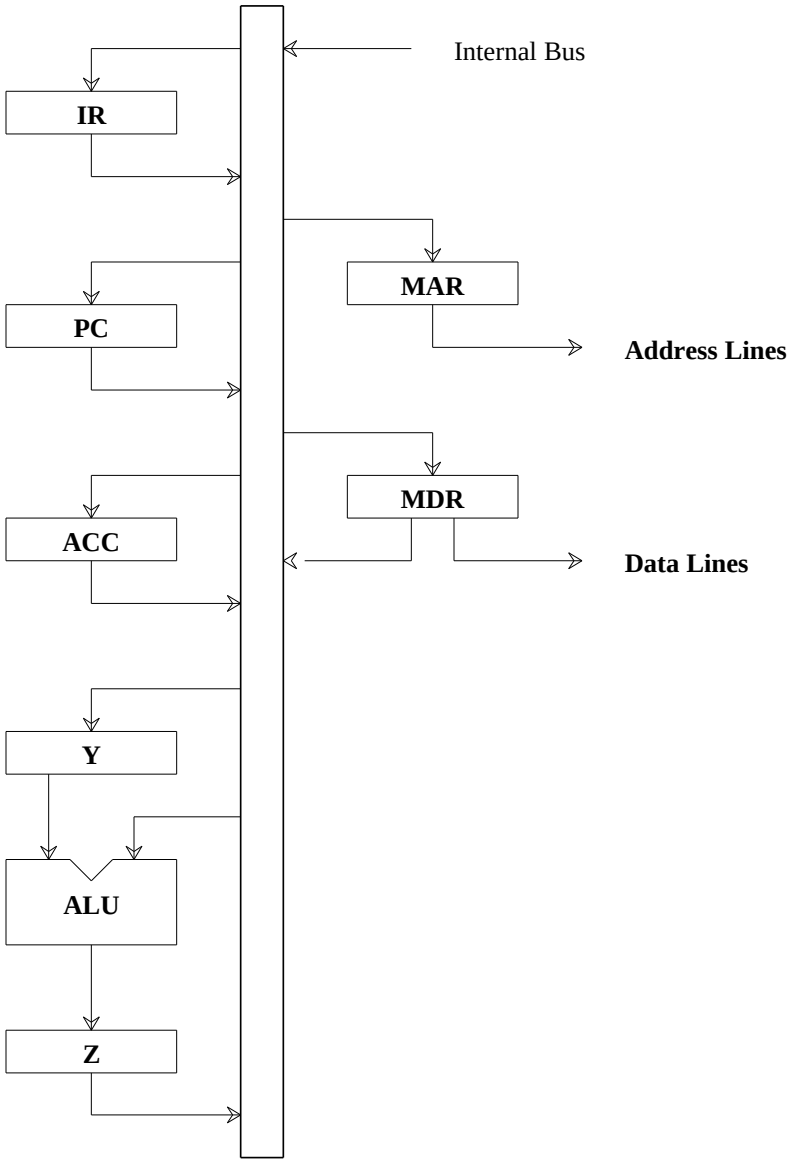
Simple Accumulator Machine

An Example of Hardwired Control

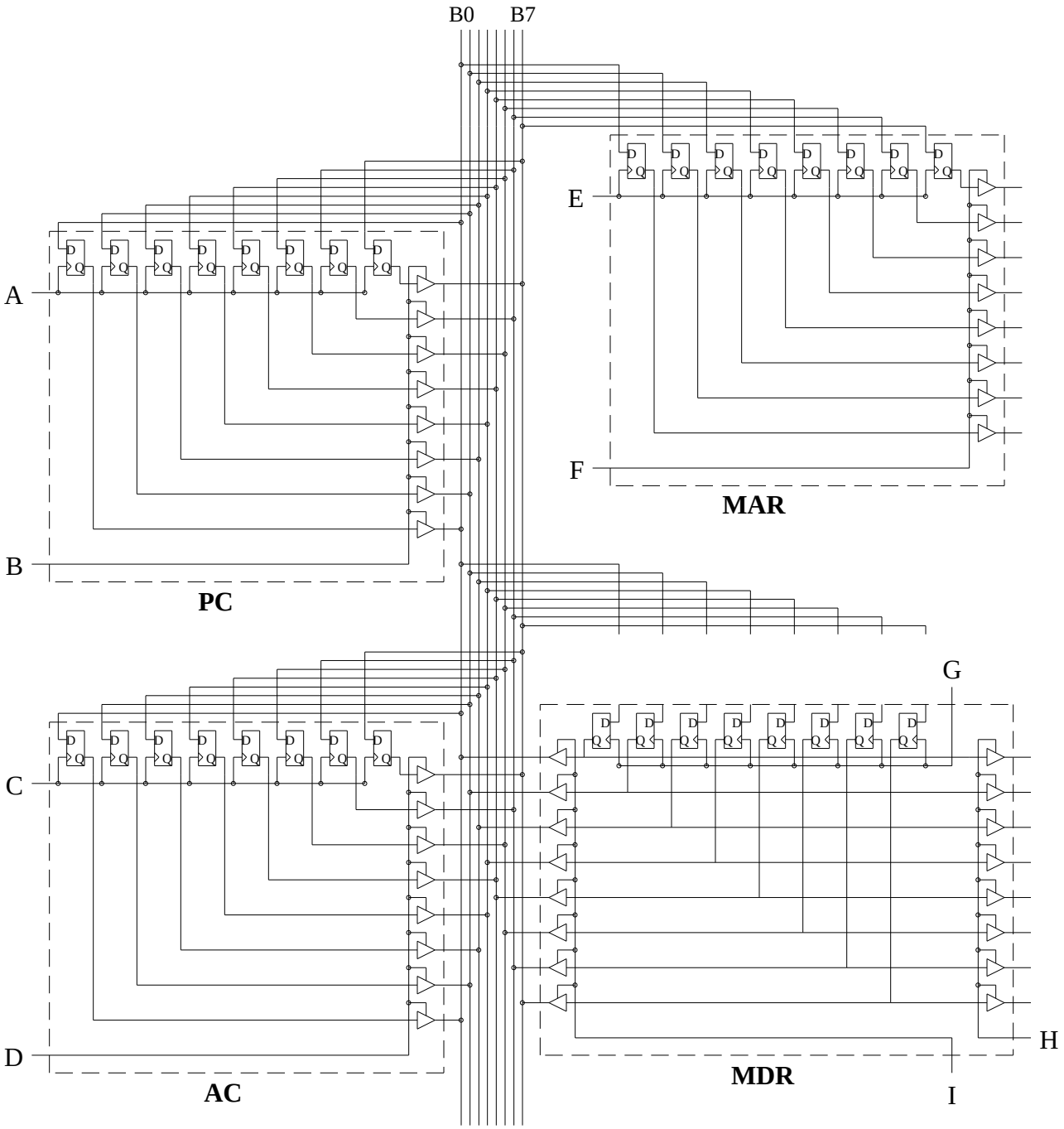
The purpose of this example is to show how one might develop hardwired control logic for a **S**imple **A**ccumulator **M**achine. While this is a very simple machine, it is as much a computer as anything else you might use or purchase. It would just run a little slower. (OK, a *lot* slower.)

A block diagram of the **SAM** architecture is presented below.

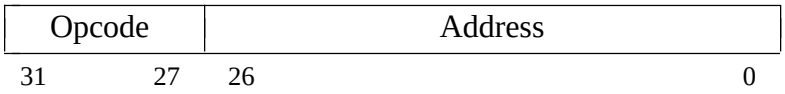


Detailed Section of SAM's Datapath

The diagram below shows in more detail a subsection of the datapath of SAM. (For the sake of clarity, only 8 bit registers and bus are shown, even though SAM actually uses 32 bits.) Note that in this example, the registers are built out of D-type latches, and that tri-state buffers are necessary to prevent more than one register from driving the bus at a time. You should be able to figure out what the signals **A,B,C,** etc. are. (For example, **A** would be connected to **PCin**, right?)



The instruction set of this machine has only one instruction format:



SAM has a single general purpose register, the *accumulator*, and most instructions use this accumulator as one of the operands. The instruction set is presented in the following table.

