK	RS

11 -- Control Store

The Control Store (CS) is a 4k X 67-bit ROM in which the microcode is stored. The scheme for determining the location of each microinstruction is arbitrary and completely at the discretion of the individual microprogrammer, but the method prescribed by Dr. Hush appeared straightforward and was adopted.

The macroinstruction execution "start area" is locations 2000_8 through 2999_8. The exact location for each instruction was determined by the particular opcode. The LAC macroinstruction for example, has the opcode value of 11110000_2. Reading left to right in groups of 3 bits (octal conversion), it is found that location 2740_8 is where the first microinstruction needed to execute the LAC macroinstruction resides. Likewise, SUB has an opcode value of 11110101_2, resulting in location 2752_8 housing the first microinstruction, and BNE with an opcode of 00100001_2 starts at location 2102_8.

Recall that when a memory reference instruction is invoked, the ea must first be calculated. Locations 3000_8 through 3077_8 are reserved for these computations. Relative addressing has values of $0011XXXX_2$ for bits 8 through 15. Reading as described above, 3014_8 emerges as the location.

Location 1000_8 was chosen to be where the FETCH routine which fetches the next macroinstruction is located. Once again, 1000_8 was purely arbitrary as long as the microprogrammer remains consistent throughout the stages of his coding.

The microcode section for the interrupt servicing routine is stored in locations 0400_8 through 0403_8 and the initialization routine is at location 0000_8 in the CS.

The CS memory map summarizing the above outline, is pictured in Fig. 11.1, and all of the locations not specifically reserved remain available to the microprogrammer for various routines deemed necessary.

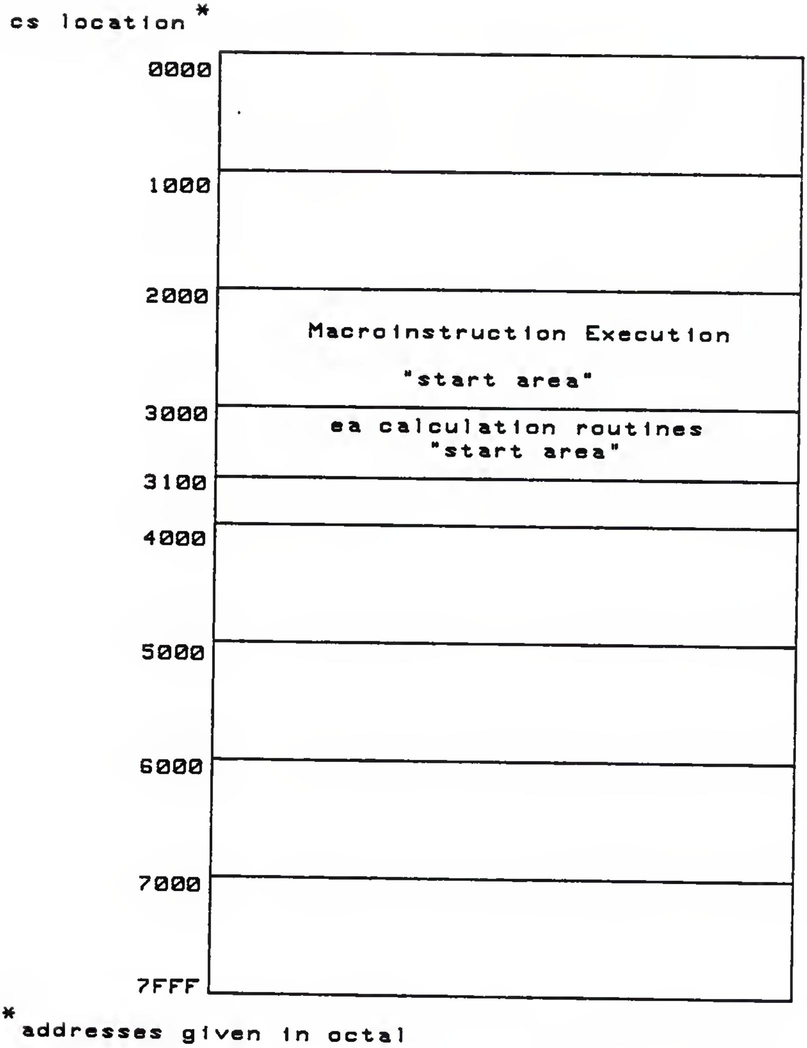


Figure 11.1 Control Store Memory Map

12 -- An Implementation Aid

12.1 Existence of a Tool

In an attempt to relieve the microprogrammer from an excessive amount of work with 1's and 0's, a tool to aid in the process of microcoding was developed. This tool is a computer program written in C, which will generate the control bits from a minimal amount of data. Only the ILL command, the CS location and five other parameters must be specified. The advantage of this program however, is that the last five elements are all related to the next address generation, and this removes the microprogrammer entirely from the data flow part of the microinstruction. This substantially raises the conceptual level: the microprogrammer needs only to determine what instruction will reside in each CS location and a few parameters dealing with testing conditions; every control bit related to the data flow is generated automatically and thus appears as a black box.

12.2 Parameters to be Specified

As stated above, the ILL command and its location in the CS must be specified. Also required for successful implementation of the program are the following:

1. it must be known if the next microinstruction to be executed is stored in the next

sequential CS location or if it needs to be jumped to;

2. it must be known where the D inputs to the 2910 originate. See Table 5.4, page 32. In the vast majority of cases, these inputs will be located in the NA field of the microinstruction register. As a matter of fact, the only time this will not apply is for ea calculations and the macroinstruction fetch routine, both of which are written only once. Therefore, unless a specific microcode routine is being written, a value of 0 for this parameter will be used for all instructions;

3. the NA field must be known. This involves no more than deciding the location for each microinstruction;

4. if a conditional jump instruction, i.e. a "test" ILL command is being executed, then the appropriate status flag must be specified. See Table 5.2, page 31; and

5. assuming a test instruction, the polarity must be determined.

Once these parameters have been determined, the microprogrammer needs only to decide the CS location for each microinstruction; the program receives this data, generates the complete microinstruction and stores it in the given CS location.

Appendix B contains the input file which the program reads for the particular instruction set presented in this paper. The following format is required:

ILL command -- j_or_ns -- s -- NA -- flag -- pol -- cs_loc.

The order of the input data is purely arbitrary.

The output of the program is a memory dump to a file and the memory dump for this instruction set may be viewed in appendix C. Please note that only the memory locations with relevant data are listed.

12.3 Dissection of the Program

The computer program, a highlevel flowchart of which appears in Fig. 12.1, consists of a main function and three "sub-functions." The main function calls these other functions to perform specific tasks.

The main function prompts the user for the name of the input and output files. The input file must of course already exist. This is not so with the output file however; the output file may exist, but if it does not, C will create it.

The clear function which clears, i.e. sets to 0 all the control bits is then called. The control bits need to be cleared each time so that a control bit that has been set for one ILL command will not carry over into the next ILL command's generation process. The clear function receives as its parameters all of the control bits.

One line of the input file is then read. This line contains the information discussed in section 12.2.

The generate function which performs the actual control bit generation is then called. Internal to this function is a mass of case and nested case statements which

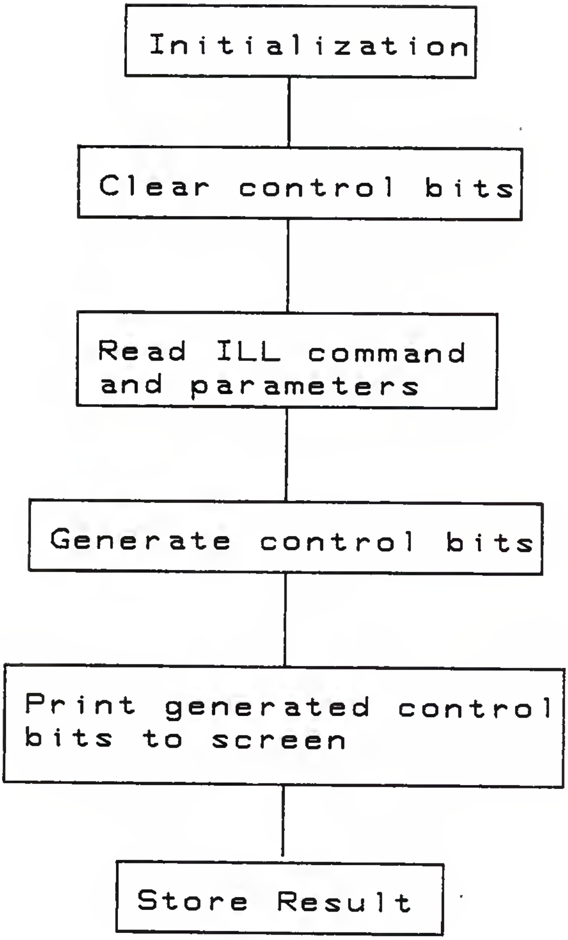

Initialization
Clear control bits
Read ILL command and parameters
Generate control bits
Print generated control bits to screen
Store Result

Figure 12.1 High-level Flowchart

parse the ILL command. Once the command has been parsed and the appropriate control bits for the data flow generated, the control bits for the next address generation part of the microinstruction are produced. This involves more case statements, but these are to a lesser degree of complexity.

Control is then returned to the main function which outputs the generated bits to the screen. This allows the user to verify the control bits if he wishes.

The main function then calls store() which stores the control bits in the specified CS location. The sequencing of the control bit fields is illustrated in Fig. 10.1.

The final operation performed by the main function is a memory dump to the output file.

The main, clear, generate and store functions appear in appendix D. They have been documented and commented quite extensively, so following the logic and sequence of operations should pose no problem for the reader.

13 -- Future Work

An abundant amount of work for the future is provided by this machine. Chronologically, the next reasonable step would be to physically build the machine and implement the given microcode. Enhancing the system by upgrading the memory, using the 2903 or 29203 rather than the 2901, etc. would be favorable. Applications to the classroom are practically limitless. Students could construct parts of the hardware as group efforts, interface and subsequently integrate these parts into a functional whole. Once the hardware aspect is completed, vast amounts of software could be written, both at the assembly language level and at the ILL. Additionally, different instruction sets could be implemented, which could lead to a study dealing with the characteristics of various instruction sets.

If a less ambitious or an individual project is desired, an excellent exercise in microprogramming could be obtained by writing the microcode for a different instruction set. Doing this would certainly bring to light many of the finer aspects involved with microprogramming a machine that are oftentimes overlooked in the classroom.

14 -- Conclusion

The primary focus of this research was to develop a workable knowledge of a 16-bit stack machine designed by Dr. Don Hush and to implement the control unit of a specific instruction set via microcode. To this end an Intermediate Level Language (ILL) was devised which represented the various required data transfers and was utilized when undergoing the breakdown of the macroinstructions.

Each ILL command dictates a unique set of control bits in completing the "action part" of a microinstruction and a program was then written which simulated this process of control bit generation.

The original objectives of verifying that an instruction set could actually be implemented on Dr. Hush's machine, providing a simple, illustrative example of microcoding, and producing a tool in the form of a computer program were all met.

References

1. "The Design Proposal of a 16-bit Microprogrammed Stack Machine", Hush, Don Rhea, Kansas State University, 1982.

2. Bipolar Microprocessor Logic and Interface, 1985 Data Book.

3. Computer Design, Langdon, Glen G., Jr., Computeach Press Inc., San Jose, CA, 1982.

4. Structured Computer Organization, Tanenbaum, Andrew S., Prentice Hall, Inc., Englewood Cliffs, N.J. 1984.

Appendix A -- Control Bits for Macroinstructions

This appendix contains an exhaustive list of the macroinstructions and the control bits required to execute them. Perhaps the most opportune method of totally understanding the precise function of each control bit and how it relates with and affects the flow of data is simply to study the following examples. Comments have been provided alongside each control bit specification to guide the reader and clear up any potential confusion. Careful study of these examples is strongly recommended.

The macroinstruction is presented first, followed by the needed algorithm. Along with any assumptions made, one will also find the necessary ILL commands, their location in the CS and an English description of what is to take place during each microinstruction. Finally, the various control bits that need to be set will be presented in tabular form.

As a step to simplify the task of the microprogrammer, the machine was designed by Dr. Hush so that a control bit was activated by setting it. (This is opposed to clearing it.) Hence, any active low control bits were routed through an inverter prior to being loaded into the PLR.

Unless otherwise stated, all control bits are assumed to be in binary form and to be clear; only those bits required to be set for proper execution of the instruction are given. One may notice however, that occasionally a control bit is specified to be 0, e.g. BUS = 0. This was done for the sake of uniformity and was believed that including this bit specification would enhance the readability.

LAC

```
 0          3    4          7
 ------------------------------
|            |                 |
|  1  1  1  1 |  0  0  0  0    |
|            |                 |
 ------------------------------
  mem. ref.  |       LAC
```

Algorithm
AC <-- M[ea]

Assumptions
1. The ea lies in TEMP1
2. Opcode = 111/100/00
   CS location 2740_8

Location	ILL Command	Description
2740_8	TEMAR	* Send TEMP1 to MAR * Fetch next sequential microinstruction
2741_8	LAC	* Read from memory to AC * Jump to FETCH routine at CS 1000_8 via NA field

Assembly

Location	Microinstruction		Comments
2740_8	2910		
	I_{3-0}	= 1110	Continue
	2901		
	I_{5-0}	= 000100	A plus 0
	C_{in}	= 0	"Carry in" = 0
	0/1	= 0	
	$M_{1,0}$	= 00	Normal mode
	ASEL	= 0100	TEMP1
	I_{8-6}	= 000	Y <-- F output Don't store result
	BUS		
	$BS_{1,0}$	= 01	Y output to BUS
	LDMAR	= 1	
2741_8	2910		
	I_{3-0}	= 0010	JMAP
	$S_{1,0}$	= 00	FETCH next uinstr.
	NA	= 1000_8	at CS 1000_8
	2901		
	I_{5-0}	= 000111	D plus 0
	C_{in}	= 0	"Carry in" = 0
	0/1	= 0	
	SS	= 1	Set status bits
	BUS	= 1	D inputs of 2901 come from BUS
	$M_{1,0}$	= 00	Normal mode
	I_{8-6}	= 010	Result to AC
	BSEL	= 0011	
	BUS		
	MEMSEL	= 1	Read from memory
	$R/\overline{W}$	= 1	

SAC

```
 0        3   4        7
 ------------------------------
|             |                |
|  1  1  1  1 |  0  0  0  1    |
|             |                |
 ------------------------------
  mem. ref.   |     SAC
```

Algorithm
M[ea] <-- AC

Assumptions
1. The ea lies in TEMP1
2. Opcode = 111/100/01
   CS location 2742_8

Location	ILL Command	Description
2742_8	TEMAR	* Send TEMP1 to MAR * Fetch next sequential microinstruction
2743_8	SAC	* Write AC to memory * Jump to FETCH routine at CS 1000_8 via NA field

Assembly

Location	Microinstruction		Comments
2742_8	2910		
	I_{3-0}	= 1110	Continue
	2901		
	I_{5-0}	= 000100	A plus 0
	C_{in}	= 0	"Carry in" = 0
	0/1	= 0	
	$M_{1,0}$	= 00	Normal mode
	ASEL	= 0100	TEMP1
	I_{8-6}	= 000	Y <-- F output
			Don't store result
	BUS		
	$BS_{1,0}$	= 01	Y output to BUS
	LDMAR	= 1	
2743_8	2910		
	I_{3-0}	= 0010	JMAP
	$S_{1,0}$	= 00	FETCH next uinstr.
	NA	= 1000_8	at CS 1000_8
	2901		
	I_{5-0}	= 000100	A plus 0
	C_{in}	= 0	"Carry in" = 0
	0/1	= 0	
	SS	= 1	Set status bits
	$M_{1,0}$	= 00	Normal mode
	ASEL	= 0011	AC
	I_{8-6}	= 000	Y <-- F output
			Don't store result
	BUS		
	MEMSEL	= 1	Write to memory
	$R/\overline{W}$	= 0	

AND

```
  0          3    4           7
 ---------------------------------
|                  |               |
|  1   1   1   1   |  0   0   1   0 |
|                  |               |
 ---------------------------------
  mem. ref.        |     AND
```

Algorithm

AC <-- AC AND M[ea]

Assumptions

1. The ea lies in TEMP1
2. Opcode = 111/100/10
   CS location 2744_8

Location	ILL Command	Description
2744_8	TEMAR	* Send TEMP1 to MAR * Fetch next sequential microinstruction
2745_8	AND	* Perform AND operation * Jump to FETCH routine at CS 1000_8 via NA field

Assembly Location	Microinstruction		Comments
2744_8	2910		
	I_{3-0}	= 1110	Continue
	2901		
	I_{5-0}	= 000100	A plus 0
	C_{in}	= 0	"Carry in" = 0
	0/1	= 0	
	$M_{1,0}$	= 00	Normal mode
	ASEL	= 0100	TEMP1
	I_{8-6}	= 000	Y <-- F output
			Don't store result
	BUS		
	$BS_{1,0}$	= 01	Y output to BUS
	LDMAR	= 1	
2745_8	2910		
	I_{3-0}	= 0010	JMAP
	$S_{1,0}$	= 00	FETCH next uinstr.
	NA	= 1000_8	at CS 1000_8
	2901		
	I_{5-0}	= 100101	D AND A
	C_{in}	= 0	"Carry in" = 0
	0/1	= 0	
	SS	= 1	Set status bits
	BUS	= 1	D inputs of 2901
			come from BUS
	$M_{1,0}$	= 00	Normal mode
	ASEL	= 0011	AC
	I_{8-6}	= 010	Result to AC
	BSEL	= 0011	
	BUS		
	MEMSEL	= 1	Read from memory
	$R/\overline{W}$	= 1	

OR

```
  0        3    4        7
 ------------------------------
|              |               |
|  1  1  1  1  |  0  0  1  1   |
|              |               |
 ------------------------------
  mem. ref.    |      OR
```

Algorithm
AC <-- AC OR M[ea]

Assumptions

1. The ea lies in TEMP1
2. Opcode = 111/100/11
   CS location 2746_8

Location	ILL Command	Description
2746_8	TEMAR	* Send TEMP1 to MAR * Fetch next sequential microinstruction
2747_8	OR	* Perform OR operation * Jump to FETCH routine at CS 1000_8 via NA field

