

## Lecture #9 – Microcode Control Unit

- Microprogrammed Control Unit

A microprogram corresponding to each instruction is stored in ROM

The Micro Control Unit (MCU) executes the appropriate microprogram based on the Instruction Register (IR).

Each microinstruction contains all the information to directly operate the ALU and BUS transfers needed.

A microinstruction is equivalent to a minor cycle in the hardwired control unit.

The microinstructions are stored in ROM that is very fast and integrated with the processor. This is called the **microstore**.

Because the microcode is not fixed in hardware, it is more flexible and also does not need fixed size major cycles.

However, microcode still requires a  $\mu$ MAR and a  $\mu$ MBR. Thus the total throughput is generally slower than hardwired control unit.

The Intel Pentium processors use a combination of hardwired control and microcode. The hardwired control is for simple instructions, and the microcode is for complex instructions.

- Micro-Instructions – Maximally versus Minimally Encoding

In the simplest example, the each memory output line is one control signal. But with a complex processor this can lead to many extra outputs that produce “illegal states”, such as Register A to BUS and Register B to BUS, and Register C to BUS. This is 3 control signals, but only one can be active at a time.

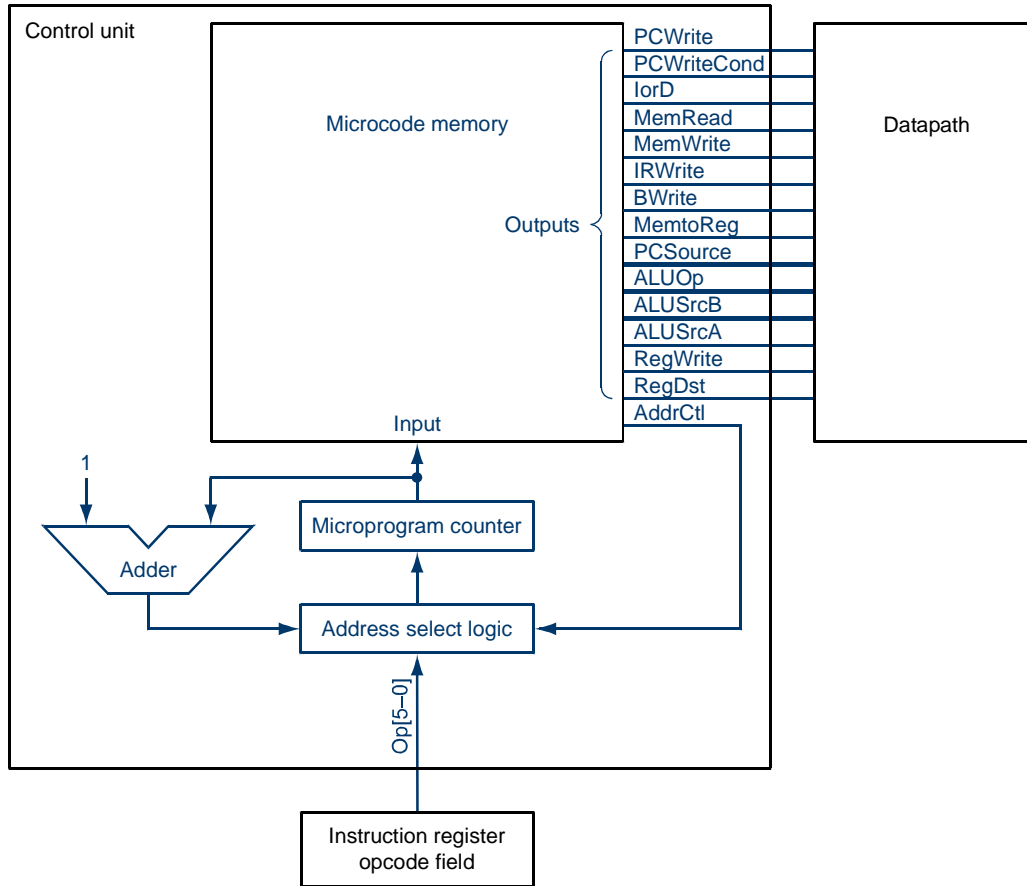
Microcode word size can be reduced with encoding. This as:

00 – No connection  
01 – Register A to BUS  
10 – Register B to BUS  
11 – Register C to BUS

Three outputs were reduced to 2, with no reduction in functionality. A 2-to-4 decoder can re-generate the 3 separate control lines, but with a slight performance penalty.