Instruction	**Action**
LOAD	Acc ← M[Address]
STORE	M[Address ← [Acc]
ADD	Acc ← [Acc] + M[Address]
SUB	Acc ← [Acc] - M[Address]
AND	Acc ← Bitwise AND of [Acc] and M[Address]
OR	Acc ← Bitwise OR of [Acc] and M[Address]
CMPL	Acc ← Bitwise complement of [Acc]
ROTL	Acc ← Rotate left of [Acc]
ROTR	Acc ← Rotate right of [Acc]
SHL	Acc ← Shift left of [Acc], zero fill
SHR	Acc ← Shift right of [Acc], zero fill
ASHR	Acc ← Arithmetic shift right of [Acc]
BREQ	Branch if [Acc] = Zero
BRNEQ	Branch if [Acc] ≠ Zero
BRN	Branch if [Acc] < Zero
BRC	Branch if Carry bit is set
BRV	Branch if Overflow bit is set
JMP	Unconditional jump to Address
JMPACC	Unconditional jump to [Acc]
JSR	Put the return address in Acc and jump to Address
CLRACC	Acc ← Zero (Clear the Accumulator)
RESET	Acc ← Zero; PC ← Zero (Clear the Accumulator and PC)

Branches use condition codes and are set every time the Accumulator is written or loaded. The condition code bits are carry (**C**), negative (**N**), zero (**Z**), and overflow (**V**). The C and V bits can only be set when an ADD or SUB instruction is performed; in all other cases, when the Accumulator is written these bits are cleared. In cases where a memory address is needed, it is taken from the address field of the instruction (the bottom 27 bits of the IR). If an address is not needed for an instruction, these bits are ignored. Keep in mind that this instruction set, while simple, should be able to execute any and all programs.

The control signals necessary to implement the previously described instruction set are presented below:

SAM Control Signals

PC_{in}	Z_{in}	ROTL
PC_{out}	Z_{out}	ROTR
IR_{in}	WMFC	SHL
IR_{out}	READ	SHR
MAR_{in}	WRITE	ASHR
MDR_{in}	ADD	Clear-Y
MDR_{out}	SUB	Clear-PC
ACC_{in}	AND	Clear-ACC
ACC_{out}	OR	Set-Carry-In
Y_{in}	CMPL	END

Associated with each of these signals is a Boolean equation that determines when the signal is active (set high). To generate these equations, we must first write down the sequence of operations that must occur in order to execute each instruction. Each step in the sequence corresponds to a time unit or clock tick.

The first three steps for each instruction are identical, and are as follows:

Step	Action
T1	PC_{out}, MAR_{in}, READ, Clear-Y, Set-Carry-In, ADD, Z_{in}
T2	Z_{out}, PC_{in}, Wait for MFC
T3	MDR_{out}, IR_{in}

The remaining steps for a given instruction vary by instruction type. Instructions can be grouped into 10 types. The control sequences for each of these are presented on the following page (each starts with T4).

Operations Involving Memory (MEMOPs): ADD, SUB, AND and OR

Step	Action
T4	IR_{out}, MAR_{in}, READ
T5	ACC_{out}, Y_{in}, Wait for MFC (WFMC)
T6	MDR_{out}, Z_{in}, OPERATION
T7	Z_{out}, ACC_{in}, END