Assembly			
Location	Microinstruction		Comments
2746_8	2910		
	I_{3-0}	= 1110	Continue
	2901		
	I_{5-0}	= 000100	A plus 0
	C_{in}	= 0	"Carry in" = 0
	0/1	= 0	
	$M_{1,0}$	= 00	Normal mode
	ASEL	= 0100	TEMP1
	I_{8-6}	= 000	Y <-- F output
			Don't store result
	BUS		
	$BS_{1,0}$	= 01	Y output to BUS
	LDMAR	= 1	
2747_8	2910		
	I_{3-0}	= 0010	JMAP
	$S_{1,0}$	= 00	FETCH next uinstr.
	NA	= 1000_8	at CS 1000_8
	2901		
	I_{5-0}	= 011101	D OR A
	C_{in}	= 0	"Carry in" = 0
	0/1	= 0	
	SS	= 1	Set status bits
	BUS	= 1	D inputs of 2901
			come from BUS
	$M_{1,0}$	= 00	Normal mode
	ASEL	= 0011	AC
	I_{8-6}	= 010	Result to AC
	BSEL	= 0011	
	BUS		
	MEMSEL	= 1	Read from memory
	$R/\overline{W}$	= 1	

ADD

```
  0          3    4          7
 ---------------------------------
|           |                    |
|  1  1  1  1  |  0  1  0  0     |
|           |                    |
 ---------------------------------
   mem. ref.   |      ADD
```

Algorithm

AC <-- AC plus M[ea]

Assumptions

1. The ea lies in TEMP1
2. Opcode = 111/101/00 CS location 2750_8

Location	ILL Command	Description
2750_8	TEMAR	* Send TEMP1 to MAR * Fetch next sequential microinstruction
2751_8	ADD	* Perform ADD operation * Jump to FETCH routine at CS 1000_8 via NA field

Assembly

Location	Microinstruction		Comments
2750_8	2910		
	I_{3-0}	= 1110	Continue
	2901		
	I_{5-0}	= 000100	A plus 0
	C_{in}	= 0	"Carry in" = 0
	0/1	= 0	
	$M_{1,0}$	= 00	Normal mode
	ASEL	= 0100	TEMP1
	I_{8-6}	= 000	Y <-- F output
			Don't store result
	BUS		
	$BS_{1,0}$	= 01	Y output to BUS
	LDMAR	= 1	
2751_8	2910		
	I_{3-0}	= 0010	JMAP
	$S_{1,0}$	= 00	FETCH next uinstr.
	NA	= 1000_8	at CS 1000_8
	2901		
	I_{5-0}	= 000101	D plus A plus C_{in}
	C_{in}	= 1	C_{in} from CARRY bit
	SS	= 1	Set status bits
	BUS	= 1	D inputs of 2901
			come from BUS
	$M_{1,0}$	= 00	Normal mode
	ASEL	= 0011	AC
	I_{8-6}	= 010	Result to AC
	BSEL	= 0011	
	BUS		
	MEMSEL	= 1	Read from memory
	R/$\overline{W}$	= 1	

SUB

```
 0        3    4        7
 ----------------------------
|                |            |
| 1  1  1  1     | 0  1  0  1 |
|                |            |
 ----------------------------
 mem. ref.       |    SUB
```

Algorithm
AC <-- AC minus M[ea]

Assumptions
1. The ea lies in TEMP1
2. Opcode = 111/101/01
   CS location 2752_8

Location	ILL Command	Description
2752_8	TEMAR	* Send TEMP1 to MAR * Fetch next sequential microinstruction
2753_8	SUB	* Perform SUB operation * Jump to FETCH routine at CS 1000_8 via NA field

Assembly

Location	Microinstruction	Comments
2752_8	2910	
	I_{3-0} = 1110	Continue
	2901	
	I_{5-0} = 000100	A plus 0
	C_{in} = 0	"Carry in" = 0
	0/1 = 0	
	$M_{1,0}$ = 00	Normal mode
	ASEL = 0100	TEMP1
	I_{8-6} = 000	Y <-- F output
		Don't store result
	BUS	
	$BS_{1,0}$ = 01	Y output to BUS
	LDMAR = 1	
2753_8	2910	
	I_{3-0} = 0010	JMAP
	$S_{1,0}$ = 00	FETCH next uinstr.
	NA = 1000_8	at CS 1000_8
	2901	
	I_{5-0} = 001101	A minus D
	C_{in} = 1	Consider CARRY bit
	SS = 1	Set status bits
	BUS = 1	D inputs of 2901
		come from BUS
	$M_{1,0}$ = 00	Normal mode
	ASEL = 0011	AC
	I_{8-6} = 010	Result to AC
	BSEL = 0011	
	BUS	
	MEMSEL = 1	Read from memory
	R/$\overline{W}$ = 1	

PUSH

```
  0           3    4          7
 ------------------------------
|           |                  |
|  1  1  1  1  |  0  1  1  0  |
|           |                  |
 ------------------------------
  mem. ref.    |     PUSH
```

Algorithm

```
   SP <-- SP plus 2
M[SP] <-- AC
```

Assumptions

1. The SP must be incremented before going to MAR.
2. Opcode = 111/101/10 CS location 2754_8

Location	ILL Command	Description
2754_8	INCSPMAR	* Increment the SP and send result to MAR * Fetch next sequential microinstruction
2755_8	SAC	* Write AC to memory * Jump to FETCH routine at CS 1000_8 via NA field

Assembly

Location	Microinstruction	Comments
2754_8	<u>2910</u>	
	I_{3-0} = 1110	Continue
	<u>2901</u>	
	I_{5-0} = 000101	D plus A
	C_{in} = 0	"Carry in" = 0
	0/1 = 0	
	$M_{1,0}$ = 00	Normal mode
	ASEL = 1110	SP
	BUS = 0	2 to D inputs
	$DS_{1,0}$ = 00	
	Immed. = 00000010	
	I_{8-6} = 011	Y <-- F output
	BSEL = 1110	Store result
	<u>BUS</u>	
	$BS_{1,0}$ = 01	Y output to BUS
	LDMAR = 1	
2755_8	<u>2910</u>	
	I_{3-0} = 0010	JMAP
	$S_{1,0}$ = 00	FETCH next uinstr.
	NA = 1000_8	at CS 1000_8
	<u>2901</u>	
	I_{5-0} = 000100	A plus 0
	C_{in} = 0	"Carry in" = 0
	0/1 = 0	
	SS = 1	Set status bits
	$M_{1,0}$ = 00	Normal mode
	ASEL = 0011	AC
	I_{8-6} = 000	Y <-- F output
		Don't store result
	<u>BUS</u>	
	MEMSEL = 1	Write to memory
	$R/\overline{W}$ = 0	

PULL

```
  0           3    4           7
 -----------------------------
|                |             |
|  1   1   1   1 |  0  1  1  1 |
|                |             |
 -----------------------------
  mem. ref.      |     PULL
```

Algorithm

AC <-- M[SP]
SP <-- SP minus 2

Assumptions

1. The SP must be decremented after reading its value
2. Opcode = 111/101/11
   CS location 2756_8

Location	ILL Command	Description
2756_8	SPMARDEC	* Send SP to MAR * Decrement SP by 2 * Fetch next sequential microinstruction
2757_8	LAC	* Read memory into AC * Jump to FETCH routine at CS 1000_8 via NA field

Assembly Location	Microinstruction		Comments
2756_8	2910		
	I_{3-0}	= 1110	Continue
	2901		
	I_{5-0}	= 001101	A minus D
	C_{in}	= 0	"Carry in" = 1
	0/1	= 1	
	$M_{1,0}$	= 00	Normal mode
	ASEL	= 1110	SP
	BUS	= 0	2 to D inputs
	$DS_{1,0}$	= 00	
	Immed.	= 00000010	
	I_{8-6}	= 010	Y <-- A select
	BSEL	= 1110	Result to SP
	BUS		
	$BS_{1,0}$	= 01	A select to BUS
	LDMAR	= 1	
2757_8	2910		
	I_{3-0}	= 0010	JMAP
	$S_{1,0}$	= 00	FETCH next uinstr.
	NA	= 1000_8	at CS 1000_8
	2901		
	I_{5-0}	= 000111	D plus 0
	C_{in}	= 0	"Carry in" = 0
	0/1	= 0	
	SS	= 1	Set status bits
	BUS	= 1	D inputs of 2901 come from BUS
	$M_{1,0}$	= 00	Normal mode
	I_{8-6}	= 010	Result to AC
	BSEL	= 0011	
	BUS		
	MEMSEL	= 1	Read from memory
	$R/\overline{W}$	= 1	

RTS

```
     0           3    4          7
 ---------------------------------
|              |                  |
|  1  1  1  1  |  1   0   0   0   |
|              |                  |
 ---------------------------------
   mem. ref.   |       RTS
```

Algorithm
PC <-- M[SP]
SP <-- SP minus 2

Assumptions
1. Opcode = 111/110/00
   CS location 2760_8

Location	ILL Command	Description
2760_8	SPMARDEC	* Send SP to MAR * Decrement SP by 2 * Fetch next sequential microinstruction
2761_8	LPC	* Read memory into PC * Jump to FETCH routine at CS 1000_8 via NA field

Assembly			
Location	Microinstruction		Comments
2760_8	2910		
	I_{3-0}	= 1110	Continue
	2901		
	I_{5-0}	= 001101	A minus D
	C_{in}	= 0	"Carry in" = 1
	0/1	= 1	
	$M_{1,0}$	= 00	Normal mode
	ASEL	= 1110	SP
	BUS	= 0	2 to D inputs
	$DS_{1,0}$	= 00	
	Immed.	= 00000010	
	I_{8-6}	= 010	Y <-- A select
	BSEL	= 1110	Result to SP
	BUS		
	$BS_{1,0}$	= 01	A select to BUS
	LDMAR	= 1	
2761_8	2910		
	I_{3-0}	= 0010	JMAP
	$S_{1,0}$	= 00	FETCH next uinstr.
	NA	= 1000_8	at CS 1000_8
	2901		
	I_{5-0}	= 000111	D plus 0
	C_{in}	= 0	"Carry in" = 0
	0/1	= 0	
	BUS	= 1	D inputs of 2901 come from BUS
	$M_{1,0}$	= 00	Normal mode
	I_{8-6}	= 010	Result to PC
	BSEL	= 1001	
	BUS		
	MEMSEL	= 1	Read from memory
	$R/\overline{W}$	= 1	

RTI

```
  0          3    4          7
 ------------------------------
|            |                 |
|  1  1  1  1 |  1  0  0  1    |
|            |                 |
 ------------------------------
  mem. ref.  |      RTI
```

Algorithm

PC <-- M[SP]
SP <-- SP minus 2
Restore status bits

Assumptions

1. Opcode = 111/110/01
   CS location 2762_8

Location	ILL Command	Description
2762_8	SPMARDEC	* Send SP to MAR * Decrement SP by 2 * Fetch next sequential microinstruction
2763_8	LPCRS	* Read memory into PC * Restore status bits * Jump to FETCH routine at CS 1000_8 via NA field

Assembly

Location	Microinstruction		Comments
2762_8	2910		
	I_{3-0}	= 1110	Continue
	2901		
	I_{5-0}	= 001101	A minus D
	C_{in}	= 0	"Carry in" = 1
	0/1	= 1	
	$M_{1,0}$	= 00	Normal mode
	ASEL	= 1110	SP
	BUS	= 0	2 to D inputs
	$DS_{1,0}$	= 00	
	Immed.	= 00000010	
	I_{8-6}	= 010	Y <-- A select
	BSEL	= 1110	Result to SP
	BUS		
	$BS_{1,0}$	= 01	A select to BUS
	LDMAR	= 1	
2763_8	2910		
	I_{3-0}	= 0010	JMAP
	$S_{1,0}$	= 00	FETCH next uinstr.
	NA	= 1000_8	at CS 1000_8
	2901		
	I_{5-0}	= 000111	D plus 0
	C_{in}	= 0	"Carry in" = 0
	0/1	= 0	
	BUS	= 1	D inputs of 2901
			come from BUS
	$M_{1,0}$	= 00	Normal mode
	I_{8-6}	= 010	Result to PC
	BSEL	= 1001	
	BUS		
	MEMSEL	= 1	Read from memory
	$R/\overline{W}$	= 1	
	RS	= 1	Restore status bits

NOP

```
   0        3   4         7
  ------------------------------
 |                |              |
 |  1  1  1  1    |  1  0  1  0  |
 |                |              |
  ------------------------------
    mem. ref.     |     NOP
```

Algorithm

Assumptions

1. Opcode = 111/110/10
   CS location 2764_8

Location	ILL	Command	Description
2764_8		NOP	* Jump to FETCH routine at CS 1000_8 via NA field

Assembly

Location	Microinstruction	Comments
2764_8	2910 I_{3-0} = 0010 $S_{1,0}$ = 00 NA = 1000_8 All other control bits = 0	JMAP FETCH next uinstr. at CS 1000_8

INC

```
 0           3    4           7    8         11    12          15
-----------------------------------------------------------------
|                |                 |              |              |
|  0   0   0   1 |  0   0   0   0  |  Source reg. |  Dest. reg.  |
|                |                 |              |              |
-----------------------------------------------------------------
   reg. ref.     |       INC          -----------    -----------
                                      |              |
                                      |              |
                                      |              |
                                      |--3:  AC      |--3:  AC
                                      |--A:  IX      |--A:  IX
                                      |--B:  IY      |--B:  IY
                                      |--E:  SP      |--E:  SP
```

Algorithm

(A 2901 reg.) <-- (A 2901 reg.) plus 1

Assumptions

1. Opcode = 000/100/00
   CS location 2040_8

Location	ILL	Command
2040_8		INC

Description

* Appropriate register via IR 2nd byte is selected
* Perform INC command
* Jump to FETCH routine at CS 1000_8 via NA field

Assembly

Location	Microinstruction		Comments
2040_8	2910		
	I_{3-0}	= 0010	JMAP
	$S_{1,0}$	= 00	Jump to FETCH
	NA	= 1000_8	at CS 1000_8
	2901		
	I_{5-0}	= 000101	D plus A
	C_{in}	= 0	"Carry in" = 0
	0/1	= 0	
	SS	= 1	Set status bits
	$M_{1,0}$	= 11	Reg. ref. mode
	*ASEL	= 0011	AC
		1110	SP
		1010	IX
		1011	IY
	BUS	= 0	1 to D inputs
	$DS_{1,0}$	= 00	
	Immed.	= 00000001	
	I_{8-6}	= 010	Result to
	*BSEL	= 0011	AC
		1110	SP
		1010	IX
		1011	IY
	BUS		
	$BS_{1,0}$	= 00	Nothing to BUS

* Selected via hardware

DEC

```
 0         3    4         7    8         11   12        15
-----------------------------------------------------------
|                |               |             |             |
| 0   0   0   1  |  0   0   0   1 | Source reg. | Dest. reg.  |
|                |               |             |             |
-----------------------------------------------------------
 reg. ref.       |     DEC         -----------   ----------
                                   |             |
                                   |             |
                                   |             |
                                   |--3:   AC    |--3:   AC
                                   |--A:   IX    |--A:   IX
                                   |--B:   IY    |--B:   IY
                                   |--E:   SP    |--E:   SP
```

Algorithm

(A 2901 reg.) <-- (A 2901 reg.) minus 1

Assumptions

1. Opcode = 000/100/01
   CS location 2042_8

Location	ILL Command
2042_8	DEC

Description

* Appropriate register via IR 2nd byte is selected
* Perform DEC command
* Jump to FETCH routine at CS 1000_8 via NA field

Assembly

Location	Microinstruction		Comments
2042_8	2910		
	I_{3-0}	= 0010	JMAP
	$S_{1,0}$	= 00	FETCH next uinstr.
	NA	= 1000_8	at CS 1000_8
	2901		
	I_{5-0}	= 001101	A minus D
	C_{in}	= 0	"Carry in" = 1
	0/1	= 1	
	SS	= 1	Set status bits
	$M_{1,0}$	= 11	Reg. ref. mode
	*ASEL	= 0011	AC
		1110	SP
		1010	IX
		1011	IY
	BUS	= 0	1 to D inputs
	$DS_{1,0}$	= 00	
	Immed.	= 00000001	
	I_{8-6}	= 010	Result to
	*BSEL	= 0011	AC
		1110	SP
		1010	IX
		1011	IY
	BUS		
	$BS_{1,0}$	= 00	Nothing to BUS

*Selected via hardware

ROR

```
 0         3    4         7    8         11   12        15
------------------------------------------------------------
|                |                |               |              |
| 0   0   0   1  |  0   0   1   0 |  Source reg.  |  Dest. reg.  |
|                |                |               |              |
------------------------------------------------------------
  reg. ref.      |      ROR         -----------     ----------
                                    |               |
                                    |               |
                                    |               |
                                    |--3:  AC       |--3:  AC
                                    |--A:  IX       |--A:  IX
                                    |--B:  IY       |--B:  IY
                                    |--E:  SP       |--E:  SP
```

Algorithm

(A 2901 reg.) <-- ROR(A 2901 reg.)

Assumptions

1. The CARRY bit is included
2. Opcode = 000/100/10
   CS location 2044_8

Location

2044_8

ILL Command

ROR

Description

* Appropriate register via IR 2nd byte is selected
* Perform ROR operation
* Jump to FETCH routine at CS 1000_8 via NA field

Assembly

Location	Microinstruction		Comments
2044_8	2910		
	I_{3-0}	= 0010	JMAP
	$S_{1,0}$	= 00	FETCH next uinstr.
	NA	= 1000_8	at CS 1000_8
	2901		
	I_{5-0}	= 000100	A plus 0
	C_{in}	= 1	Include CARRY
	SS	= 1	Set status bits
	$M_{1,0}$	= 11	Reg. ref. mode
	*ASEL	= 0011	AC
		1110	SP
		1010	IX
		1011	IY
	I_{8-6}	= 100	ROR -- Result to
	*BSEL	= 0011	AC
		1110	SP
		1010	IX
		1011	IY
	BUS		
	$BS_{1,0}$	= 00	Nothing to BUS

*Selected via hardware

ROL

```
  0        3    4        7    8       11    12          15
 -----------------------------------------------------------
|                |               |                |               |
|  0   0   0   1 |  0   0   1   1 |  Source reg.  |  Dest. reg.   |
|                |               |                |               |
 -----------------------------------------------------------
   reg. ref.     |      ROL          -----------     ----------
                                     |                |
                                     |                |
                                     |                |
                                     |--3:   AC       |--3:   AC
                                     |--A:   IX       |--A:   IX
                                     |--B:   IY       |--B:   IY
                                     |--E:   SP       |--E:   SP
```

Algorithm

(A 2901 reg.) <-- ROL(A 2901 reg.)