- Micro-Instructions – MIPS Example

The textbook uses a control unit that has the following configuration:



- Micro-Instructions – Microinstruction format

PCWrite	– 1 bit	MemToReg	– 1 bit
PCWriteCond	– 1 bit	PCSource	– 2 bits
IorD	– 1 bit	ALUOp	– 2 bits
MemRead	– 1 bit	ALUSrcB	– 2 bits
MemWrite	– 1 bit	ALUSrcA	– 1 bit
IRWrite	– 1 bit	RegWrite	– 1 bit
BWrite	– 1 bit	RegDst	– 1 bit
AddrCtl	– 2 bits		

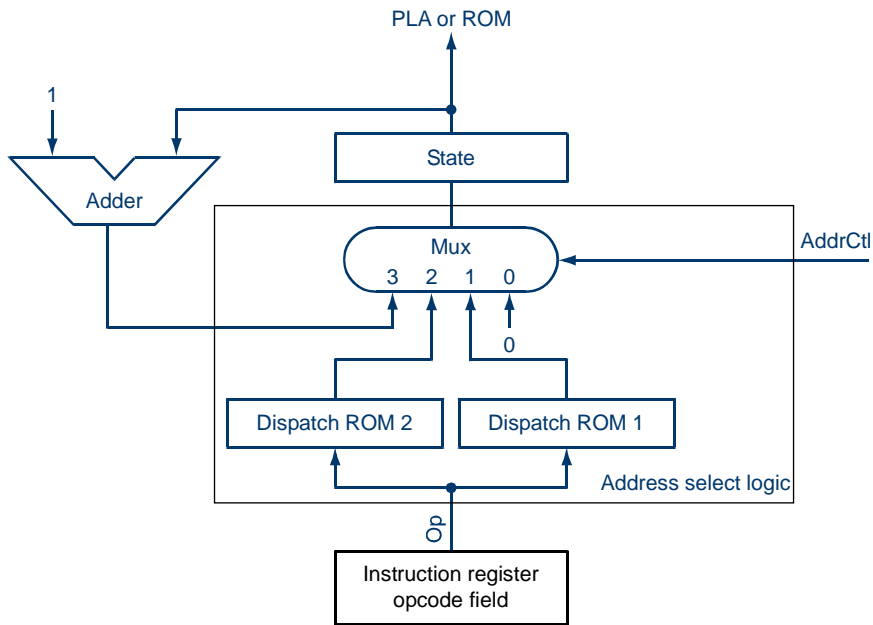
Total = 19 bits wide of control.

Field name	Value	Signals active	Comment
ALU control	Add	ALUOp = 00	Cause the ALU to add.
	Subt	ALUOp = 01	Cause the ALU to subtract; this implements the compare for branches.
	Func code	ALUOp = 10	Use the instruction's function code to determine ALU control.
SRC1	PC	ALUSrcA = 0	Use the PC as the first ALU input.
	A	ALUSrcA = 1	Register A is the first ALU input.
	B	ALUSrcB = 00	Register B is the second ALU input.
SRC2	4	ALUSrcB = 01	Use 4 as the second ALU input.
	Extend	ALUSrcB = 10	Use output of the sign extension unit as the second ALU input.
	Extshft	ALUSrcB = 11	Use the output of the shift-by-two unit as the second ALU input.
Register control	Read		Read two registers using the rs and rt fields of the IR as the register numbers and putting the data into registers A and B.
	Write ALU	RegWrite, RegDst = 1, MemtoReg = 0	Write a register using the rd field of the IR as the register number and the contents of the ALUOut as the data.
	Write MDR	RegWrite, RegDst = 0, MemtoReg = 1	Write a register using the rt field of the IR as the register number and the contents of the MDR as the data.
Memory	Read PC	MemRead, lorD = 0	Read memory using the PC as address; write result into IR (and the MDR).
	Read ALU	MemRead, lorD = 1	Read memory using the ALUOut as address; write result into MDR.
	Write ALU	MemWrite, lorD = 1	Write memory using the ALUOut as address, contents of B as the data.
PC write control	ALU	PCSource = 00 PCWrite	Write the output of the ALU into the PC.
	ALUOut-cond	PCSource = 01, PCWriteCond	If the Zero output of the ALU is active, write the PC with the contents of the register ALUOut.
	jump address	PCSource = 10, PCWrite	Write the PC with the jump address from the instruction.
Sequencing	Seq	AddrCtl = 11	Choose the next microinstruction sequentially.
	Fetch	AddrCtl = 00	Go to the first microinstruction to begin a new instruction.
	Dispatch 1	AddrCtl = 01	Dispatch using the ROM 1.
	Dispatch 2	AddrCtl = 10	Dispatch using the ROM 2.