Operations Not Involving Memory (NMOPs): CMPL, ROTL, ROTR, SHR, SHL, ASHR

Step	Action
T4	ACC_{out}, Z_{in}, OPERATION
T5	Z_{out}, ACC_{in}, END

LOAD

Step	Action
T4	IR_{out}, MAR_{in}, READ, WMFC
T5	MDR_{out}, ACC_{in}, END

STORE

Step	Action
T4	IR_{out}, MAR_{in}
T5	ACC_{out}, MDR_{in}, WRITE
T6	WMFC, END

JMP

Step	Action
T4	IR_{out}, PC_{in}, END

JMPACC

Step	Action
T4	ACC_{out}, PC_{in}, END

JSR

Step	Action
T4	PC_{out}, ACC_{in}
T5	IR_{out}, PC_{in}, END

CLRACC

Step	Action
T4	Clear-ACC, END

RESET

Step	Action
T4	Clear-PC, Clear-ACC, END

BRANCH

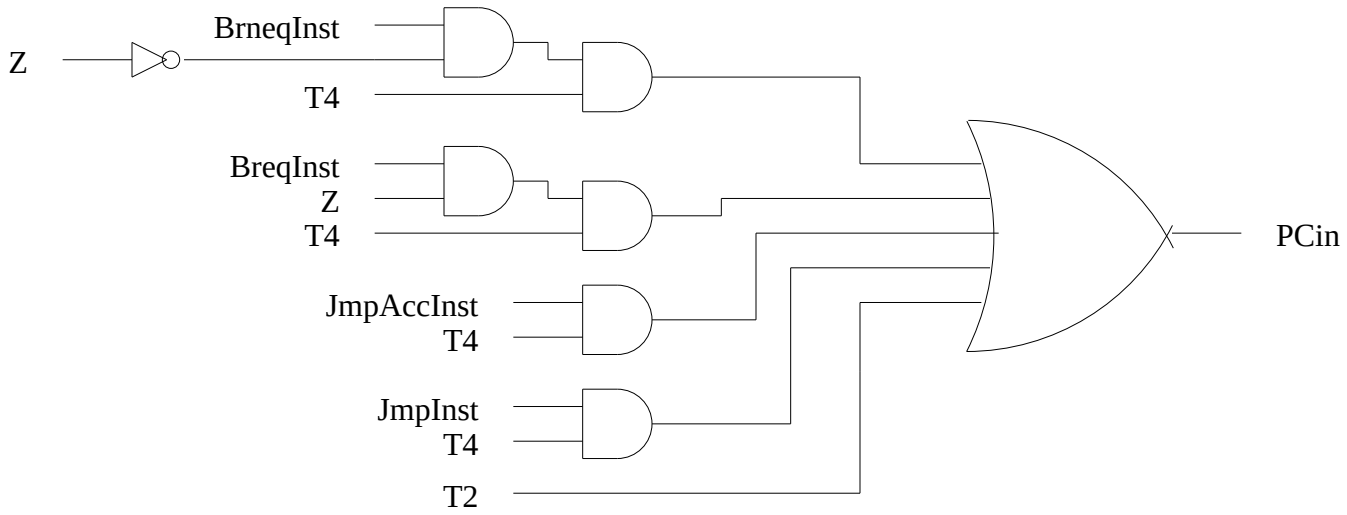
Step	Action
T4	If *cond* THEN (IR_{out}, PC_{in}), END where *cond* is Z, Z, N, V or C

Given these control sequences we can now calculate when a control signal becomes activated. One way to do this is to tabulate the information, placing on one side of the table the control signals, and across the top of the table the time step. During T1, for example, certain signals are active for all instructions (since this is part of the instrution fetch sequence). The same is true of T2 and T3. Thus, **All** can be written into the locations for these signals.

Fill out the enclosed table, in order to generate the necessary control signals. In the table, mnemonics will be used in order to make it clear that certain signals are coming from the decoder (for example, AddInst indicates an ADD instruction, LoadInst is a LOAD instruction, etc.) In addition, in order to save space and make the table legible, several of the signals will be grouped as follows:

MEMOPS = AddInst + SubInst + AndInst + OrInst
NMOPS = CmplInst + RotlInst + RotrInst + ShrInst + ShlInst + AshrInst
BRANCH = (BreqInst • Z) + (BrneqInst • $\overline{Z}$) + (BrcInst • C) + (BrvInst • V)

Once this table has been filled out, the boolean equations for each control signal can be written by simply reading the rows of the table. So, going through the table, we can fill in the equations on the following page. Using these equations we can design the hardware for each signal (an example is below).



PC_{in} = T2 + T4·[JmpInst+JmpAccInst+(BreqInst·Z)+BrneqInst·$\bar{Z}$

PC_{out} =

IR_{in} =

IR_{out} =

MAR_{in} =

MDR_{in} =

MDR_{out} =

ACC_{in} =

ACC_{out} =

Y_{in} =

Z_{in} =

Z_{out} =

WMFC =

READ =

WRITE =

ADD =

SUB =

AND =

OR =

CMPL =

ROTL =

ROTR =

SHL =

SHR =

ASHR =

Clear-Y =

Clear-PC =

Clear-ACC =

Set-Carry-In =

END =

Simple Accumulator Machine

An Example of Microcoded Control

We have now described in relative detail how to create a hardwired implementation of SAM's control signals. Next we will show how to create these control signals using **Microcode**. Microcode is a very simple concept, which recognizes the step-by-step nature of generating control signals. For example, looking back at the steps on page 4 and 5 necessary to execute an instruction, we see a sequence of steps executed one after another, just like in programs. So - what if we assigned each control signal to a bit in a word, and then generated the control signals by stepping through a memory containing the control words in the proper order? What would this entail?