Assumptions

1. The CARRY bit is included
2. Opcode = 000/100/11
   CS location 2046_8

Location	**ILL**	**Command**
2046_8		ROL

Description

* Appropriate register via IR 2nd byte is selected
* Perform ROL operation
* Jump to FETCH routine at CS 1000_8 via NA field

Assembly

Location	Microinstruction		Comments
2046_8	2910		
	I_{3-0}	= 0010	JMAP
	$S_{1,0}$	= 00	FETCH next uinstr.
	NA	= 1000_8	at CS 1000_8
	2901		
	I_{5-0}	= 000100	A plus 0
	C_{in}	= 1	Include CARRY
	SS	= 1	Set status bits
	$M_{1,0}$	= 11	Reg. ref. mode
	*ASEL	= 0011	AC
		1110	SP
		1010	IX
		1011	IY
	I_{8-6}	= 110	ROL -- Result to
	*BSEL	= 0011	AC
		1110	SP
		1010	IX
		1011	IY
	BUS		
	$BS_{1,0}$	= 00	Nothing to BUS

*Selected via hardware

CLR

```
 0         3   4         7   8        11   12          15
----------------------------------------------------------
|  0  0  0  1  |  0  1  0  0  |  Source reg. |  Dest. reg. |
|              |              |              |             |
----------------------------------------------------------
 reg. ref.     |     CLR          -----------   ----------
                                  |             |
                                  |             |
                                  |             |
                                  |--3:  AC     |--3:  AC
                                  |--A:  IX     |--A:  IX
                                  |--B:  IY     |--B:  IY
                                  |--E:  SP     |--E:  SP
```

Algorithm

(A 2901 reg.) <-- 0

Assumptions

1. Opcode = 000/101/00
   CS location 2050_8

Location	ILL	Command
2050_8		CLR

Description

- * Appropriate register via IR 2nd byte is selected
- * Perform CLR command
- * Jump to FETCH routine at CS 1000_8 via NA field

Assembly

Location	Microinstruction		Comments
2050_8	2910		
	I_{3-0}	= 0010	JMAP
	$S_{1,0}$	= 00	FETCH next uinstr.
	NA	= 1000_8	at CS 1000_8
	2901		
	I_{5-0}	= 000111	D plus 0
	C_{in}	= 0	"Carry in" = 0
	0/1	= 0	
	SS	= 1	Set status bits
	$M_{1,0}$	= 11	Reg. ref. mode
	*ASEL	= 0011	AC
		1110	SP
		1010	IX
		1011	IY
	BUS	= 0	0 to D inputs
	$DS_{1,0}$	= 00	
	Immed.	= 00000000	
	I_{8-6}	= 010	Result to
	*BSEL	= 0011	AC
		1110	SP
		1010	IX
		1011	IY
	BUS		
	$BS_{1,0}$	= 00	Nothing to BUS

*Selected via hardware

COM

```
 0           3    4            7    8          11    12          15
----------------------------------------------------------------------
|                |                |                |                 |
|  0   0   0   1 |  0   1   0   1 |  Source  reg.  |  Dest.  reg.    |
|                |                |                |                 |
----------------------------------------------------------------------
  reg.  ref.     |       COM        -----------        ----------
                                    |                  |
                                    |                  |
                                    |                  |
                                    |--3:   AC         |--3:   AC
                                    |--A:   IX         |--A:   IX
                                    |--B:   IY         |--B:   IY
                                    |--E:   SP         |--E:   SP
```

Algorithm

(A 2901 reg.) <-- COM(A 2901 reg.)

Assumptions

1. To find COM, XOR with 77_8
2. Opcode = 000/101/01 CS location 2052_8

Location	ILL	Command	Description
2052_8		COM	* Appropriate register via IR 2nd byte is selected * Perform COM operation * Jump to FETCH routine at CS 1000_8 via NA field

Assembly

Location	Microinstruction	Comments
2052_8	2910	
	I_{3-0} = 0010	JMAP
	$S_{1,0}$ = 00	FETCH next uinstr.
	NA = 1000_8	at CS 1000_8
	2901	
	I_{5-0} = 110101	D XOR A
	C_{in} = 0	"Carry in" = 0
	0/1 = 0	
	SS = 1	Set status bits
	$M_{1,0}$ = 11	Reg. ref. mode
	*ASEL = 0011	AC
	1110	SP
	1010	IX
	1011	IY
	BUS = 0	FF_{16} to D inputs
	SE = 1	
	$DS_{1,0}$ = 00	
	Immed. = 11111111	
	I_{8-6} = 010	Result to
	*BSEL = 0011	AC
	1110	SP
	1010	IX
	1011	IY
	BUS	
	$BS_{1,0}$ = 00	Nothing to BUS

*Selected via hardware

TFR

```
  0         3   4         7   8        11   12        15
 -------------------------------------------------------
|               |               |              |              |
| 0   0   0   1 | 0   1   1   0 | Source reg.  |  Dest. reg.  |
|               |               |              |              |
 -------------------------------------------------------
   reg. ref.    |     TFR         -----------    ----------
                                  |              |
                                  |              |
                                  |              |
                                  |--3:   AC     |--3:   AC
                                  |--A:   IX     |--A:   IX
                                  |--B:   IY     |--B:   IY
                                  |--E:   SP     |--E:   SP
```

Algorithm

(A 2901 reg.) <-- (A 2901 reg.)

Assumptions

1. Opcode = 000/101/10
   CS location 2054_8

Location

2054_8

ILL Command

TFRRR

Description

* Appropriate register via IR 2nd byte is selected
* Perform TFR command
* Jump to FETCH routine at CS 1000_8 via NA field

Assembly

Location	Microinstruction		Comments
2054$_8$	2910		
	I_{3-0}	= 0010	JMAP
	$S_{1,0}$	= 00	FETCH next uinstr.
	NA	= 1000$_8$	at CS 1000$_8$
	2901		
	I_{5-0}	= 000100	A plus 0
	C_{in}	= 0	"Carry in" = 0
	0/1	= 0	
	SS	= 1	Set status bits
	$M_{1,0}$	= 11	Reg. ref. mode
	*ASEL	= 0011	AC
		1110	SP
		1010	IX
		1011	IY
	I_{8-6}	= 010	Result to
	*BSEL	= 0011	AC
		1110	SP
		1010	IX
		1011	IY
	BUS		
	$BS_{1,0}$	= 00	Nothing to BUS

*Selected via hardware

HLT

0 3	4 7	8 11	12 15
0 0 0 1	0 1 1 1	X X X X	X X X X
reg. ref.	HLT		

Algorithm

Halt execution; continually jump to itself

Assumptions

1. Opcode = 000/101/11
   CS location 2056_8
2. X implies Don't Cares

Location	ILL	Command	Description
2056_8		HLT	* Make current instr. a no-operation * Test "Halt" = 0 pass - Jump to FETCH fail - Fetch next sequential microinstr.
2057_8		NOP	* Make current instr. a no-operation * Jump to CS loc. 2056_8

Assembly

Location	Microinstruction	Comments
2056_8	2910 I_{3-0} = 0011 $TEST_{3-0}$ = 1000 POL = 1 $S_{1,0}$ = 00 NA = 1000_8 All other control bits set to 0	CJP "Halt" Negative polarity Jump to FETCH at CS 1000_8
2057_8	2910 I_{3-0} = 0010 $S_{1,0}$ = 00 NA = 2056_8 All other control bits set to 0	JMAP Jump to CS loc. 2056_8

BEQ

0			3	4			7
0	0	1	0	0	0	0	0
branch				BEQ			

Algorithm

```
if "Z" = 1
then PC <-- PC plus
            rel. addr.$_{SE}$
else PC <-- PC plus 0
```

Assumptions

1. The rel. addr. is located in the IR 2nd byte
2. Opcode = 001/000/00 CS location 2100_8

Location	ILL	Command
2100_8		TZ1

Description

* Test for "Z" = 1
  pass - PC <-- PC plus
         rel. addr.$_{SE}$
  fail - PC <-- PC plus 0
* Jump to FETCH routine
  at CS 1000_8 via NA field

Assembly

Location	Microinstruction		Comments
2100_8	2910		
	I_{3-0}	= 0011	CJP
	$TEST_{3-0}$	= 0100	ZERO status flag
	POL	= 0	Positive polarity
	$S_{1,0}$	= 00	FETCH next uinstr.
	NA	= 1000_8	at CS 1000_8
	2901		
	I_{5-0}	= 000101	D plus A
	C_{in}	= 0	"Carry in" = 0
	0/1	= 0	
	$M_{1,0}$	= 00	Normal mode
	ASEL	= 1001	PC
	BUS	= 0	2nd byte of IR
	$DS_{1,0}$	= 10	to D inputs
	SE	= 1	Sign extend
	TE	= 1	Enable test to
			force D inputs to 0
	I_{8-6}	= 010	Result to PC
	BSEL	= 1001	
	BUS		
	$BS_{1,0}$	= 00	Nothing to BUS

BNE

0			3	4			7
0	0	1	0	0	0	0	1
branch				BNE			

Algorithm

```
if "Z" = 0
then PC <-- PC plus
            rel. addr.$_{SE}$
else PC <-- PC plus 0
```

Assumptions

1. The rel. addr. is located in the IR 2nd byte
2. Opcode = 001/000/01
   CS location 2102_8

Location	ILL	Command
2102_8		TZ0

Description

```
* Test for "Z" = 0
  pass - PC <-- PC plus
             rel. addr.$_{SE}$
  fail - PC <-- PC plus 0
* Jump to FETCH routine
  at CS $1000_8$ via NA field
```

Assembly

Location	Microinstruction		Comments
2102_8	2910		
	I_{3-0}	= 0011	CJP
	$TEST_{3-0}$	= 0100	ZERO status flag
	POL	= 0	Positive polarity
	$S_{1,0}$	= 00	FETCH next uinstr.
	NA	= 1000_8	at CS 1000_8
	2901		
	I_{5-0}	= 000101	D plus A
	C_{in}	= 0	"Carry in" = 0
	0/1	= 0	
	$M_{1,0}$	= 00	Normal mode
	ASEL	= 1001	PC
	BUS	= 0	2nd byte of IR
	$DS_{1,0}$	= 10	to D inputs
	SE	= 1	Sign extend
	TE	= 1	Enable test to
			force D inputs to 0
	I_{8-6}	= 010	Result to PC
	BSEL	= 1001	
	BUS		
	$BS_{1,0}$	= 00	Nothing to BUS

BGT

0			3	4			7
0	0	1	0	0	0	1	0
branch				BGT			

Algorithm

```
if "N XOR V" = 0
then
    if "Z" = 0
    then PC <-- PC plus
                rel. addr.$_{SE}$
    else PC <-- PC plus 0
else  PC <-- PC plus 0
```

Assumptions

1. The rel. addr. is located in the IR 2nd byte
2. Negative polarity is assumed for the first test
3. Opcode = 001/000/10 CS location 2104_8

Location	ILL Command	Description
2104_8	TNV0NC	* Test for "N XOR V" = 1 pass - Jump to FETCH at CS 1000_8 fail - Fetch next sequential microinstruction
2105_8	TZ0	* Test for "Z" = 0 pass - PC <-- PC plus rel. addr.$_{SE}$ fail - PC <-- PC plus 0 * Jump to FETCH routine at CS 1000_8 via NA field

Assembly

Location	Microinstruction		Comments
2104_8	2910		
	I_{3-0}	= 0011	CJP
	$TEST_{3-0}$	= 0111	N XOR V status flag
	POL	= 1	Negative polarity
	$S_{1,0}$	= 00	Jump to FETCH
	NA	= 1000_8	at CS 1000_8
	All other control bits = 0		
2105_8	2910		
	I_{3-0}	= 0011	CJP
	$TEST_{3-0}$	= 0100	ZERO status flag
	POL	= 0	Positive polarity
	$S_{1,0}$	= 00	FETCH next uinstr.
	NA	= 1000_8	at CS 1000_8
	2901		
	I_{5-0}	= 000101	D plus A
	C_{in}	= 0	"Carry in" = 0
	0/1	= 0	
	$M_{1,0}$	= 00	Normal mode
	ASEL	= 1001	PC
	BUS	= 0	2nd byte of IR
	$DS_{1,0}$	= 10	to D inputs
	SE	= 1	Sign extend
	TE	= 1	Enable test to
			force D inputs to 0
	I_{8-6}	= 010	Result to PC
	BSEL	= 1001	
	BUS		
	$BS_{1,0}$	= 00	Nothing to BUS

BLT

```
 0           3    ·4          7
 ------------------------------
|            |                 |
|  0  0  1  0 |  0  0  1  1    |
|            |                 |
 ------------------------------
    branch    |     BLT
```

Algorithm

```
if "N XOR V" = 1
then PC <-- PC plus
            rel. addr.$_{SE}$
else PC <-- PC plus 0
```

Assumptions

1. The rel. addr. is located in the IR 2nd byte
2. Opcode = 001/000/11
   CS location 2106_8

Location	ILL	Command
2106_8		TNV1

Description

* Test for "N XOR V" = 1
pass - PC <-- PC plus
rel. addr.$_{SE}$
fail - PC <-- PC plus 0

* Jump to FETCH routine at CS 1000_8 via NA field

Assembly

Location	Microinstruction		Comments
2106_8	2910		
	I_{3-0}	= 0011	CJP
	$TEST_{3-0}$	= 0111	N XOR V status flag
	POL	= 0	Positive polarity
	$S_{1,0}$	= 00	FETCH next uinstr.
	NA	= 1000_8	at CS 1000_8
	2901		
	I_{5-0}	= 000101	D plus A
	C_{in}	= 0	"Carry in" = 0
	0/1	= 0	
	$M_{1,0}$	= 00	Normal mode
	ASEL	= 1001	PC
	BUS	= 0	2nd byte of IR
	$DS_{1,0}$	= 10	to D inputs
	SE	= 1	Sign extend
	TE	= 1	Enable test to
			force D inputs to 0
	I_{8-6}	= 010	Result to PC
	BSEL	= 1001	
	BUS		
	$BS_{1,0}$	= 00	Nothing to BUS

BGE

```
 0          3    4          7
 ------------------------------
|                |              |
|  0  0  1  0    |  0  1  0  0  |
|                |              |
 ------------------------------
     branch      |     BGE
```

Algorithm

```
if "N XOR V " = 0
then PC <-- PC plus
            rel. addr.$_{SE}$
else PC <-- PC plus 0
```

Assumptions

1. The rel. addr. is located in the IR 2nd byte
2. Opcode = 001/001/00 CS location 2110_8

Location	ILL Command
2110_8	TNV0

Description

* Test for "N EOR V" = 0
pass - PC <-- PC plus rel. addr.$_{SE}$
fail - PC <-- PC plus 0

* Jump to FETCH routine at CS 1000_8 via NA field

Assembly

Location	Microinstruction		Comments
2110_8	2910		
	I_{3-0}	= 0011	CJP
	$TEST_{3-0}$	= 0111	N XOR V status flag
	POL	= 0	Positive polarity
	$S_{1,0}$	= 00	FETCH next uinstr.
	NA	= 1000_8	at CS 1000_8
	2901		
	I_{5-0}	= 000101	D plus A
	C_{in}	= 0	"Carry in" = 0
	0/1	= 0	
	$M_{1,0}$	= 00	Normal mode
	ASEL	= 1001	PC
	BUS	= 0	2nd byte of IR
	$DS_{1,0}$	= 10	to D inputs
	SE	= 1	Sign extend
	TE	= 1	Enable test to
			force D inputs to 0
	I_{8-6}	= 010	Result to PC
	BSEL	= 1001	
	BUS		
	$BS_{1,0}$	= 00	Nothing to BUS

BLE

```
 0          3   4          7
 --------------------------
|           |              |
|  0 0 1 0  |  0  1  0  1  |
|           |              |
 --------------------------
   branch   |     BLE
```

Algorithm

```
if "N XOR V" = 1
then PC <-- PC plus
            rel. addr.$_{SE}$
else
    if "Z" = 1
    then PC <-- PC plus
                rel. addr.$_{SE}$
    else PC <-- PC plus 0
```

Assumptions

1. The rel. addr. is located in the IR 2nd byte
2. Opcode = 001/001/01 CS location 2112_8

Location	ILL Command	Description
2112_8	TNV1NC	* Test for "N XOR V" = 1 pass - Jump to CS location 4000_8 fail - Fetch next sequential microinstruction
2113_8	TZ1	* Test "Z" = 1 pass - PC <-- PC plus rel. addr.$_{SE}$ fail - PC <-- PC plus 0 * Jump to FETCH routine at CS 1000_8 via NA field
4000_8	PCREL	* PC <-- PC plus rel. addr.$_{SE}$ * Jump to FETCH routine at CS 1000_8 via NA field

Assembly

Location	Microinstruction	Comments
2112_8	2910	
	I_{3-0} = 0011	CJP
	$TEST_{3-0}$ = 0111	N XOR V status flag
	POL = 0	Positive polarity
	$S_{1,0}$ = 00	Jump to CS 4000_8
	NA = 4000_8	
	All other control bits = 0	
2113_8	2910	
	I_{3-0} = 0011	CJP
	$TEST_{3-0}$ = 0100	ZERO status flag
	POL = 0	Positive polarity
	$S_{1,0}$ = 00	FETCH next uinstr.
	NA = 1000_8	at CS 1000_8
	2901	
	I_{5-0} = 000101	D plus A
	C_{in} = 0	"Carry in" = 0
	0/1 = 0	
	$M_{1,0}$ = 00	Normal mode
	ASEL = 1001	PC
	BUS = 0	2nd byte of IR
	$DS_{1,0}$ = 10	to D inputs
	SE = 1	Sign extend
	TE = 1	Enable test to force D inputs to 0
	I_{8-6} = 010	Result to PC
	BSEL = 1001	
	BUS	
	$BS_{1,0}$ = 00	Nothing to BUS

Assembly (cont.)