- Micro-Instructions – MIPS implementation

Label	ALU control	SRC1	SRC2	Register control	Memory	PCWrite control	Sequencing
Fetch	Add	PC	4		Read PC	ALU	Seq
	Add	PC	Extshft	Read			Dispatch 1
Mem1	Add	A	Extend				Dispatch 2
LW2					Read ALU		Seq
				Write MDR			Fetch
SW2					Write ALU		Fetch
Rformat1	Func code	A	B				Seq
				Write ALU			Fetch
BEQ1	Subt	A	B			ALUOut-cond	Fetch
JUMP1						Jump address	Fetch

Dispatch ROM 1			Dispatch ROM 2		
Op	Opcode name	Value	Op	Opcode name	Value
000000	R-format	0110	100011	lw	0011
000010	jmp	1001	101011	sw	0101
000100	beq	1000			
100011	lw	0010			
101011	sw	0010			



State number	Address-control action	Value of AddrCtl
0	Use incremented state	3
1	Use dispatch ROM 1	1
2	Use dispatch ROM 2	2
3	Use incremented state	3
4	Replace state number by 0	0
5	Replace state number by 0	0
6	Use incremented state	3
7	Replace state number by 0	0
8	Replace state number by 0	0
9	Replace state number by 0	0

- Micro-Instructions – Example #2

Each microinstruction is 21 bits long.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----

For the ASC there are two types of microinstruction:

0	Control Signals
---	-----------------

1	Condition	Branch address
---	-----------	----------------

The control signal microinstruction is the one that does all of the work.

Microinstructions can be decoded directly into register transfer operations.

Even though there is a  $\mu$ MAR and  $\mu$ MBR the MCU does not have a separate control unit or instruction register. Rather it decodes directly from the  $\mu$ MBR.

- Control Microinstructions

The 20 bits of control signals are broken up into fields that are all executed simultaneously:

BUS1	3 bits	Determines which register to write onto BUS1
BUS2	3 bits	Determines which register to write onto BUS2
BUS3	3 bits	Determines which register to read from BUS3.
FIELD1	2 bits	Used for I/O.
FIELD2	2 bits	Used for I/O.
FIELD3	2 bits	Used for I/O.
ALU	3 bits	Specifies which operation the ALU should perform.
READ	1 bit	Tells memory to READ into MBR.
WRITE	1 bit	Tells memory to WRITE from MBR.

- Branch Microinstructions

The branch microinstruction is for conditional microcode.

In this microinstruction, the fields are different, but can still be directly decoded.

The 20 bits of branch signals are:

CONDITION 3 bits                      Branch condition for the microinstruction

ADDRESS 7 bits                      Branch location for the microinstruction

NOT USED 10 bits

In this case the address is for the next **microinstruction** and is placed into the  $\mu$ MAR.

There is one special condition, the GOTO\* OPCODE. This condition does not actually use a branch address, but rather computes the branch address from the opcode.

- Example Microcode

The FETCH logic of the MCU is the following:

- $MAR \leftarrow PC, \text{READ}$
- $PC \leftarrow PC + 1$
- $IR \leftarrow MBR$
- GOTO\* OPCODE

Notice that in this case the FETCH routine does not compute the MAR for the instruction, but rather leaves that to be done during the instruction EXECUTE routine.

The microcode for these instructions would be:

T	BUS1	BUS2	BUS3	ALU				MEM
0	PC	None	MAR	TRA1	None	None	None	READ
0	PC	1	PC	ADD	None	None	None	None
0	None	MBR	IR	TRA2	None	None	None	None
T	COND	ADDRESS		NOT USED				
1	OPCODE	None		None				

In binary this translates to

<b>T</b>	<b>BUS1</b>	<b>BUS2</b>	<b>BUS3</b>	<b>ALU</b>				<b>MEM</b>
0	100	000	100	001	00	00	00	01
0	100	011	110	011	00	00	00	00
0	000	010	011	010	00	00	00	00
<b>T</b>	<b>COND</b>	<b>ADDRESS</b>			<b>NOT USED</b>			
1	001	0000000			0000000000			

So the microprogram in HEX is:

```

082101
08F300
009A00
120000

```