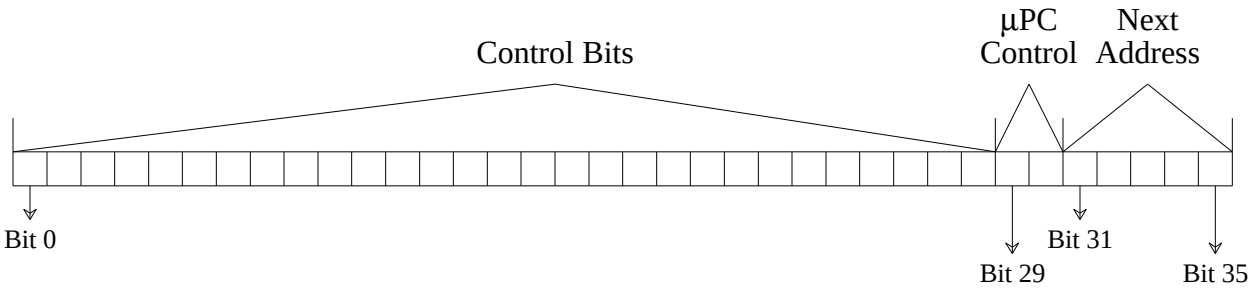
Step 1: Assign control signals to bits within a word.

Step 2: Decide how to sequence through the micro-instructions. This step implies a **micro**-PC, which performs the same function at the microcode level that the regular PC does at the assembly language level.

It can easily be argued that it is more than a 2-step process, since quite a bit of stuff has been lumped into each step. It is an arguement that you would probably win. Regardless, the next 4 pages contain an example assignment of control signals to bits in a control word, an example MicroEngine that is responsible for the sequencing and selection of the control words, a detailed example of one section of the MicroEngine, an example assignment of opcodes to assembly language instructions, and the contents of the control memory that would result given these design parameters.

You should understand that this is certainly not the *only* way to do this. However, by being fairly clever in the way opcodes are selected and how branches are recognized, a significant savings in Control Store memory can be achieved.

SAM 1-Address Control Word Example



Example MicroControl Word

Control Word Assignments

Register Output

Bit	Signal
0	PC_{out}
1	IR_{out}
2	MDR_{out}
3	ACC_{out}
4	Z_{out}

Register Input

Bit	Signal
5	PC_{in}
6	IR_{in}
7	MAR_{in}
8	MDR_{in}
9	ACC_{in}
10	Y_{in}
11	Z_{in}