Location	Microinstruction		Comments
4000_8	2910		
	I_{3-0}	= 0010	JMAP
	$S_{1,0}$	= 00	FETCH next uinstr.
	NA	= 1000_8	at CS 1000_8
	2901		
	I_{5-0}	= 000101	D plus A
	C_{in}	= 0	"Carry in" = 0
	0/1	= 0	
	$M_{1,0}$	= 00	Normal mode
	ASEL	= 1001	PC
	BUS	= 0	2nd byte of IR
	$DS_{1,0}$	= 10	to D inputs
	SE	= 1	Sign extend
			force D inputs to 0
	I_{8-6}	= 010	Result to PC
	BSEL	= 1001	
	BUS		
	$BS_{1,0}$	= 00	Nothing to BUS

BSR

```
  0          3    4          7
 ------------------------------
|                |              |
|  0  0  1  0    |  0  1  1  0  |
|                |              |
 ------------------------------
     branch      |     BSR
```

Algorithm

```
   SP <-- SP plus 2
M[SP] <-- PC
   PC <-- PC plus rel. addr.$_{SE}$
```

Assumptions

1. The rel. addr. is located in the IR 2nd byte
2. Opcode = 001/001/10 CS location 2114_8

Location	ILL Command	Description
2114_8	INCSPMAR	* Increment SP by 2 and send to the MAR * Fetch next sequential microinstruction
2115_8	SPC	* Write PC to memory * Jump to CS loc. 4000_8

Assembly			
Location	Microinstruction		Comments
2114_8	2910		
	I_{3-0}	= 1110	Continue
	2901		
	I_{5-0}	= 000101	D plus A
	C_{in}	= 0	"Carry in" = 0
	0/1	= 0	
	$M_{1,0}$	= 00	Normal mode
	ASEL	= 1110	SP
	BUS	= 0	2 to D inputs
	$DS_{1,0}$	= 00	
	Immed.	= 00000010	
	I_{8-6}	= 011	Y <-- F output
	BSEL	= 1110	SP
	BUS		
	$BS_{1,0}$	= 01	Y output to BUS
	LDMAR	= 1	
2115_8	2910		
	I_{3-0}	= 0010	JMAP
	$S_{1,0}$	= 00	Jump to CS 4000_8
	NA	= 4000_8	
	2901		
	I_{5-0}	= 000100	A plus 0
	C_{in}	= 0	"Carry in" = 0
	0/1	= 0	
	$M_{1,0}$	= 00	Normal mode
	ASEL	= 1001	PC
	I_{8-6}	= 000	Don't store result
	BUS		
	$BS_{1,0}$	= 01	Write PC to memory
	MEMSEL	= 1	
	$R/\overline{W}$	= 0	

BRA

```
  0         3   4         7
 -----------------------------
|                |             |
|  0  0  1  0    |  0  1  1  1 |
|                |             |
 -----------------------------
    branch       |    BRA
```

Algorithm

PC <-- PC plus rel. addr.$_{SE}$

Assumptions

1. The rel. addr. is located in the IR 2nd byte
2. Opcode = 001/001/11
   CS location 2116_8

Location	ILL Command
2116_8	PCREL

Description

- * PC <-- PC plus rel. addr.$_{SE}$
- * Jump to FETCH routine at CS 1000_8 via NA field

Assembly

Location	Microinstruction	Comments
2116_8	2910	
	I_{3-0} = 0010	JMAP
	$S_{1,0}$ = 00	FETCH next uinstr.
	NA = 1000_8	at CS 1000_8
	2901	
	I_{5-0} = 000101	D plus A
	C_{in} = 0	"Carry in" = 0
	0/1 = 0	
	$M_{1,0}$ = 00	Normal mode
	ASEL = 1001	PC
	BUS = 0	2nd byte of IR
	$DS_{1,0}$ = 10	to D inputs
	SE = 1	Sign extend
	I_{8-6} = 010	Result to PC
	BSEL = 1001	
	BUS	
	$BS_{1,0}$ = 00	Nothing to BUS

DO

```
      0        3   4        7
     ---------------------------
    |  0  0  0  0 |  0  0  0  0 |
     ---------------------------
          I/O     |   output

 8                                                  15
 ---------------------------------------------------
|            D e v i c e        C o d e              |
 ---------------------------------------------------
```

Algorithm
Put device code on BUS
Put AC (i.e. data) on BUS

Assumptions
1. Assume DCR is valid
2. Data written from AC
3. Opcode = 000/000/00
   CS location 2000_8

Location	ILL Command	Description
2000_8	DCRTOBUS	* BUS <-- DCR * Fetch next sequential microinstruction
2001_8	ACTOBUS	* BUS <-- AC, i.e. data * Jump to FETCH routine at CS 1000_8 via NA field

Assembly

Location	Microinstruction		Comments
2000_8	2910		
	I_{3-0}	= 1110	Continue
	2901		
	I_{5-0}	= 000100	A plus 0
	C_{in}	= 0	"Carry in" = 0
	$0/\overline{1}$	= 0	
	$M_{1,0}$	= 00	Normal mode
	ASEL	= 0000	DCR
	I_{8-6}	= 000	Don't store result
	BUS		
	$BS_{1,0}$	= 01	Result to I/O as
	I/O SEL	= 1	a device code
	I/O S/$\overline{D}$	= 0	
2001_8	2910		
	I_{3-0}	= 0010	JMAP
	$S_{1,0}$	= 00	FETCH next uinstr.
	NA	= 1000_8	at CS 1000_8
	2901		
	I_{5-0}	= 000100	A plus 0
	C_{in}	= 0	"Carry in" = 0
	$0/\overline{1}$	= 0	
	$M_{1,0}$	= 00	Normal mode
	ASEL	= 0011	AC
	I_{8-6}	= 000	Don't store result
	BUS		
	$BS_{1,0}$	= 01	Result to I/O
	I/O SEL	= 1	as data
	I/O S/$\overline{D}$	= 0	

DI

0 3	4 7
0 0 0 0	0 0 0 1
I/O	input

8 15
Device Code

Algorithm
Put device code on BUS
Read data into AC from BUS

Assumptions

1. Assume DCR is valid
2. Data read via AC
3. Opcode = 000/000/01
   CS location 2002_8

Location	ILL Command	Description
2002_8	DCRTOBUS	* BUS <-- DCR * Fetch next sequential microinstruction
2003_8	DATATOAC	* AC <-- BUS, i.e. data * Jump to FETCH routine at CS 1000_8 via NA field

Assembly

Location	Microinstruction	Comments
2002_8	2910	
	I_{3-0} = 1110	Continue
	2901	
	I_{5-0} = 000100	A plus 0
	C_{in} = 0	"Carry in" = 0
	0/1 = 0	
	$M_{1,0}$ = 00	Normal mode
	ASEL = 0000	DCR
	I_{8-6} = 000	Don't store result
	BUS	
	$BS_{1,0}$ = 01	Result to I/O as
	I/O SEL = 1	a device code
	I/O S/$\overline{D}$ = 0	
2003_8	2910	
	I_{3-0} = 0010	JMAP
	$S_{1,0}$ = 00	FETCH next uinstr.
	NA = 1000_8	at CS 1000_8
	2901	
	I_{5-0} = 000111	D plus 0
	C_{in} = 0	"Carry in" = 0
	0/1 = 0	
	$M_{1,0}$ = 00	Normal mode
	BUS = 1	D inputs from BUS
	I_{8-6} = 010	Result to AC
	BSEL = 0011	
	BUS	
	I/O SEL = 1	Data from input
	I/O S/$\overline{D}$ = 0	device onto BUS

Immediate Addressing

```
              8        11   12        15
             ---------------------------
            |  0  0  0  0 |  X  X  X  X |
             ---------------------------
               Immediate  |

      16                                          31
     ----------------------------------------------
    |    I m m e d i a t e   d a t a               |
     ----------------------------------------------
```

Algorithm

TEMP1 <-- PC

Assumptions

1. Upon completion of the ea calculation, TEMP1 will contain the ea
2. X's imply Don't Cares
3. Opcode = 000/0XX/XX CS location 3000_8

Location	ILL Command
3000_8	PCTEMP1

Description

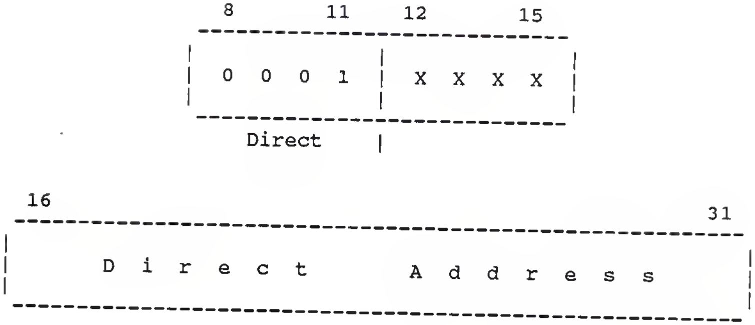
* Move PC to TEMP1

* Fetch next micro-instruction using the IR opcode address

Assembly

Location	Microinstruction	Comments
3000_8	2910	
	I_{3-0} = 0010	JMAP
	$S_{1,0}$ = 10	NA from IR opcode
	2901	
	I_{5-0} = 000100	A plus 0
	C_{in} = 0	"Carry in" = 0
	0/1 = 0	
	$M_{1,0}$ = 00	Normal mode
	ASEL = 1001	PC
	I_{8-6} = 010	Result to TEMP1
	BSEL = 0100	
	BUS	
	$BS_{1,0}$ = 00	Nothing to BUS

Direct Addressing



Algorithm

```
   PC <-- PC plus 2
TEMP1 <-- M[PC]
```

Assumptions

1. Upon completion of the ea calculation, TEMP1 will contain the ea
2. X's imply Don't Cares
3. Opcode = 000/1XX/XX CS location 3004_8

Location	ILL Command	Description
3004_8	INCPCMAR	* Increment PC by 2 * Send result to MAR * Fetch next sequential microinstruction
3005_8	LTEMP	* Read memory into TEMP1 * Fetch next microinstruction using IR opcode address

Assembly

Location	Microinstruction	Comments
3004_8	2910	
	I_{3-0} = 1110	Continue
	2901	
	I_{5-0} = 000101	D plus A
	C_{in} = 0	"Carry in" = 0
	0/1 = 0	
	$M_{1,0}$ = 00	Normal mode
	ASEL = 1001	PC
	BUS = 0	2 to D inputs
	$DS_{1,0}$ = 00	
	Immed. = 00000010	
	I_{8-6} = 011	Result to PC
	BSEL = 1001	
	BUS	
	$BS_{1,0}$ = 01	Y output to BUS
	LDMAR = 1	
3005_8	2910	
	I_{3-0} = 0010	JMAP
	$S_{1,0}$ = 10	NA from IR opcode
	2901	
	I_{5-0} = 000111	D plus 0
	C_{in} = 0	"Carry in" = 0
	0/1 = 0	
	$M_{1,0}$ = 00	Normal mode
	BUS = 1	D inputs from BUS
	I_{8-6} = 010	Result to TEMP1
	BSEL = 0100	
	BUS	
	MEMSEL = 1	Read from memory
	$R/\overline{W}$ = 1	

Indirect Addressing

```
     8         11   12          15
    -------------------------------
   |                |               |
   |  0   0   1   0 |  X   X   X   X |
   |                |               |
    -------------------------------
        Indirect    |

 16                                                31
 ---------------------------------------------------
|                                                   |
|     Indirect                 Address              |
|                                                   |
 ---------------------------------------------------
```

Algorithm

```
PC    <-- PC plus 2
TEMP1 <-- M[PC]
TEMP1 <-- M[TEMP1]
```

Assumptions

1. Upon completion of the ea calculation, TEMP1 will contain the ea
2. X's imply Don't Cares
3. Opcode = 001/0XX/XX
   CS location 3010_8

Location	ILL Command	Description
3010_8	INCPCMAR	* Increment PC by 2 * Send result to MAR * Fetch next sequential microinstruction
3011_8	LTEMP	* Read memory into TEMP1 * Fetch next sequential microinstruction
3012_8	TEMAR	* Send TEMP1 to MAR * Fetch next sequential microinstruction
3013_8	LTEMP	* Read memory into TEMP1 * Fetch next micro-instruction using IR opcode address

Assembly

Location	Microinstruction		Comments
3010_8	2910		
	I_{3-0}	= 1110	Continue
	2901		
	I_{5-0}	= 000101	D plus A
	C_{in}	= 0	"Carry in" = 0
	0/1	= 0	
	$M_{1,0}$	= 00	Normal mode
	ASEL	= 1001	PC
	BUS	= 0	2 to D inputs
	$DS_{1,0}$	= 00	
	Immed.	= 00000010	
	I_{8-6}	= 011	Result to PC
	BSEL	= 1001	
	BUS		
	$BS_{1,0}$	= 01	Y output to BUS
	LDMAR	= 1	
3011_8	2910		
	I_{3-0}	= 1110	Continue
	2901		
	I_{5-0}	= 000111	D plus 0
	C_{in}	= 0	"Carry in" = 0
	0/1	= 0	
	$M_{1,0}$	= 00	Normal mode
	BUS	= 1	D inputs from BUS
	I_{8-6}	= 010	Result to TEMP1
	BSEL	= 0100	
	BUS		
	MEMSEL	= 1	Read from memory
	$R/\overline{W}$	= 1	

Assembly (Cont.)

3012_8	2910		
	I_{3-0}	= 1110	Continue
	2901		
	I_{5-0}	= 000100	A plus 0
	C_{in}	= 0	"Carry in" = 0
	0/1	= 0	
	$M_{1,0}$	= 00	Normal mode
	ASEL	= 0100	TEMP1
	I_{8-6}	= 000	Don't store result
	BUS		
	$BS_{1,0}$	= 01	Y output to BUS
	LDMAR	= 1	
3013_8	2910		
	I_{3-0}	= 0010	JMAP
	$S_{1,0}$	= 10	NA from IR opcode
	2901		
	I_{5-0}	= 000111	D plus 0
	C_{in}	= 0	"Carry in" = 0
	0/1	= 0	
	$M_{1,0}$	= 00	Normal mode
	BUS	= 1	D inputs from BUS
	I_{8-6}	= 010	Result to TEMP1
	BSEL	= 0100	
	BUS		
	MEMSEL	= 1	Read from memory
	R/$\overline{W}$	= 1	

Relative Addressing

```
      8         11   12         15
   ----------------------------------
  |                 |                 |
  |  0   0   1   1  |  X   X   X   X  |
  |                 |                 |
   ----------------------------------
       Relative     |

 16                                                      31
 ----------------------------------------------------------
|                                                          |
|   R  e  l  a  t  i  v  e      A  d  d  r  e  s  s        |
|                                                          |
 ----------------------------------------------------------
```

Algorithm

```
   PC <-- PC plus 2
TEMP1 <-- M[PC]
TEMP1 <-- PC plus TEMP1
```

Assumptions

1. Upon completion of the ea calculation, TEMP1 will contain the ea
2. X's imply Don't Cares
3. Opcode = 001/1XX/XX CS location 3014_8

Location	ILL Command	Description
3014_8	INCPCMAR	* Increment PC by 2 * Send result to MAR * Fetch next sequential microinstruction
3015_8	LTEMP	* Read memory into LTEMP * Fetch next sequential microinstruction
3016_8	TEMPC	* Store in TEMP1 the sum of PC plus TEMP1 * Fetch next microinstruction using IR opcode address

Assembly

Location	Microinstruction		Comments
3014_8	2910		
	I_{3-0}	= 1110	Continue
	2901		
	I_{5-0}	= 000101	D plus A
	C_{in}	= 0	"Carry in" = 0
	0/1	= 0	
	$M_{1,0}$	= 00	Normal mode
	ASEL	= 1001	PC
	BUS	= 0	2 to D inputs
	$DS_{1,0}$	= 00	
	Immed.	= 00000010	
	I_{8-6}	= 011	Result to PC
	BSEL	= 1001	
	BUS		
	$BS_{1,0}$	= 01	Y output to BUS
	LDMAR	= 1	
3015_8	2910		
	I_{3-0}	= 1110	Continue
	2901		
	I_{5-0}	= 000111	D plus 0
	C_{in}	= 0	"Carry in" = 0
	0/1	= 0	
	$M_{1,0}$	= 00	Normal mode
	BUS	= 1	D inputs from BUS
	I_{8-6}	= 010	Result to TEMP1
	BSEL	= 0100	
	BUS		
	MEMSEL	= 1	Read from memory
	R/$\overline{W}$	= 1	

Assembly (Cont.)

3016_8	2910		
	I_{3-0}	= 0010	JMAP
	$S_{1,0}$	= 10	NA from IR opcode
	2901		
	I_{5-0}	= 000001	A plus B
	C_{in}	= 0	"Carry in" = 0
	0/1	= 0	
	$M_{1,0}$	= 00	Normal mode
	ASEL	= 1001	PC
	BSEL	= 0100	TEMP1
	I_{8-6}	= 010	Store result
	BUS		
	$BS_{1,0}$	= 00	Nothing to BUS

Indexed Addressing

```
   8          11     12          15
 ------------------------------------
|               |                    |
|  0   1   0   0  |   Index reg.     |
|               |                    |
 ------------------------------------
     Indexed    |    ---------
                    |
                    |
                    |
                    |--3:  AC
                    |--A:  IX
                    |--B:  IY
                    |--E:  SP
```

Algorithm

```
   PC <-- PC plus 2
TEMP1 <-- M[PC]
TEMP1 <-- TEMP1 plus AC
                     SP
                     IX
                     IY
```

Assumptions

1. Upon completion of the ea calculation, TEMP1 will contain the ea
2. Opcode = 010/0--/-- CS location 3020_8

Location	ILL Command	Description
3020_8	TFRIA	* Transfer Index to AC * Fetch next sequential microinstruction
3021_8	INCPCMAR	* Increment PC and send new value to MAR * Fetch next sequential microinstruction
3022_8	ADD	* Perform AC plus M[PC] * Fetch next sequential microinstruction
3023_8	TFRIA	* Transfer AC to Index * Fetch next sequential microinstruction
3024_8	TFRIA	* Transfer AC to TEMP1 * Fetch next microinstruction using IR opcode address

Assembly

Location	Microinstruction	Comments
3020_8	2910	
	I_{3-0} = 1110	Continue
	2901	
	I_{5-0} = 000100	A plus 0
	C_{in} = 0	"Carry in" = 0
	0/1 = 0	
	$M_{1,0}$ = 00	Normal mode
	ASEL = 1001	PC
	I_{8-6} = 010	Result to TEMP1
	BSEL = 0100	
	BUS	
	$BS_{1,0}$ = 00	Nothing to BUS
3021_8	2910	
	I_{3-0} = 1110	Continue
	2901	
	I_{5-0} = 000101	D plus A
	C_{in} = 0	"Carry in" = 0
	0/1 = 0	
	$M_{1,0}$ = 00	Normal mode
	ASEL = 1001	PC
	BUS = 0	2 to D inputs
	$DS_{1,0}$ = 00	
	Immed. = 00000010	
	I_{8-6} = 011	Result to PC
	BSEL = 1001	PC
	BUS	
	$BS_{1,0}$ = 01	Y output to BUS
	LDMAR = 1	

Assembly

Location			
3022_8	2910		
	I_{3-0}	= 1110	Continue
	2901		
	I_{5-0}	= 000101	A plus D plus C_{in}
	C_{in}	= 1	C_{in} from CARRY bit
	SS	= 1	Set status bits
	BUS	= 1	D inputs of 2901 come from BUS
	$M_{1,0}$	= 00	Normal mode
	ASEL	= 0011	AC
	I_{8-6}	= 010	Result to AC
	BSEL	= 0011	
	BUS		
	MEMSEL	= 1	Read from memory
	R/$\overline{W}$	= 1	

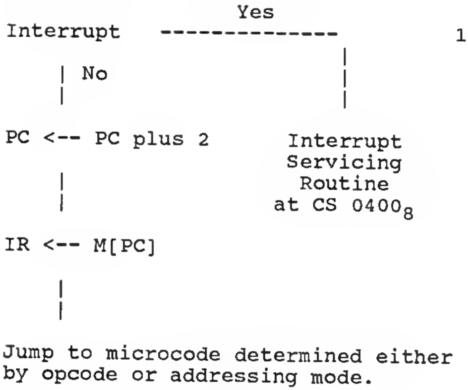
Location			
3023_8	2910		
	I_{3-0}	= 1110	Continue
	2901		
	I_{5-0}	= 000100	A plus 0
	C_{in}	= 0	"Carry in" = 0
	0/1	= 0	
	$M_{1,0}$	= 10	Ind. addr. mode
	ASEL	= 0011	AC
	I_{8-6}	= 010	Result to
	*BSEL	= 0011	AC
		1110	SP
		1010	IX
		1011	IY
	BUS		
	$BS_{1,0}$	= 00	Nothing to BUS

Assembly

Location			
3024_8	2910		
	I_{3-0}	= 0010	JMAP
	$S_{1,0}$	= 10	NA from IR opcode
	2901		
	I_{5-0}	= 000100	A plus 0
	C_{in}	= 0	"Carry in" = 0
	0/1	= 0	
	$M_{1,0}$	= 10	Ind. addr. mode
	ASEL	= 0011	AC
	I_{8-6}	= 010	Result to
	BSEL	= 0100	TEMP1
	BUS		
	$BS_{1,0}$	= 00	Nothing to BUS

Macroinstruction FETCH

Algorithm


Yes
Interrupt
No
PC <-- PC plus 2
Interrupt Servicing Routine at CS 0400$_8$
IR <-- M[PC]
Jump to microcode determined either by opcode or addressing mode.