ALU Control

Bit	Signal
12	ADD
13	SUB
14	AND
15	OR
16	CMPL
17	ROTL
18	ROTR
19	SHL
20	SHR
21	ASHR

Memory Interface

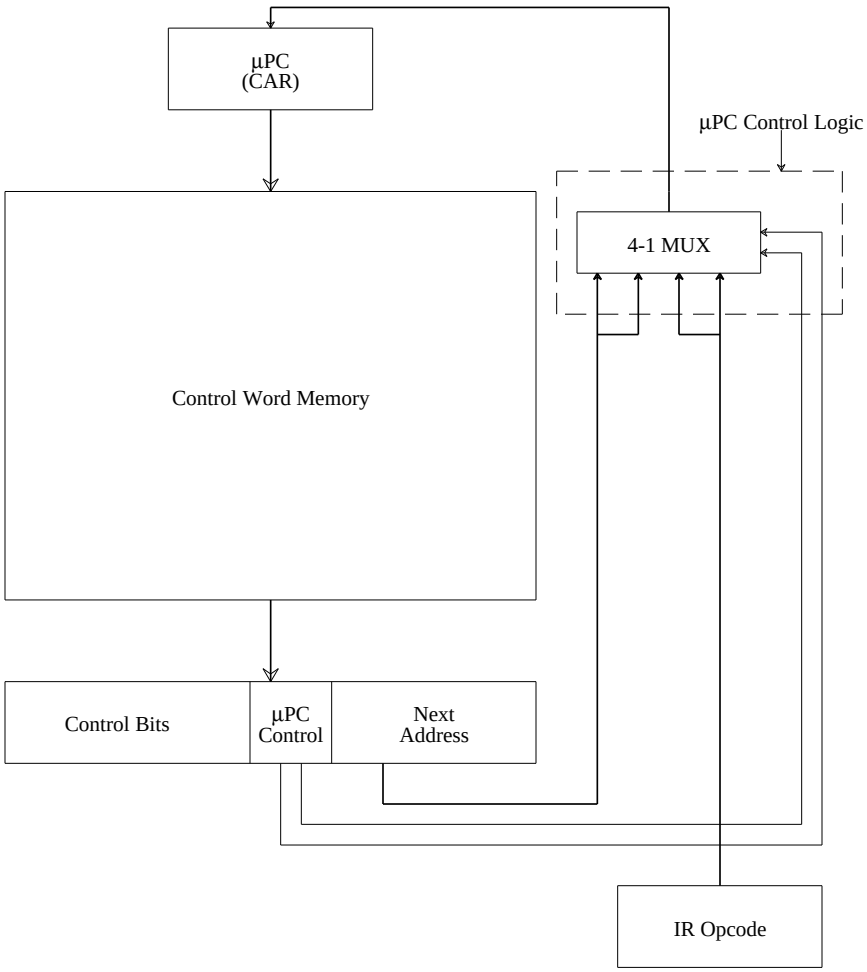
Bit	Signal
22	WMFC
23	READ
24	WRITE

Miscellaneous

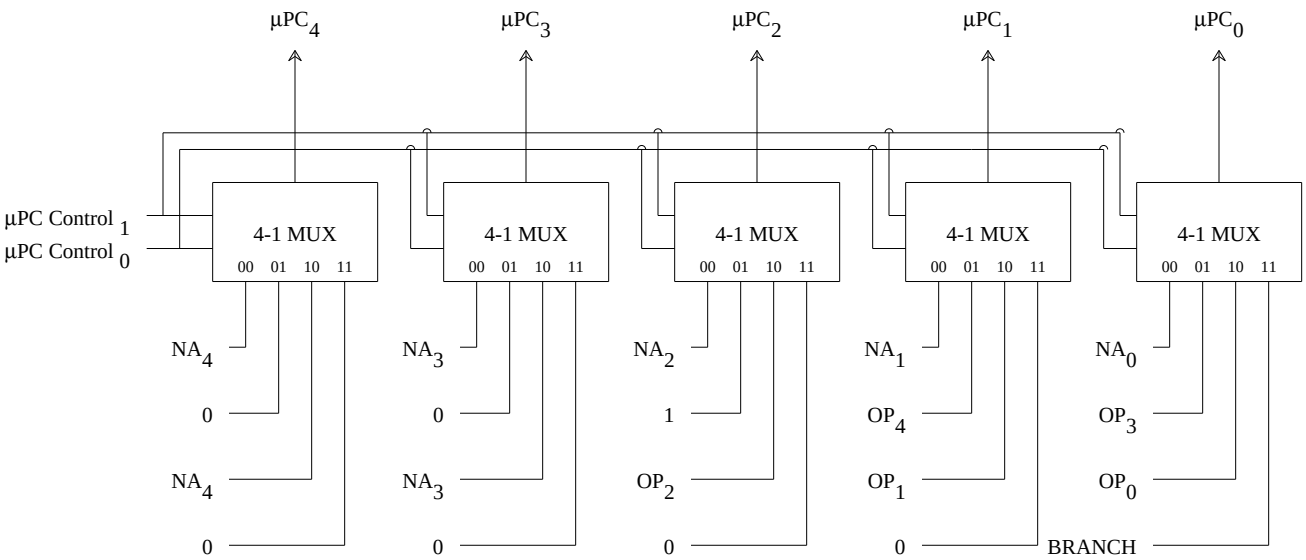
Bit	Signal
25	Clear-Y
26	Clear-PC
27	Clear-ACC
28	Set-Carry-In

μPC Control

Bit 29	Bit 30	Result
0	0	μPC ← Next Address
0	1	μPC ← 0 0 1 OP_4 OP_3
1	0	μPC ← NA_4 NA_3 OP_2 OP_1 OP_0
0	1	if BRANCH then μPC ← 0 0 0 0 1 else μPC ← Next Address