Assumptions

1. CS location at 1000$_8$

Location	ILL	Command	Description
1000$_8$		TINT	* TEST for INTERRUPT pass - Jump to CS location 0400$_8$ fail - Fetch next sequential microinstruction * Add 2 to PC but don't store result (in case of an interrupt.) * Send result to MAR
1001$_8$		LIRPCINC	* Read memory into IR * Add 2 to PC and store * Fetch next microinstruction via the IR ea/opcode address

Assembly

Location	Microinstruction		Comments
1000_8	2910		
	I_{3-0}	= 0011	CJP
	$TEST_{3-0}$	= 0101	Interrupt status flag
	POL	= 0	Positive polarity
	$S_{1,0}$	= 00	Jump to CS 0400_8
	NA	= 0400_8	
	2901		
	I_{5-0}	= 000101	A plus D
	C_{in}	= 0	"Carry in" = 0
	0/1	= 0	
	$M_{1,0}$	= 00	Normal mode
	ASEL	= 1001	PC
	BUS	= 0	2 to D inputs
	$DS_{1,0}$	= 00	
	Immed.	= 00000010	
	I_{8-6}	= 000	Don't store result
	BUS		
	$BS_{1,0}$	= 01	Y output to BUS
	LDMAR	= 1	

Assembly (Cont.)

1001_8	2910		
	I_{3-0}	= 0010	JMAP
	$S_{1,0}$	= 11	NA from IR (ea/ opcode)
	2901		
	I_{5-0}	= 000101	A plus D
	C_{in}	= 0	"Carry in" = 0
	0/1	= 0	
	$M_{1,0}$	= 00	Normal mode
	ASEL	= 1001	PC
	BUS	= 0	2 to D inputs
	$DS_{1,0}$	= 00	
	Immed.	= 00000010	
	I_{8-6}	= 010	Result to PC
	BSEL	= 1001	
	BUS		
	MEMSEL	= 1	Read value from memory into IR
	$R/\overline{W}$	= 1	
	LDIR	= 1	

Interrupt Servicing Routine

Algorithm

```
Set INTACK
M[SP] <-- PC
Save status register
Set PC to macroinstruction
    servicing routine
```

Assumptions

1. CS location 0400_8
2. Current PC will be saved on user stack via microcode
3. Current status flags will also be saved on the stack.
4. Macrocode servicing routine at location 100_{10}
5. The last macroinstruction in the servicing routine will be RTI

Location	ILL Command	Description
0400_8	INCIASM	* Increment SP and send result to MAR * Acknowledge interrupt * Fetch next sequential microinstruction
0401_8	SPC	* Write PC to memory * Fetch next sequential microinstruction
0402_8	INCSPMAR	* Increment SP and send result to MAR * Fetch next sequential microinstruction
0403_8	FORPC	* PC <-- 100_{10} * Save status register

Assembly

Location	Microinstruction		Comments
0400_8	2910		
	I_{3-0}	= 1110	Continue
	2901		
	I_{5-0}	= 000101	D plus A
	C_{in}	= 0	"Carry in" = 0
	0/1	= 0	
	$M_{1,0}$	= 00	Normal mode
	ASEL	= 1110	SP
	BUS	= 0	2 to D inputs
	$DS_{1,0}$	= 00	
	Immed.	= 00000010	
	I_{8-6}	= 011	Result to SP
	BSEL	= 1110	
	BUS		
	$BS_{1,0}$	= 01	Y output to BUS
	LDMAR	= 1	
	INTACK	= 1	
0401_8	2910		
	I_{3-0}	= 1110	Continue
	2901		
	I_{5-0}	= 000100	A plus 0
	C_{in}	= 0	"Carry in" = 0
	0/1	= 0	
	$M_{1,0}$	= 00	Normal mode
	ASEL	= 1001	PC
	I_{8-6}	= 000	Don't store result
	BUS		
	$BS_{1,0}$	= 01	Write PC to memory
	MEMSEL	= 1	
	R/$\overline{W}$	= 0	

Assembly (Cont.)

0402$_8$	2910		
	I_{3-0}	= 1110	Continue
	2901		
	I_{5-0}	= 000101	D plus A
	C_{in}	= 0	"Carry in" = 0
	0/1	= 0	
	$M_{1,0}$	= 00	Normal mode
	ASEL	= 1110	SP
	BUS	= 0	2 to D inputs
	$DS_{1,0}$	= 00	
	Imméd.	= 00000010	
	I_{8-6}	= 011	Result to SP
	BSEL	= 1110	
	BUS		
	$BS_{1,0}$	= 01	Y output to BUS
	LDMAR	= 1	

Assembly (Cont.)

0403_8	2910		
	I_{3-0}	= 0010	JMAP
	$S_{1,0}$	= 00	FETCH next uinstr.
	NA	= 1000_8	at CS 1000_8
	2901		
	I_{5-0}	= 000111	D plus 0
	C_{in}	= 0	"Carry in" = 0
	0/1	= 0	
	$M_{1,0}$	= 00	Normal mode
	BUS	= 0	100_{10} to D inputs
	$DS_{1,0}$	= 00	
	Immed.	= 01100100	
	I_{8-6}	= 011	Result to PC
	BSEL	= 1001	
	BUS		
	$BS_{1,0}$	= 10	Write CPU status
	MEMSEL	= 1	to memory
	$R/\overline{W}$	= 0	

Initialization

Algorithm

```
PC <-- 0
Reset BUS
FETCH 1st macroinstruction
```

Assumptions

1. The bootstrap program is located in ROM 0_{16}
2. CS location 0000_8

Location	ILL Command	Description
0000_8	INIT	* PC <-- 0 * Jump to FETCH routine at CS 1000_8 via NA field

Assembly

Location	Microinstruction		Comments
0000_8	2910		
	I_{3-0}	= 0010	JMAP
	$S_{1,0}$	= 00	FETCH next uinstr.
	NA	= 1000_8	at CS 1000_8
	2901		
	I_{5-0}	= 000111	D plus 0
	C_{in}	= 0	"Carry in" = 0
	0/1	= 0	
	$M_{1,0}$	= 00	Normal mode
	BUS	= 0	0 to D inputs
	$DS_{1,0}$	= 00	
	Immed.	= 00000000	
	I_{8-6}	= 010	Result to PC
	BSEL	= 1001	
	BUS		
	$BS_{1,0}$	= 00	Nothing to BUS

Appendix B -- Input File for Generation Program

The following pages contain the input file that the control bit generation program read. The format for the various fields is as follows:

ILL command -- j_or_ns -- s -- NA -- flag -- pol -- cs_loc

These parameters were explained in considerable detail in section 12.2.

```
TEMAR      0  0  0000  0  0  2740
LAC        1  0  1000  0  0  2741
TEMAR      0  0  0000  0  0  2742
SAC        1  0  1000  0  0  2743
TEMAR      0  0  0000  0  0  2744
AND        1  0  1000  0  0  2745
TEMAR      0  0  0000  0  0  2746
OR         1  0  1000  0  0  2747
TEMAR      0  0  0000  0  0  2750
ADD        1  0  1000  0  0  2751
TEMAR      0  0  0000  0  0  2752
SUB        1  0  1000  0  0  2753
INCSPMAR   0  0  0000  0  0  2754
SAC        1  0  1000  0  0  2755
SPMARDEC   0  0  0000  0  0  2756
LAC        1  0  1000  0  0  2757
SPMARDEC   0  0  0000  0  0  2760
LPC        1  0  1000  0  0  2761
SPMARDEC   0  0  0000  0  0  2762
LPCRS      1  0  1000  0  0  2763
NOP        1  0  1000  0  0  2764
INC        1  0  1000  0  0  2040
DEC        1  0  1000  0  0  2042
ROR        1  0  1000  0  0  2044
ROL        1  0  1000  0  0  2046
CLR        1  0  1000  0  0  2050
COM        1  0  1000  0  0  2052
TFR        1  0  1000  0  0  2054
HLT        2  0  1000  8  1  2056
NOP        1  0  2056  0  0  2057
TZ1        2  0  1000  4  0  2100
TZ0        2  0  1000  4  1  2102
TNV0NC     2  0  1000  7  1  2104
TZ0        2  0  1000  4  0  2105
TNV1       2  0  1000  7  0  2106
TNV0       2  0  1000  7  1  2110
TNV1NC     2  0  4000  7  0  2112
TZ1        2  0  1000  4  0  2113
PCREL      1  0  1000  0  0  4000
INCSPMAR   0  0  0000  0  0  2114
SPC        1  0  4000  0  0  2115
PCREL      1  0  1000  0  0  2116
DCRTOBUS   0  0  0000  0  0  2000
ACTOBUS    1  0  1000  0  0  2001
DCRTOBUS   0  0  0000  0  0  2002
DATATOAC   1  0  1000  0  0  2003
PCTEMP1    1  2  0000  0  0  3000
INCPCMAR   0  0  0000  0  0  3004
LTEMP      1  2  0000  0  0  3005
INCPCMAR   0  0  0000  0  0  3010
```

```
LTEMP     0  0  0000  0  0  3011
TEMAR     0  0  0000  0  0  3012
LTEMP     1  2  0000  0  0  3013
INCPCMAR  0  0  0000  0  0  3014
LTEMP     0  0  0000  0  0  3015
TEMPC     1  2  0000  0  0  3016
TFRIA     0  0  0000  0  0  3020
INCPCMAR  0  0  0000  0  0  3021
ADD       0  0  0000  0  0  3022
TFRIA     0  0  0000  0  0  3023
TFRIA     1  2  1000  0  0  3024
TINT      2  0  0400  5  0  1000
LIRPCINC  1  3  0000  0  0  1001
INCIASM   0  0  0000  0  0  0400
SPC       0  0  0000  0  0  0401
INCSPMAR  0  0  0000  0  0  0402
FORPC     1  0  1000  0  0  0403
INIT      1  0  1000  0  0  0000
```

Appendix C -- Control Store Memory Dump

A memory dump of the CS memory, the output of the computer program written to generate the control bits, is seen on the following pages. Please note that only the locations with relevant data are shown.

CS	s	NA	i3_0	pol	t3_0	immed	ds	BUS	se	te
0000	00	1000	0010	0	0000	00000000	00	0	0	0
0001	00	0000	0000	0	0000	00000000	00	0	0	0
0400	00	0000	1110	0	0000	00000010	00	0	0	0
0401	00	0000	1110	0	0000	00000000	00	0	0	0
0402	00	0000	1110	0	0000	00000010	00	0	0	0
0403	00	1000	0010	0	0000	01100100	00	0	0	0
0404	00	0000	0000	0	0000	00000000	00	0	0	0
1000	00	0400	0011	0	0101	00000010	00	0	0	0
1001	11	0000	0010	0	0000	00000010	00	0	0	0
1002	00	0000	0000	0	0000	00000000	00	0	0	0
2000	00	0000	1110	0	0000	00000000	00	0	0	0
2001	00	1000	0010	0	0000	00000000	00	0	0	0
2002	00	0000	1110	0	0000	00000000	00	0	0	0
2003	00	1000	0010	0	0000	00000000	00	1	0	0
2004	00	0000	0000	0	0000	00000000	00	0	0	0
2040	00	1000	0010	0	0000	00000001	00	0	0	0
2041	00	0000	0000	0	0000	00000000	00	0	0	0
2042	00	1000	0010	0	0000	11111111	00	0	0	0
2043	00	0000	0000	0	0000	00000000	00	0	0	0
2044	00	1000	0010	0	0000	00000000	00	0	0	0
2045	00	0000	0000	0	0000	00000000	00	0	0	0
2046	00	1000	0010	0	0000	00000000	00	0	0	0
2047	00	0000	0000	0	0000	00000000	00	0	0	0
2050	00	1000	0010	0	0000	00000000	00	0	0	0
2051	00	0000	0000	0	0000	00000000	00	0	0	0
2052	00	1000	0010	0	0000	11111111	00	0	0	0
2053	00	0000	0000	0	0000	00000000	00	0	0	0
2054	00	1000	0010	0	0000	00000000	00	0	0	0
2055	00	0000	0000	0	0000	00000000	00	0	0	0
2056	00	1000	0011	1	0000	00000000	00	0	0	0
2057	00	2056	0010	0	0000	00000000	00	0	0	0
2100	00	1000	0011	0	0100	00000000	10	0	1	1
2101	00	0000	0000	0	0000	00000000	00	0	0	0
2102	00	1000	0011	1	0100	00000000	10	0	1	1
2103	00	0000	0000	0	0000	00000000	00	0	0	0
2104	00	1000	0011	1	0111	00000000	00	0	0	0
2105	00	1000	0011	0	0100	00000000	10	0	1	1
2106	00	1000	0011	0	0111	00000000	10	0	1	1
2107	00	0000	0000	0	0000	00000000	00	0	0	0
2110	00	1000	0011	1	0111	00000000	10	0	1	1
2111	00	0000	0000	0	0000	00000000	00	0	0	0
2112	00	4000	0011	0	0111	00000000	00	0	0	0
2113	00	1000	0011	0	0100	00000000	10	0	1	1
2114	00	0000	1110	0	0000	00000010	00	0	0	0

```
2115   00   4000   0010   0   0000   00000000   00   0    0    0
2116   00   1000   0010   0   0000   00000000   10   0    1    0
2117   00   0000   0000   0   0000   00000000   00   0    0    0

2740   00   0000   1110   0   0000   00000000   00   0    0    0
2741   00   1000   0010   0   0000   00000000   00   1    0    0
2742   00   0000   1110   0   0000   00000000   00   0    0    0
2743   00   1000   0010   0   0000   00000000   00   0    0    0
2744   00   0000   1110   0   0000   00000000   00   0    0    0
2745   00   1000   0010   0   0000   00000000   00   1    0    0
2746   00   0000   1110   0   0000   00000000   00   0    0    0
2747   00   1000   0010   0   0000   00000000   00   1    0    0
2750   00   0000   1110   0   0000   00000000   00   0    0    0
2751   00   1000   0010   0   0000   00000000   00   1    0    0
2752   00   0000   1110   0   0000   00000000   00   0    0    0
2753   00   1000   0010   0   0000   00000000   00   1    0    0
2754   00   0000   1110   0   0000   00000010   00   0    0    0
2755   00   1000   0010   0   0000   00000000   00   0    0    0
2756   00   0000   1110   0   0000   11111110   00   0    0    0
2757   00   1000   0010   0   0000   00000000   00   1    0    0
2760   00   0000   1110   0   0000   11111110   00   0    0    0
2761   00   1000   0010   0   0000   00000000   00   1    0    0
2762   00   0000   1110   0   0000   11111110   00   0    0    0
2763   00   1000   0010   0   0000   00000000   00   1    0    0
2764   00   1000   0010   0   0000   00000000   00   0    0    0

3000   10   0000   0010   0   0000   00000000   00   0    0    0
3001   00   0000   0000   0   0000   00000000   00   0    0    0
3002   00   0000   0000   0   0000   00000000   00   0    0    0
3003   00   0000   0000   0   0000   00000000   00   0    0    0
3004   00   0000   1110   0   0000   00000010   00   0    0    0
3005   10   0000   0010   0   0000   00000000   00   1    0    0
3006   00   0000   0000   0   0000   00000000   00   0    0    0
3007   00   0000   0000   0   0000   00000000   00   0    0    0
3010   00   0000   1110   0   0000   00000010   00   0    0    0
3011   00   0000   1110   0   0000   00000000   00   1    0    0
3012   00   0000   1110   0   0000   00000000   00   0    0    0
3013   10   0000   0010   0   0000   00000000   00   1    0    0
3014   00   0000   1110   0   0000   00000010   00   0    0    0
3015   00   0000   1110   0   0000   00000000   00   1    0    0
3016   10   0000   0010   0   0000   00000000   00   0    0    0
3017   00   0000   0000   0   0000   00000000   00   0    0    0
3020   00   0000   1110   0   0000   00000000   00   0    0    0
3021   00   0000   1110   0   0000   00000010   00   0    0    0
3022   00   0000   1110   0   0000   00000000   00   1    0    0
3023   00   0000   1110   0   0000   00000000   00   0    0    0
3024   10   0000   0010   0   0000   00000000   00   0    0    0

4000   00   1000   0010   0   0000   00000000   10   0    1    0
```

CS	ASEL	BSEL	mode	i8_6	i5_0	z_or_o	cin	ss	bs	ldir
0000	0000	1001	00	010	000111	0	0	0	00	0
0001	0000	0000	00	000	000000	0	0	0	00	0
0400	1110	1110	00	011	000101	0	0	0	01	0
0401	1001	0000	00	000	000100	0	0	0	01	0
0402	1110	1110	00	011	000101	0	0	0	01	0
0403	0000	1001	00	011	000111	0	0	0	10	0
0404	0000	0000	00	000	000000	0	0	0	00	0
1000	1001	0000	00	000	000101	0	0	0	01	0
1001	1001	1001	00	010	000101	0	0	0	00	1
1002	0000	0000	00	000	000000	0	0	0	00	0
2000	0000	0000	00	000	000100	0	0	0	01	0
2001	0011	0000	00	000	000100	0	0	0	01	0
2002	0000	0000	00	000	000100	0	0	0	01	0
2003	0000	0011	00	010	000111	0	0	0	00	0
2004	0000	0000	00	000	000000	0	0	0	00	0
2040	9999	9999	11	010	000101	0	0	1	00	0
2041	0000	0000	00	000	000000	0	0	0	00	0
2042	9999	9999	11	010	001101	1	0	1	00	0
2043	0000	0000	00	000	000000	0	0	0	00	0
2044	9999	9999	11	100	000100	0	1	1	00	0
2045	0000	0000	00	000	000000	0	0	0	00	0
2046	9999	9999	11	110	000100	0	1	1	00	0
2047	0000	0000	00	000	000000	0	0	0	00	0
2050	9999	9999	11	010	000111	0	0	1	00	0
2051	0000	0000	00	000	000000	0	0	0	00	0
2052	9999	9999	11	010	110101	0	0	1	00	0
2053	0000	0000	00	000	000000	0	0	0	00	0
2054	9999	9999	11	010	000100	0	0	1	00	0
2055	0000	0000	00	000	000000	0	0	0	00	0
2056	0000	0000	00	000	000000	0	0	0	00	0
2057	0000	0000	00	000	000000	0	0	0	00	0
2100	1001	1001	00	010	000101	0	0	0	00	0
2101	0000	0000	00	000	000000	0	0	0	00	0
2102	1001	1001	00	010	000101	0	0	0	00	0
2103	0000	0000	00	000	000000	0	0	0	00	0
2104	0000	0000	00	000	000000	0	0	0	00	0
2105	1001	1001	00	010	000101	0	0	0	00	0
2106	1001	1001	00	010	000101	0	0	0	00	0
2107	0000	0000	00	000	000000	0	0	0	00	0
2110	1001	1001	00	010	000101	0	0	0	00	0
2111	0000	0000	00	000	000000	0	0	0	00	0
2112	0000	0000	00	000	000000	0	0	0	00	0
2113	1001	1001	00	010	000101	0	0	0	00	0
2114	1110	1110	00	011	000101	0	0	0	01	0

```
2115   1001   0000    00    000   000100     0      0     0   01      0
2116   1001   1001    00    010   000101     0      0     0   00      0
2117   0000   0000    00    000   000000     0      0     0   00      0

2740   0100   0000    00    000   000100     0      0     0   01      0
2741   0000   0011    00    010   000111     0      0     1   00      0
2742   0100   0000    00    000   000100     0      0     0   01      0
2743   0011   0000    00    000   000100     0      0     1   00      0
2744   0100   0000    00    000   000100     0      0     0   01      0
2745   0011   0011    00    010   100101     0      0     1   00      0
2746   0100   0000    00    000   000100     0      0     0   01      0
2747   0011   0011    00    010   011101     0      0     1   00      0
2750   0100   0000    00    000   000100     0      0     0   01      0
2751   0011   0011    00    010   000101     0      1     1   00      0
2752   0100   0000    00    000   000100     0      0     0   01      0
2753   0011   0011    00    010   001101     0      1     1   00      0
2754   1110   1110    00    011   000101     0      0     0   01      0
2755   0011   0000    00    000   000100     0      0     1   00      0
2756   1110   1110    00    010   001100     1      0     0   01      0
2757   0000   0011    00    010   000111     0      0     1   00      0
2760   1110   1110    00    010   001100     1      0     0   01      0
2761   0000   1001    00    010   000111     0      0     0   00      0
2762   1110   1110    00    010   001100     1      0     0   01      0
2763   0000   1001    00    010   000111     0      0     0   00      0
2764   0000   0000    00    000   000000     0      0     0   00      0

3000   1001   0100    00    010   000100     0      0     0   00      0
3001   0000   0000    00    000   000000     0      0     0   00      0
3002   0000   0000    00    000   000000     0      0     0   00      0
3003   0000   0000    00    000   000000     0      0     0   00      0
3004   1001   1001    00    011   000101     0      0     0   01      0
3005   0000   0100    00    010   000111     0      0     0   00      0
3006   0000   0000    00    000   000000     0      0     0   00      0
3007   0000   0000    00    000   000000     0      0     0   00      0
3010   1001   1001    00    011   000101     0      0     0   01      0
3011   0000   0100    00    010   000111     0      0     0   00      0
3012   0100   0000    00    000   000100     0      0     0   01      0
3013   0000   0100    00    010   000111     0      0     0   00      0
3014   1001   1001    00    011   000101     0      0     0   01      0
3015   0000   0100    00    010   000111     0      0     0   00      0
3016   1001   0100    00    010   000001     0      0     0   00      0
3017   0000   0000    00    000   000000     0      0     0   00      0
3020   1001   0100    00    010   000100     0      0     0   00      0
3021   1001   1001    00    011   000101     0      0     0   01      0
3022   0011   0011    00    010   000101     0      1     1   00      0
3023   0011   9999    10    010   000100     0      0     0   00      0
3024   0011   0100    10    010   000100     0      0     1   00      0

4000   1001   1001    00    010   000101     0      0     0   00      0
```

CS	memsel	r_w	ldmar	io_sel	io_s_d	intack	rs
0000	0	0	0	0	0	0	0
0001	0	0	0	0	0	0	0
0400	0	0	1	0	0	1	0
0401	1	0	0	0	0	0	0
0402	0	0	1	0	0	0	0
0403	1	0	0	0	0	0	0
0404	0	0	0	0	0	0	0
1000	0	0	1	0	0	0	0
1001	1	1	0	0	0	0	0
1002	0	0	0	0	0	0	0
2000	0	0	0	1	0	0	0
2001	0	0	0	1	0	0	0
2002	0	0	0	1	0	0	0
2003	0	0	0	1	0	0	0
2004	0	0	0	0	0	0	0
2040	0	0	0	0	0	0	0
2041	0	0	0	0	0	0	0
2042	0	0	0	0	0	0	0
2043	0	0	0	0	0	0	0
2044	0	0	0	0	0	0	0
2045	0	0	0	0	0	0	0
2046	0	0	0	0	0	0	0
2047	0	0	0	0	0	0	0
2050	0	0	0	0	0	0	0
2051	0	0	0	0	0	0	0
2052	0	0	0	0	0	0	0
2053	0	0	0	0	0	0	0
2054	0	0	0	0	0	0	0
2055	0	0	0	0	0	0	0
2056	0	0	0	0	0	0	0
2057	0	0	0	0	0	0	0
2100	0	0	0	0	0	0	0
2101	0	0	0	0	0	0	0
2102	0	0	0	0	0	0	0
2103	0	0	0	0	0	0	0
2104	0	0	0	0	0	0	0
2105	0	0	0	0	0	0	0
2106	0	0	0	0	0	0	0
2107	0	0	0	0	0	0	0
2110	0	0	0	0	0	0	0
2111	0	0	0	0	0	0	0
2112	0	0	0	0	0	0	0
2113	0	0	0	0	0	0	0
2114	0	0	1	0	0	0	0

2115	1	0	0	0	0	0	0
2116	0	0	0	0	0	0	0
2117	0	0	0	0	0	0	0
2740	0	0	1	0	0	0	0
2741	1	1	0	0	0	0	0
2742	0	0	1	0	0	0	0
2743	1	0	0	0	0	0	0
2744	0	0	1	0	0	0	0
2745	1	1	0	0	0	0	0
2746	0	0	1	0	0	0	0
2747	1	1	0	0	0	0	0
2750	0	0	1	0	0	0	0
2751	1	1	0	0	0	0	0
2752	0	0	1	0	0	0	0
2753	1	1	0	0	0	0	0
2754	0	0	1	0	0	0	0
2755	1	0	0	0	0	0	0
2756	0	0	1	0	0	0	0
2757	1	1	0	0	0	0	0
2760	0	0	1	0	0	0	0
2761	1	1	0	0	0	0	0
2762	0	0	1	0	0	0	0
2763	1	1	0	0	0	0	1
2764	0	0	0	0	0	0	0
3000	0	0	0	0	0	0	0
3001	0	0	0	0	0	0	0
3002	0	0	0	0	0	0	0
3003	0	0	0	0	0	0	0
3004	0	0	1	0	0	0	0
3005	1	1	0	0	0	0	0
3006	0	0	0	0	0	0	0
3007	0	0	0	0	0	0	0
3010	0	0	1	0	0	0	0
3011	1	1	0	0	0	0	0
3012	0	0	1	0	0	0	0
3013	1	1	0	0	0	0	0
3014	0	0	1	0	0	0	0
3015	1	1	0	0	0	0	0
3016	0	0	0	0	0	0	0
3017	0	0	0	0	0	0	0
3020	0	0	0	0	0	0	0
3021	0	0	1	0	0	0	0
3022	1	1	0	0	0	0	0
3023	0	0	0	0	0	0	0
3024	0	0	0	0	0	0	0
4000	0	0	0	0	0	0	0