Block Diagram of MicroEngine



Details of µPC Control Logic

Opcode Assignments for 1-Address Machine

Instruction	Opcode	Action
LOAD	00110	Acc ← M[Address]
STORE	00111	M[Address ← [Acc]
ADD	11000	Acc ← [Acc] + M[Address]
SUB	11001	Acc ← [Acc] - M[Address]
AND	11010	Acc ← Bitwise AND of [Acc] and M[Address]
OR	11011	Acc ← Bitwise OR of [Acc] and M[Address]
CMPL	10000	Acc ← Bitwise complement of [Acc]
ROTL	10001	Acc ← Rotate left of [Acc]
ROTR	10010	Acc ← Rotate right of [Acc]
SHL	10011	Acc ← Shift left of [Acc], zero fill
SHR	10100	Acc ← Shift right of [Acc], zero fill
ASHR	10101	Acc ← Arithmetic shift right of [Acc]
BREQ	01000	Branch if [Acc] = Zero
BRNEQ	01001	Branch if [Acc] ≠ Zero
BRN	01010	Branch if [Acc] < Zero
BRC	01011	Branch if Carry bit is set
BRV	01100	Branch if Overflow bit is set
JMP	00100	Unconditional jump to Address
JMPACC	00011	Unconditional jump to [Acc]
JSR	00010	Put the return address in Acc and jump to Address
CLRACC	00001	Acc ← Zero (Clear the Accumulator)
RESET	00000	Acc ← Zero; PC ← Zero (Clear the Accumulator and PC)

Contents of Control Memory

Control Mem Address	Control Bits					μPC Control	Next Address	Action
00000	10000	0010001	1000000000	010	1001	00	00010	T1 of IFetch
00001	01000	1000000	0000000000	000	0000	00	00000	Branch Taken
00010	00001	1000000	0000000000	100	0000	00	00011	T2 of IFetch
00011	00100	0100000	0000000000	000	0000	01	00000	T3 of IFetch
00100	00000	0000000	0000000000	000	0000	10	10000	Jump to Misc Inst
00101	00000	0000000	0000000000	000	0000	11	00000	Evaluate Branch
00110	00000	0000000	0000000000	000	0000	10	01000	Jump to NMOPS
00111	01000	0010000	0000000000	010	0000	00	01111	T4 of MEMOPS
01000	00010	0000001	0000100000	000	0000	00	01110	T4 of CMPL
01001	00010	0000001	0000010000	000	0000	00	01110	T4 of ROTL
01010	00010	0000001	0000001000	000	0000	00	01110	T4 of ROTR
01011	00010	0000001	0000000100	000	0000	00	01110	T4 of SHL
01100	00010	0000001	0000000010	000	0000	00	01110	T4 of SHR
01101	00010	0000001	0000000001	000	0000	00	01110	T4 of ASHR
01110	00001	0000100	0000000000	000	0000	00	00000	T5 of NMOPS; T7 of MEMOPS
01111	00010	0000010	0000000000	100	0000	10	11000	T5 of MEMOPS
10000	00000	0000000	0000000000	000	0110	00	00000	RESET
10001	00000	0000000	0000000000	000	0010	00	00000	CLRACC
10010	10000	0000100	0000000000	000	0000	00	10100	JSR - Finish in JMP
10011	00010	1000000	0000000000	000	0000	00	00000	JMPACC
10100	01000	1000000	0000000000	000	0000	00	00000	JMP
10101	00100	0000100	0000000000	000	0000	00	00000	End of Load (T5)
10110	01000	0010000	0000000000	110	0000	00	10101	LOAD (T4)
10111	01000	0010000	0000000000	000	0000	00	11100	STORE (T4)
11000	00100	0000001	1000000000	000	0000	00	01110	T6 of ADD
11001	00100	0000001	0100000000	000	0000	00	01110	T6 of SUB
11010	00100	0000001	0010000000	000	0000	00	01110	T6 of AND
11011	00100	0000001	0001000000	000	0000	00	01110	T6 of OR
11100	00010	0001000	0000000000	001	0000	00	11101	T5 of STORE
11101	00000	0000000	0000000000	100	0000	00	00000	T6 of STORE
11110	xxxxx	xxxxxxx	xxxxxxxxxx	xxx	xxxx	xx	xxxxx	Unused
11111	xxxxx	xxxxxxx	xxxxxxxxxx	xxx	xxxx	xx	xxxxx	Unused