Appendix D -- Control Bit Generation Program

This appendix contains the functions written by the author to generate the control bits. The four functions were written in the C programming language and include a main function which calls the clear, generate and store functions. These functions were introduced in section 12.3 and are fully documented to promote the ease in which they may be read.

```
/************************************************************
 *
 *   SOURCE FILE:  [culp]main.c
 *
 *
 *   FUNCTION:     main.c
 *
 *
 *   DESCRIPTION:  This program will read data from an input
 *                     file, generate the corresponding
 *                     control bits, store these control bits
 *                     in the specified Control Store memory
 *                     location and print to an output
 *                     file a dump of the Control Store memory.
 *
 *                 Note -- The input file should use the
 *                         following form for listing the data,
 *                         with at least one space between
 *                         each field.
 *
 *
 *
 * ILL command -- j_or_ns -- s -- NA -- flag -- pol -- cs_loc
 *
 *
 *
 *
 *   DOCUMENTATION
 *   FILES:        None.
 *
 *
 *   ARGUMENTS:    None.
 *
 *
 *   RETURN:       int
 *                 NORMAL :  Normal execution
 *                 ERROR  :  An error has occurred
 *
 *
 *
 *   FUNCTIONS
 *   CALLED:       clear()      --   clears all the parameters
 *                                   passed to it.  In this
 *                                   case, all the control bits
 *                                   received by clear() are
 *                                   set to 0.
 *
 *                 generate()   --   produces the control bits
 *                                   given the ILL command and
 *                                   other needed information.
 *
 *                store()     --  stores the generated control
 *                                bits into the Control Store.
 *
 *
 *
 *  AUTHOR:       Steven H. Culp
 *
 *
 *  DATE
 *  CREATED:      22Mar88                         Version  1.00
 *
 *
 *  REVISIONS:    None.
 *
 *
***************************************************************/
#include   <stdio.h>


/* ------------------------------------------------------------ */
/* The following defs. are for the types of termination.       */
/* ------------------------------------------------------------ */
#define     NORMAL       0
#define     ERROR       10

#define     MAX_LEN      8    /*  max. length of ILL command */
#define     BUFFER      88    /*  no. of buffer elements     */


/* ------------------------------------------------------------ */
/*    Main program begins here.                                 */
/* ------------------------------------------------------------ */
main()
{

char command[MAX_LEN],        /*  holds the ILL command of   */
                              /*    which the control bits   */
                              /*    are to be generated      */
     i_f_name[15],            /*  input file name            */
     o_f_name[15],            /*  output  "     "            */
     s_buf[BUFFER];           /*  string buffer              */

int  clear(),      /* a fcn. which clears the control bits  */
     cs_loc,       /* the Control Store location, i.e. the  */
                   /*   addr. of the microinstr.            */
     cs[4001][88], /* Control Store is 4k by 88 bits wide */
     flag,         /*  which status flag is being tested    */
     generate(), /*  a fcn. which generates the control    */
```

```
                 /*    bits                                */
     i,          /*  i is an index                         */
     j,          /*  j   "  "   "                          */
     j_or_ns,    /*  jump vs. next sequential microinstr.  */
     store(),    /*  a fcn. which stores the generated     */
                 /*    microinstr. in the Control Store    */

/* ------------------------------------------------------- */
/* These are the control bits. They are used as specified */
/*     in the thesis.                                      */
/* ------------------------------------------------------- */
     ASEL,          /*  2901 reg. A select                 */
     BSEL,          /*    "   "   B    "                   */
     bs,            /*  bus select control bits            */
     BUS,           /*  determines if BUS is selected      */
     cin,           /*  "carry in" source determination    */
     ds,            /*  2901 D input source                */
     io_sel,        /*  I/O device selected                */
     io_s_d,        /*  I/O status or data                 */
     i3_0,          /*  2910 instruction bits              */
     i5_0,          /*  2901 source and function bits      */
     i8_6,          /*  2901 destination bits              */
     immed,         /*  immed. field in the microinstr.    */
     intack,        /*  acknowledge an interrupt           */
     ldir,          /*  load instr. reg., a.k.a. IR        */
     ldmar,         /*   "  memory addr. reg., a.k.a. MAR */
     memsel,        /*  memory selected                    */
     mode,          /*  mode of 2901 ALU operation         */
     NA,            /*  next address field in microinstr. */
     pol,           /*  polarity determination bit         */
     rs,            /*  restore status bits                */
     r_w,           /*  read or write operation            */
     s,             /*  2910 D input source                */
     se,            /*  sign extend                        */
     ss,            /*  save status                        */
     t3_0,          /*  2910 test determination bits       */
     te,            /*  test enable                        */
     z_or_one;      /*  0 or 1                             */

FILE *input_file,   /*  a pointer to the input file        */
     *output_file;  /*  "    "     "  "  output  "         */


/* ------------------------------------------------------- */
/* Prompt for the input and output file names, then open  */
/*   each file for reading and writing, respectively.      */
/* ------------------------------------------------------- */
printf("\n\nInput file name:   ");
scanf ("%s", i_f_name);
```

```
printf("\nOutput file name:  ");
scanf ("%s", o_f_name);

input_file  =  fopen(i_f_name, "r");
if (input_file == NULL)
     {
     printf("\nFile can not be opened for input.");
     return(ERROR);
     }

output_file =  fopen(o_f_name, "w");
if (output_file == NULL)
     {
     printf("\nFile can not be opened for output.");
     return(ERROR);
     }


/* ------------------------------------------------------------ */
/*    Initialize the entire Control Store memory to 0.          */
/* ------------------------------------------------------------ */
for (i = 0; i < 4001; i++)
     for (j = 0; j < 88; j++)
          cs[i][j]  = 0;


/* ------------------------------------------------------------ */
/*    The main loop begins here.                                */
/* ------------------------------------------------------------ */
while ((fgets(s_buf, BUFFER, input_file)) != NULL)
     {

/* ------------------------------------------------------------ */
/*    Reset/initialize all control bits to 0.                   */
/* ------------------------------------------------------------ */
     if (clear(&ASEL, &BSEL, &bs, &BUS, &cin, &ds, &io_sel,
                 &io_s_d, &i3_0, &i5_0, &i8_6, &immed, &intack,
                 &ldir, &ldmar, &memsel, &mode, &NA, &pol, &rs,
                 &r_w, &s, &se, &ss, &t3_0, &te, &z_or_one))
          (
          printf("\nAn error has occurred in clear().");
          return(ERROR);
          }


/* ------------------------------------------------------------ */
/*    Read the data.                                            */
/* ------------------------------------------------------------ */
fscanf(input_file, "%s", command);
fscanf(input_file, "%d %d %d %d %d %d",
             &j_or_ns, &s, &NA, &flag, &pol, &cs_loc);


      /* ------------------------------------------------------ */
      /* The generate() is now called to generate the            */
      /*   appropriate control bits that correspond to the       */
      /*    given ILL command.                                   */
      /* ------------------------------------------------------ */
      if (generate(command, flag, j_or_ns, &ASEL, &BSEL, &bs,
                   &BUS, &cin, &ds, &io_sel, &io_s_d, &i3_0, &i5_0,
                   &i8_6, &immed, &intack, &ldir, &ldmar, &memsel,
                   &mode, NA, pol, &rs, &r_w, &s, &se, &ss, &t3_0,
                   &te, &z_or_one))
              {
              printf("\nAn error has occurred in generate().");
              return(ERROR);
              }


     /* ------------------------------------------------------ */
     /*    Print generated control bits to verify that all is   */
     /*      correct.                                           */
     /* ------------------------------------------------------ */
         printf("\n\n\n\n\nThe generated control bits for:      ");
         for (i = 0; i < MAX_LEN; i++)
             printf("%c", command[i]);

     printf("\n\nFor the 2910          For the 2901           ");
     printf(" For the BUS");
     printf("\ni3_0 = %04d    i5_0     =    %06d    bs     =    %02d",
          i3_0, i5_0, bs);
  printf("\nt3_0 = %04d    cin      =     %4d    ldir   =  %4d",
          t3_0, cin, ldir);
  printf("\npol   = %4d    z_or_one =     %4d    ldmar  =  %4d",
          pol, z_or_one, ldmar);
  printf("\ns     =   %02d    ss       =     %4d    io_sel =  %4d",
          s,    ss,        io_sel);
  printf("\nNA    = %04d    mode     =      %02d    io_s_d =  %04d",
        NA,    mode,      io_s_d);
printf("\n                          ASEL     =     %04d    memsel =  %4d",
       ASEL,      memsel);
printf("\n                          BUS      =     %4d    r_w    =  %4d",
       BUS,       r_w);
printf("\n                          ds       =     %4d    intack =  %4d",
       ds,        intack);
printf("\n                          immed    =  %08d    rs      %1d",
       immed,     rs);
printf("\n                          se       =     %4d            ",se);
printf("\n                          te       =     %4d              ",
        te);
    printf("\n                    i8_6     =      %03d           ", i8_6);
    printf("\n                    BSEL     =     %04d            ", BSEL);


       if (store(cs, cs_loc, ASEL, BSEL, bs, BUS, cin, ds,
               io_sel, io_s_d, i3_0, i5_0, i8_6, immed,
               intack, ldir, ldmar, memsel, mode, NA, pol
               rs, r_w, s, se, ss, t3_0, te, z_or_one))
          {
          printf("\nAn error has occurred in store().");
          return(ERROR);
          }

       }


    fprintf(output_file, " CS     s    NA   i3_0  pol  t3_0");
    fprintf(output_file, "    immed   ds BUS  se   te");
    for (i = 0; i < 4001; i++)
       {
       fprintf(output_file, "\n%04d   %02d   %04d  %04d",
             i, cs[i][0], cs[i][1], cs[i][2]);
       fprintf(output_file, "   %1d   %04d",
             cs[i][3], cs[i][4]);
       fprintf(output_file, "  %08d  %02d  %1d    %1d",
             cs[i][5], cs[i][6], cs[i][7], cs[i][8]);
       fprintf(output_file, "   %1d",
             cs[i][9]);

       }
    fprintf(output_file,"\n\n\n CS   ASEL  BSEL  mode i8_6");
    fprintf(output_file,"   i5_0   z_or_o cin ss  bs  ldir");
    for (i = 0; i < 4001; i++)
       {
       fprintf(output_file, "\n%04d  %04d  %04d   %02d   %03d",
             i, cs[i][10], cs[i][11], cs[i][12], cs[i][13]);
       fprintf(output_file, "  %06d     %1d    %1d    %1d",
             cs[i][14], cs[i][15], cs[i][16], cs[i][17]);
       fprintf(output_file, "  %02d     %1d",
             cs[i][18], cs[i][19]);
       }
    fprintf(output_file,"\n\n\n CS    memsel r_w ldmar io_sel");
    fprintf(output_file," io_s_d  intack  rs");
    for (i = 0; i < 4001; i++)
       {
       fprintf(output_file, "\n%04d      %1d    %1d    %1d",
             i, cs[i][20], cs[i][21], cs[i][22]);
       fprintf(output_file, "   %1d      %1d       %1d      %1d",
             cs[i][23], cs[i][24], cs[i][25], cs[i][26]);
```

```
   }


return(NORMAL);
}


/*   EOJ  --  Steven H. Culp  --  main()                     */
```

```
/*************************************************************
 *
 *  SOURCE FILE:  [culp]clear.c
 *
 *
 *  FUNCTION:     clear.c
 *
 *
 *  DESCRIPTION:  This function clears, i.e. sets to 0 all
 *                    the parameters it receives.
 *
 *
 *  DOCUMENTATION
 *  FILES:        None.
 *
 *
 *  ARGUMENTS:
 *
 *    ASEL        (input)   int  *
 *                a four-bit field which selects one of the
 *                16 RAM locations to be fed into the A inputs
 *                of the 2901 ALU.
 *
 *    BSEL        (input)   int  *
 *                a four-bit field which selects one of the
 *                16 RAM locations to be fed into the B inputs
 *                of the 2901 ALU.  BSEL also functions as the
 *                address of the destination register.
 *
 *    bs          (input)   int  *
 *                determine what data is transferred to the BUS
 *
 *    BUS         (input)   int  *
 *                indicates whether D inputs to 2901 come from
 *                the BUS or from ds
 *
 *    cin         (input)   int  *
 *                indicates whether the carry in value comes
 *                from the CARRY bit or from 0/1
 *
 *    ds          (input)   int  *
 *                determines the source of the D inputs to
 *                the 2901
 *
 *    io_sel      (input)   int  *
 *                determines if a transfer to an I/O device is
 *                to take place.
 *
 *    io_s_d      (input)   int  *
 *                determines whether the status of a device
*                   is requested or if data is being sent
*
*    i3_0           (input)   int  *
*                   four bits which determine the instruction
*                   the 2910 will execute
*
*    i5_0           (input)   int  *
*                   six bits which determine the source for the
*                   ALU operands and the ALU function
*
*    i8_6           (input)   int  *
*                   three bits which determine the destination
*                   of the ALU result
*
*    immed          (input)   int  *
*                   supplies an immediate value for the D inputs
*                   to the 2901
*
*    intack         (input)   int  *
*                   acknowledges that an I/O device is prompting
*                   to be serviced.
*
*    ldir           (input)   int  *
*                   indicates that the IR is to be loaded with
*                   the value currently on the BUS
*
*    ldmar          (input)   int  *
*                   indicates that the MAR is to be loaded with
*                   the value currently on the BUS
*
*    memsel         (input)   int  *
*                   indicates that a transfer to or from memory
*                   is to take place
*
*    mode           (input)   int  *
*                   two bits which define the four modes in
*                   which the 2901 ALU is capable of operating
*
*    NA             (input)   int  *
*                   12-bit field which supplies a potential
*                   next address for the CS
*
*    pol            (input)   int  *
*                   determines whether positive or negative
*                   polarity is being used
*
*    rs             (input)   int  *
*                   determines whether to restore the status
*                   flags
*
```

```
*    r_w            (input)   int  *
*                   indicates whether a read or a write
*                   operation is to take place
*
*    s              (input)   int  *
*                   determines where the D inputs to the 2910
*                   originate
*
*    se             (input)   int  *
*                   indicates whether to sign extend the D
*                   inputs to the 2901
*
*    ss             (input)   int  *
*                   indicates whether current operation should
*                   set the status flags
*
*    t3_0           (input)   int  *
*                   four bits which indicate the status flag to
*                   be tested
*
*    te             (input)   int  *
*                   allows the test result to force the D inputs
*                   to zero
*
*    z_or_one       (input)   int  *
*                   functions as a forced set or clear for cin
*
*
*
*  RETURN:          int
*                   NORMAL    :  normal return
*                   ERR_CLEAR :  an error has occurred
*
*
*
*  FUNCTIONS
*  CALLED:          None.
*
*
*  AUTHOR:          Steven H. Culp
*
*
*  DATE
*  CREATED:         25Mar88                         Version  1.00
*
*
*  REVISIONS:       None.
*
*
*************************************************************/
```

```
  /* ------------------------------------------------------- */
  /* The following defs. are for the types of termination.  */
  /* ------------------------------------------------------- */
  #define   NORMAL      0
  #define   ERR_CLEAR   20


  int  clear(ASEL, BSEL, bs, BUS, cin, ds, io_sel, io_s_d, i3_0,
             i5_0, i8_6, immed, intack, ldir, ldmar, memsel, mode,
            NA, pol, rs, r_w, s, se, ss, t3_0, te, z_or_one)

/* ------------------------------------------------------- */
/* The following parameters are defined exactly as in the */
/*   main() and in the thesis.                             */
/* ------------------------------------------------------- */
int    *ASEL, *BSEL, *bs, *BUS, *cin, *ds, *io_sel, *io_s_d,
       *i3_0, *i5_0, *i8_6, *immed, *intack, *ldir, *ldmar,
       *memsel, *mode, *NA, *pol, *rs, *r_w, *s, *se, *ss,
       *t3_0, *te, *z_or_one;


/* ------------------------------------------------------- */
/* Clear function begins here.  All the parameters are     */
/*  being set to 0.  Unless otherwise stated, all control */
/*  bits are assumed to be 0.                              */
/* ------------------------------------------------------- */
{
*ASEL       =  0;
*BSEL       =  0;
*bs         =  0;
*BUS        =  0;
*cin        =  0;
*ds         =  0;
*io_sel     =  0;
*io_s_d     =  0;
*i3_0       =  0;
*i5_0       =  0;
*i8_6       =  0;
*immed      =  0;
*intack     =  0;
*ldir       =  0;
*ldmar      =  0;
*memsel     =  0;
*mode       =  0;
*NA         =  0;
*pol        =  0;
*rs         =  0;
*r_w        =  0;
*s          =  0;
*se         =  0;
*ss         =  0;
*t3_0       =  0;
*te         =  0;
*z_or_one   =  0;

return(NORMAL);
}


/*   EOJ  --  Steven H. Culp  --  clear()                    */
```

```
/*************************************************************
 *
 *  SOURCE FILE:  [culp]generate.c
 *
 *
 *  FUNCTION:     generate.c
 *
 *
 *  DESCRIPTION:  This function generates the control bits
 *                     for the given ILL command.
 *
 *                Note -- for some instructions, the ASEL and
 *                        BSEL fields are taken directly from
 *                        the IR.  When this occurs, a value
 *                        of 9999 for ASEL and/or BSEL is
 *                        generated.
 *
 *
 *  DOCUMENTATION
 *  FILES:        None.
 *
 *
 *  ARGUMENTS:
 *
 *    ASEL        (input)   int  *
 *                a four-bit field which selects one of the
 *                16 RAM locations to be fed into the A inputs
 *                of the 2901 ALU.
 *
 *    BSEL        (input)   int  *
 *                a four-bit field which selects one of the
 *                16 RAM locations to be fed into the B inputs
 *                of the 2901 ALU.  BSEL also functions as the
 *                address of the destination register.
 *
 *    bs          (input)   int  *
 *                determine what data is transferred to the BUS
 *
 *    BUS         (input)   int  *
 *                indicates whether D inputs to 2901 come from
 *                the BUS or from ds
 *
 *    cin         (input)   int  *
 *                indicates whether the carry in value comes
 *                from the CARRY bit or from 0/1
 *
 *    command[]   (input)   char *
 *                contains the ILL command for which the control
 *                bits are to be generated
 *
```

```
*     ds            (input)   int  *
*                   determines the source of the D inputs to
*                   the 2901
*
*     flag          (input)   int
*                   indicates which status flag is to be tested
*
*     io_sel        (input)   int  *
*                   determines if a transfer to an I/O device is
*                   to take place.
*
*     io_s_d        (input)   int  *
*                   determines whether the status of a device
*                   is requested or if data is being sent
*
*     i3_0          (input)   int  *
*                   four bits which determine the instruction
*                   the 2910 will execute
*
*     i5_0          (input)   int  *
*                   six bits which determine the source for the
*                   ALU operands and the ALU function
*
*     i8_6          (input)   int  *
*                   three bits which determine the destination
*                   of the ALU result
*
*     immed         (input)   int  *
*                   supplies an immediate value for the D inputs
*                   to the 2901
*
*     intack        (input)   int  *
*                   acknowledges that an I/O device is prompting
*                   to be serviced.
*
*     j_or_ns       (input)   int
*                   indicates whether the next microinstruction
*                   is jumped to or if it is the next sequential
*                   microinstruction
*
*     ldir          (input)   int  *
*                   indicates that the IR is to be loaded with
*                   the value currently on the BUS
*
*     ldmar         (input)   int  *
*                   indicates that the MAR is to be loaded with
*                   the value currently on the BUS
*
*     memsel        (input)   int  *
*                   indicates that a transfer to or from memory
*                     is to take place
*
*   mode              (input)   int  *
*                     two bits which define the four modes in
*                     which the 2901 ALU is capable of operating
*
*   NA                (input)   int
*                     12-bit field which supplies a potential
*                     address for the CS
*
*   pol               (input)   int
*                     determines whether positive or negative
*                     polarity is being used
*
*   rs                (input)   int  *
*                     indicates if the status bits should be
*                     restored
*
*   r_w               (input)   int  *
*                     indicates whether a read or a write
*                     operation is to take place
*
*   s                 (input)   int  *
*                     determines where the D inputs to the 2910
*                     originate
*
*   se                (input)   int  *
*                     indicates whether to sign extend the D
*                     inputs to the 2901
*
*   ss                (input)   int  *
*                     indicates whether current operation should
*                     set the status flags
*
*   t3_0              (input)   int  *
*                     four bits which indicate the status flag to
*                     be tested
*
*   te                (input)   int  *
*                     allows the test result to force the D inputs
*                     to zero
*
*   z_or_one          (input)   int  *
*                     functions as a forced set or clear for cin
*
*
*
* RETURN:             int
*                     NORMAL        :  normal return
*                     ERR_GENERATE  :  an error has occurred
```

```
 *
 *
 *
 *   FUNCTIONS
 *   CALLED:        None.
 *
 *
 *   AUTHOR:        Steven H. Culp
 *
 *
 *   DATE
 *   CREATED:       25Mar88                          Version  1.00
 *
 *
 *   REVISIONS:     None.
 *
 *
*************************************************************/


/* ---------------------------------------------------------- */
/*  The following defs. are for the types of termination.   */
/* ---------------------------------------------------------- */
#define   NORMAL          0
#define   ERR_GENERATE   30


int  generate(command, flag, j_or_ns, ASEL, BSEL, bs, BUS,
              cin, ds, io_sel, io_s_d, i3_0, i5_0, i8_6, immed,
              intack, ldir, ldmar, memsel, mode, NA, pol, rs,
              r_w, s, se, ss, t3_0, te, z_or_one)

/* ---------------------------------------------------------- */
/*    The following variables are defined exactly as in     */
/*      main() and in the thesis.                            */
/* ---------------------------------------------------------- */
char *command;          /*  a ptr. to the command array     */

int  flag, j_or_ns, *ASEL, *BSEL, *bs, *BUS, *cin, *ds,
     *io_sel, *io_s_d, *i3_0, *i5_0, *i8_6, *immed, *intack,
     *ldir, *ldmar, *memsel, *mode, NA, pol, *rs, *r_w, *s,
     *se, *ss, *t3_0, *te, *z_or_one;


/* ---------------------------------------------------------- */
/* Generate function begins here.  In this function, only */
/*   the bits which need to be set or the ones which need */
/*   to be included for uniformity are generated.  All    */
/*   other control bits are assumed to be clear, i.e. set */
/*   equal to 0.  This was performed by the clear().        */
/* -------------------------------------------------------- */
{

switch(command[0])
    {
    case 'A':
        switch(command[1])
            {
            case 'C':                             /*   ACTOBUS      */
                {
                *i5_0      =  100;
                *cin       =  0;
                *z_or_one  =  0;
                *mode      =  0;
                *ASEL      =  11;
                *i8_6      =  0;
                *bs        =  1;
                *io_sel    =  1;
                *io_s_d    =  0;
                break;
                }
            case 'D':                             /*   ADD          */
                {
                *i5_0      =  101;
                *cin       =  1;
                *ss        =  1;
                *BUS       =  1;
                *mode      =  0;
                *ASEL      =  11;
                *i8_6      =  10;
                *BSEL      =  11;
                *memsel    =  1;
                *r_w       =  1;
                break;
                }
            case 'N':                             /*   AND          */
                {
                *i5_0      =  100101;
                *cin       =  0;
                *z_or_one  =  0;
                *ss        =  1;
                *BUS       =  1;
                *mode      =  0;
                *ASEL      =  11;
                *i8_6      =  10;
                *BSEL      =  11;
                *memsel    =  1;
                *r_w       =  1;
                break;
                }
            }

    case 'C':
        switch(command[1])
            {
            case 'L':                             /*  CLR         */
                {
                *i5_0      =  111;
                *cin       =  0;
                *z_or_one  =  0;
                *ss        =  1;
                *mode      =  11;
                *ASEL      =  9999;
                *BUS       =  0;
                *ds        =  0;
                *immed     =  0;
                *i8_6      =  10;
                *BSEL      =  9999;
                *bs        =  0;
                break;
                }
            case 'O':                             /*  COM         */
                {
                *i5_0      =  110101;
                *cin       =  0;
                *z_or_one  =  0;
                *ss        =  1;
                *mode      =  11;
                *ASEL      =  9999;
                *BUS       =  0;
                *se        =  1;
                *ds        =  0;
                *immed     =  11111111;
                *i8_6      =  10;
                *BSEL      =  9999;
                *bs        =  0;
                break;
                }
            }
        break;

    case 'D':
        switch(command[1])
            {
            case 'A':                             /*  DATATOAC    */
                {
                *i5_0      =  111;
                *cin       =  0;
                *z_or_one  =  0;
                *mode     =  0;
                *BUS      =  1;
                *i8_6     =  10;
                *BSEL     =  11;
                *io_sel   =  1;
                *io_s_d   =  0;
                break;
                }
            case 'C':                     /*   DCRTOBUS    */
                {
                *i5_0     =  100;
                *cin      =  0;
                *z_or_one =  0;
                *mode     =  0;
                *ASEL     =  0;
                *i8_6     =  0;
                *bs       =  1;
                *io_sel   =  1;
                *io_s_d   =  0;
                break;
                }
            case 'E':                     /*   DEC         */
                {
                *i5_0     =  1101;
                *cin      =  0;
                *z_or_one =  1;
                *ss       =  1;
                *mode     =  11;
                *ASEL     =  9999;
                *BUS      =  0;
                *ds       =  0;
                *immed    =  00000001;
                *i8_6     =  10;
                *BSEL     =  9999;
                *bs       =  0;
                break;
                }
            }
        break;

    case 'F':                             /*   FORPC       */
        {
        *i5_0             =  111;
        *cin              =  0;
        *z_or_one         =  0;
        *mode             =  0;
        *BUS              =  0;
        *ds               =  0;
        *immed            =  1100100;
        *i8_6             =  11;
    *BSEL                 =  1001;
    *bs                   =  10;
    *memsel               =  1;
    *r_w                  =  0;
    break;
    )

case 'H':                             /*  HLT        */
    {
    break;
    }

case 'I':
    switch(command[2])
        {
        case 'C':
            switch(command[3])
                {
                case ' ':                     /*  INC        */
                    {
                    *i5_0     =  101;
                    *cin      =  0;
                    *z_or_one =  0;
                    *ss       =  1;
                    *mode     =  11;
                    *ASEL     =  9999;
                    *BUS      =  0;
                    *ds       =  0;
                    *immed    =  1;
                    *i8_6     =  10;
                    *BSEL     =  9999;
                    *bs       =  0;
                    break;
                    }
                case 'I':                     /*  INCIASM    */
                    {
                    *i5_0     =  101;
                    *cin      =  0;
                    *z_or_one =  0;
                    *mode     =  0;
                    *ASEL     =  1110;
                    *BUS      =  0;
                    *ds       =  0;
                    *immed    =  10;
                    *i8_6     =  11;
                    *BSEL     =  1110;
                    *bs       =  1;
                    *ldmar    =  1;
                    *intack   =  1;
                    break;
```

```
             }
         case 'P':             /*  INCPCMAR    */
             {
             *i5_0     =  101;
             *cin      =  0;
             *z_or_one =  0;
             *mode     =  0;
             *ASEL     =  1001;
             *BUS      =  0;
             *ds       =  0;
             *immed    =  10;
             *i8_6     =  11;
             *BSEL     =  1001;
             *bs       =  1;
             *ldmar    =  1;
             break;
             }
         case 'S':             /*  INCSPMAR    */
             {
             *i5_0     =  101;
             *cin      =  0;
             *z_or_one =  0;
             *mode     =  0;
             *ASEL     =  1110;
             *BUS      =  0;
             *ds       =  0;
             *immed    =  10;
             *i8_6     =  11;
             *BSEL     =  1110;
             *bs       =  1;
             *ldmar    =  1;
             break;
             }
         }
    break;

case 'I':                     /*  INIT        */
    {
    *i5_0             =  111;
    *cin              =  0;
    *z_or_one         =  0;
    *mode             =  0;
    *BUS              =  0;
    *ds               =  0;
    *immed            =  0;
    *i8_6             =  10;
    *BSEL             =  1001;
    *bs               =  0;
    break;
    }
```

```
        }
    break;

case 'L':
    switch(command[1])
        {
        case 'A':                      /*   LAC        */
            {
            *i5_0             =  111;
            *cin              =  0;
            *z_or_one         =  0;
            *ss               =  1;
            *BUS              =  1;
            *mode             =  0;
            *i8_6             =  10;
            *BSEL             =  11;
            *memsel           =  1;
            *r_w              =  1;
            break;
            }
        case 'I':                      /*   LIRPCINC  */
            {
            *i5_0             =  101;
            *cin              =  0;
            *z_or_one         =  0;
            *mode             =  0;
            *ASEL             =  1001;
            *BUS              =  0;
            *ds               =  0;
            *immed            =  10;
            *i8_6             =  10;
            *BSEL             =  1001;
            *memsel           =  1;
            *r_w              =  1;
            *ldir             =  1;
            break;
            }
        case 'P':
            switch(command[3])
                {
                case ' ':              /*   LPC      */
                    {
                    *i5_0             =  111;
                    *cin              =  0;
                    *z_or_one         =  0;
                    *BUS              =  1;
                    *mode             =  0;
                    *i8_6             =  10;
                    *BSEL             =  1001;
                    *memsel           =  1;
                    *r_w              =  1;
                    break;
                    }
                case 'R':              /*  LPCRS     */
                    {
                    *i5_0             =  111;
                    *cin              =  0;
                    *z_or_one         =  0;
                    *BUS              =  1;
                    *mode             =  0;
                    *i8_6             =  10;
                    *BSEL             =  1001;
                    *memsel           =  1;
                    *r_w              =  1;
                    *rs               =  1;
                    break;
                    }
                }
            break;
        case 'T':                      /*  LTEMP     */
            {
            *i5_0             =  111;
            *cin              =  0;
            *z_or_one         =  0;
            *mode             =  0;
            *BUS              =  1;
            *i8_6             =  10;
            *BSEL             =  100;
            *memsel           =  1;
            *r_w              =  1;
            break;
            }
        }
    break;

case 'N':                               /*  NOP       */
    {
    break;
    }

case 'O':                               /*  OR        */
    {
    *i5_0                     =  11101;
    *cin                      =  0;
    *z_or_one                 =  0;
    *ss                       =  1;
    *BUS                      =  1;
    *mode                     =  0;
    *ASEL                     =  11;
    *i8_6                     =  10;
```

```
        *BSEL                     =  11;
        *memsel                   =  1;
        *r_w                      =  1;
        break;
        }

    case 'P':
        switch(command[2])
            {
            case 'R':                       /*  PCREL    */
                {
                *i5_0             =  101;
                *cin              =  0;
                *z_or_one         =  0;
                *mode             =  0;
                *ASEL             =  1001;
                *BUS              =  0;
                *ds               =  10;
                *se               =  1;
                *i8_6             =  10;
                *BSEL             =  1001;
                *bs               =  0;
                break;
                )
            case 'T':                       /*  PCTEMP1  */
                {
                *i5_0             =  100;
                *cin              =  0;
                *z_or_one         =  0;
                *mode             =  0;
                *ASEL             =  1001;
                *i8_6             =  10;
                *BSEL             =  100;
                *bs               =  0;
                break;
                }
            }
        break;

    case 'R':
        switch(command[2])
            {
            case 'L':                       /*  ROL      */
                {
                *i5_0             =  100;
                *cin              =  1;
                *ss               =  1;
                *mode             =  11;
                *ASEL             =  9999;
                *i8_6             =  110;
                *BSEL             =  9999;
                *bs               =  0;
                break;
                }
            case 'R':                             /*  ROR      */
                {
                *i5_0             =  100;
                *cin              =  1;
                *ss               =  1;
                *mode             =  11;
                *ASEL             =  9999;
                *i8_6             =  100;
                *BSEL             =  9999;
                *bs               =  0;
                break;
                }
            }
        break;

    case 'S':
        switch(command[1])
            {
            case 'A':                             /*  SAC      */
                {
                *i5_0             =  100;
                *cin              =  0;
                *z_or_one         =  0;
                *ss               =  1;
                *mode             =  0;
                *ASEL             =  11;
                *i8_6             =  0;
                *memsel           =  1;
                *r_w              =  0;
                break;
                }
            case 'P':
                switch(command[2])
                    {
                    case 'C':                     /*  SPC      */
                        {
                        *i5_0     =  100;
                        *cin      =  0;
                        *z_or_one =  0;
                        *mode     =  0;
                        *ASEL     =  1001;
                        *i8_6     =  0;
                        *bs       =  1;
                        *memsel   =  1;
                        *r_w      =  0;
                        break;
                    }
                case 'M':               /*  SPMARDEC   */
                    {
                    *i5_0     =  1101;
                    *cin      =  0;
                    *z_or_one =  1;
                    *mode     =  0;
                    *ASEL     =  1110;
                    *BUS      =  0;
                    *ds       =  0;
                    *immed    =  00000010;
                    *i8_6     =  10;
                    *BSEL     =  1110;
                    *bs       =  1;
                    *ldmar    =  1;
                    break;
                    }
                }
            break;
        case 'U':                               /*  SUB        */
            {
            *i5_0                 =  1101;
            *cin                  =  1;
            *ss                   =  1;
            *BUS                  =  1;
            *mode                 =  0;
            *ASEL                 =  11;
            *i8_6                 =  10;
            *BSEL                 =  11;
            *memsel               =  1;
            *r_w                  =  1;
            break;
            }
        }
    break;

case 'T':
    switch(command[1])
        {
        case 'E':
            switch(command[3])
                {
                case 'A':                   /*  TEMAR      */
                    {
                    *i5_0     =  100;
                    *cin      =  0;
                    *z_or_one =  0;
                    *mode     =  0;
                    *ASEL     =  100;
                    *i8_6     =  0;
            *bs       =  1;
            *ldmar    =  1;
            break;
            }
        case 'P':
            switch(command[4])
                {
                case 'A':       /*  TEMPADD  */
                    {
                    *i5_0     =  1;
                    *cin      =  0;
                    *z_or_one =  0;
                    *mode     =  10;
                    *ASEL     =  9999;
                    *BSEL     =  100;
                    *i8_6     =  10;
                    *bs       =  0;
                    break;
                    }
                case 'C':       /*  TEMPC    */
                    {
                    *i5_0     =  1;
                    *cin      =  0;
                    *z_or_one =  0;
                    *mode     =  0;
                    *ASEL     =  1001;
                    *BSEL     =  100;
                    *i8_6     =  10;
                    *bs       =  0;
                    break;
                    }
                }
            break;
        }
    break;
case 'F':
    switch(command[3])
        {
        case 'R':               /*  TFRRR    */
            {
            *i5_0               =  100;
            *cin                =  0;
            *z_or_one           =  0;
            *ss                 =  1;
            *mode               =  11;
            *ASEL               =  9999;
            *i8_6               =  10;
            *BSEL               =  9999;
            *bs                 =  0;
            break;
```

```
             }
         case 'I':                      /*  TFRIA   */
             {
             *i5_0                  =  100;
             *cin                   =  0;
             *z_or_one              =  0;
             *ss                    =  1;
             *mode                  =  10;
             *ASEL                  =  9999;
             *i8_6                  =  10;
             *BSEL                  =  9999;
             *bs                    =  0;
             break;
             }
        }
case 'I':                                /*  TINT     */
    {
    *t3_0                  =  101;
    *i5_0                  =  101;
    *cin                   =  0;
    *z_or_one              =  0;
    *mode                  =  0;
    *ASEL                  =  1001;
    *BUS                   =  0;
    *ds                    =  0;
    *immed                 =  10;
    *i8_6                  =  0;
    *bs                    =  1;
    *ldmar                 =  1;
    break;
    }

case 'N':
    switch(command[3])
        {
        case '0':
            switch(command[4])
                {
                case ' ':                 /*  TNVO    */
                    {
                    *t3_0         =  111;
                    *i5_0         =  101;
                    *cin          =  0;
                    *z_or_one     =  0;
                    *mode         =  0;
                    *ASEL         =  1001;
                    *BUS          =  0;
                    *ds           =  10;
                    *se           =  1;
                    *te           =  1;
                    *i8_6      =  10;
                    *BSEL      =  1001;
                    *bs        =  0;
                    break;
                    }
                case 'N':             /*  TNV0NC */
                    {
                    *t3_0      =  111;
                    }
                }
            break;
            case '1':
                switch(command[4])
                    {
                    case ' ':     /*  TNV1   */
                        {
                        *t3_0     =  111;
                        *i5_0     =  101;
                        *cin      =  0;
                        *z_or_one =  0;
                        *mode     =  0;
                        *ASEL     =  1001;
                        *BUS      =  0;
                        *ds       =  10;
                        *se       =  1;
                        *te       =  1;
                        *i8_6     =  10;
                        *BSEL     =  1001;
                        *bs       =  0;
                        break;
                        }
                    case 'N':     /*  TNV1NC */
                        {
                        *t3_0     =  111;
                        break;
                        }
                    }
                break;
            }
        break;
case 'Z':
    switch(command[2])
        {
        case '0':                     /*  TZ0     */
            {
            *t3_0                     =  100;
            *i5_0                     =  101;
            *cin                      =  0;
            *z_or_one                 =  0;
            *mode                     =  0;
```

```
                        *ASEL                 =  1001;
                        *BUS                  =  0;
                        *ds                   =  10;
                        *se                   =  1;
                        *te                   =  1;
                        *i8_6                 =  10;
                        *BSEL                 =  1001;
                        *bs                   =  0;
                        break;
                        }
                    case '1':                 /*  TZ1     */
                        {
                        *t3_0                 =  100;
                        *i5_0                 =  101;
                        *cin                  =  0;
                        *z_or_one             =  0;
                        *mode                 =  0;
                        *ASEL                 =  1001;
                        *BUS                  =  0;
                        *ds                   =  10;
                        *se                   =  1;
                        *te                   =  1;
                        *i8_6                 =  10;
                        *BSEL                 =  1001;
                        *bs                   =  0;
                        break;
                        }
                    }
        }

    }
/* ------------------------------------------------------ */
/* At this point, all the control bits for the data flow  */
/*   have been generated. All that remains to be performed*/
/*   is to determine the next address control bits.       */
/* ------------------------------------------------------ */
switch(j_or_ns)
    {
    case 0:                        /*  next sequential    */
        {
        *i3_0 =  1110;
        break;
        }
    case 1:                        /*  unconditional jump */
        {
        *i3_0 =  10;
        if (*s == 2)
            *s =  10;
        if (*s == 3)
```

```
            *s =  11;
        NA    =  NA;
        break;
        }
    case 2:                        /*  conditional jump   */
        {
        *i3_0 =  11;
        pol   =  pol;
        if (*s == 2)
            *s =  10;
        if (*s == 3)
            *s =  11;
        NA    =  NA;
        switch(flag)
            {
            case 0:                /*  "1"                */
                {
                *t3_0 =  0;
                break;
                }
            case 1:                /*  carry              */
                {
                *t3_0 =  1;
                break;
                }
            case 2:                /*  overflow           */
                {
                *t3_0 =  10;
                break;
                }
            case 3:                /*  sign               */
                {
                *t3_0 =  11;
                break;
                }
            case 4:                /*  zero               */
                {
                *t3_0 =  100;
                break;
                }
            case 5:                /*  interrupt          */
                {
                *t3_0 =  101;
                break;
                }
            case 6:                /*  I/O ready          */
                {
                *t3_0 =  110;
                break;
                }
            case 7:                /*   N XOR V               */
                {
                *t3_0 =  111;
                break;
                }
            case 8:                /*   Halt                  */
                {
                t3_0 =  1000;
                break;
                }
            break;
            }
        )
    }

return(NORMAL};
}


/*    EOJ  --  Steven H. Culp  --  generate()               */
```

```
/*********************************************************
*
*   SOURCE FILE:  [culp]store.c
*
*
*   FUNCTION:     store.c
*
*
*   DESCRIPTION:  This function stores the generated control
*                     bits into the Control Store memory.
*
*
*   DOCUMENTATION
*   FILES:        None.
*
*
*   ARGUMENTS:
*
*     ASEL        (input)   int
*                 a four-bit field which selects one of the
*                 16 RAM locations to be fed into the A inputs
*                 of the 2901 ALU.
*
*     BSEL        (input)   int
*                 a four-bit field which selects one of the
*                 16 RAM locations to be fed into the B inputs
*                 of the 2901 ALU.  BSEL also functions as the
*                 address of the destination register.
*
*     bs          (input)   int
*                 determine what data is transferred to the BUS
*
*     BUS         (input)   int
*                 indicates whether D inputs to 2901 come from
*                 the BUS or from ds
*
*     cin         (input)   int
*                 indicates whether the carry in value comes
*                 from the CARRY bit or from 0/1
*
*     cs[]        (input)   int
*                 a two-dimensional array which models the
*                 Control Store memory
*
*     cs_loc      (input)   int
*                 the location in the Control Store where the
*                 control bits are to be stored.
*
*     ds          (input)   int
*                 determines the source of the D inputs to
*                    the 2901
*
*    io_sel          (input)   int
*                    determines if a transfer to an I/O device is
*                    to take place.
*
*    io_s_d          (input)   int
*                    determines whether the status of a device
*                    is requested or if data is being sent
*
*    i3_0            (input)   int
*                    four bits which determine the instruction
*                    the 2910 will execute
*
*    i5_0            (input)   int
*                    six bits which determine the source for the
*                    ALU operands and the ALU function
*
*    i8_6            (input)   int
*                    three bits which determine the destination
*                    of the ALU result
*
*    immed           (input)   int
*                    supplies an immediate value for the D inputs
*                    to the 2901
*
*    intack          (input)   int
*                    acknowledges that an I/O device is prompting
*                    to be serviced.
*
*    ldir            (input)   int
*                    indicates that the IR is to be loaded with
*                    the value currently on the BUS
*
*    ldmar           (input)   int
*                    indicates that the MAR is to be loaded with
*                    the value currently on the BUS
*
*    memsel          (input)   int
*                    indicates that a transfer to or from memory
*                    is to take place
*
*    mode            (input)   int
*                    two bits which define the four modes in
*                    which the 2901 ALU is capable of operating
*
*    NA              (input)   int
*                    12-bit field which supplies a potential
*                    address for the CS
*
```

```
*    pol            (input)   int
*                   determines whether positive or negative
*                   polarity is being used
*
*    rs             (input)   int
*                   indicates whether the status register
*                   should be restored
*
*    r_w            (input)   int
*                   indicates whether a read or a write
*                   operation is to take place
*
*    s              (input)   int
*                   determine where the D inputs to the 2910
*                   originate
*
*    se             (input)   int
*                   indicates whether to sign extend the D
*                   inputs to the 2901
*
*    ss             (input)   int
*                   indicates whether current operation should
*                   set the status flags
*
*    t3_0           (input)   int
*                   four bits which indicate the status flag to
*                   be tested
*
*    te             (input)   int
*                   allows the test result to force the D inputs
*                   to zero
*
*    z_or_one       (input)   int
*                   functions as a forced set or clear for cin
*
*
*  RETURN:          int
*                   NORMAL    :  normal return
*                   ERR_STORE :  an error has occurred
*
*
*
*  FUNCTIONS
*  CALLED:          None.
*
*
*  AUTHOR:          Steven H. Culp
*
*
```

```
*   DATE
*   CREATED:       23Mar88                           Version  1.00
*
*
*   REVISIONS:     None.
*
*
***********************************************************/


/* -------------------------------------------------------- */
/*  The following defs. are for the types of termination.  */
/* -------------------------------------------------------- */
#define    NORMAL       0
#define    ERR_STORE   40



int   store(cs, cs_loc, ASEL, BSEL, bs, BUS, cin, ds, io_sel,
            io_s_d, i3_0, i5_0, i8_6, immed, intack, ldir,
            ldmar, memsel, mode, NA, pol, rs, r_w, s, se, ss,
            t3_0, te, z_or_one)


int     cs[4001][88],   /*  ptr. to Control Store array         */
        cs_loc,         /*  location of the Control Store       */
        ASEL,           /*  The following are the control       */
        BSEL,           /*    bits and are defined as before.  */
        bs,
        BUS,
        cin,
        ds,
        io_sel,
        io_s_d,
        i3_0,
        i5_0,
        i8_6,
        immed,
        intack,
        ldir,
        ldmar,
        memsel,
        mode,
        NA,
        pol,
        rs,
        r_w,
        s,
        se,
        ss,
        t3_0,
        te,
        z_or_one;



/* ------------------------------------------------------- */
/*    Store fcn. begins here.                              */
/* ------------------------------------------------------- */
{

cs[cs_loc][0]  =   s;
cs[cs_loc][1]  =   NA;
cs[cs_loc][2]  =   i3_0;
cs[cs_loc][3]  =   pol;
cs[cs_loc][4]  =   t3_0;
cs[cs_loc][5]  =   immed;
cs[cs_loc][6]  =   ds;
cs[cs_loc][7]  =   BUS;
cs[cs_loc][8]  =   se;
cs[cs_loc][9]  =   te;
cs[cs_loc][10] =   ASEL;
cs[cs_loc][11] =   BSEL;
cs[cs_loc][12] =   mode;
cs[cs_loc][13] =   i8_6;
cs[cs_loc][14] =   i5_0;
cs[cs_loc][15] =   z_or_one;
cs[cs_loc][16] =   cin;
cs[cs_loc][17] =   ss;
cs[cs_loc][18] =   bs;
cs[cs_loc][19] =   ldir;
cs[cs_loc][20] =   memsel;
cs[cs_loc][21] =   r_w;
cs[cs_loc][22] =   ldmar;
cs[cs_loc][23] =   io_sel;
cs[cs_loc][24] =   io_s_d;
cs[cs_loc][25] =   intack;
cs[cs_loc][26] =   rs;

return(NORMAL);
} .


/*   EOJ  --  Steven H. Culp  --  store()                 */
```

MICROPROGRAMMING A PROPOSED
16-BIT STACK MACHINE

by

STEVEN HOWARD CULP

B.A., Mid-America Nazarene College, 1985

AN ABSTRACT OF A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Electrical and Computer Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1988

ABSTRACT

In 1982, Dr. Don Rhea Hush presented a thesis which contained the complete design of a 16-bit, educational computer system. The ALU hardware was to be implemented using the Am2901 Four-Bit Bipolar Microprocessor Slice chip and the Am2910 Microprogram Controller chip generated the next address for the Control Unit. The Control Store was specified to have 4k locations which were 88 bits wide. Once built, the machine could be used as an excellent educational tool for areas involved with computer design, instruction sets, and microprogramming.

This paper presents an instruction set and the corresponding microcode that must be generated in order to implement the instruction set on Dr. Hush's machine. To this end, an ILL (Intermediate Level Language) was created which symbolized the required microcode and a program was then written which generated the microcode from the ILL.

Inclusive to this thesis are the instruction set to be implemented, the devised ILL, a complete listing of the bit specifications for the microcode and the program that converts the individual ILL statements to their particular microinstructions.