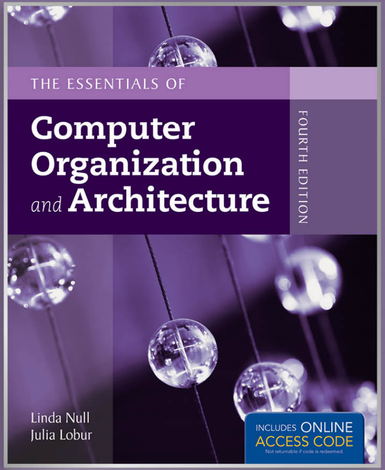
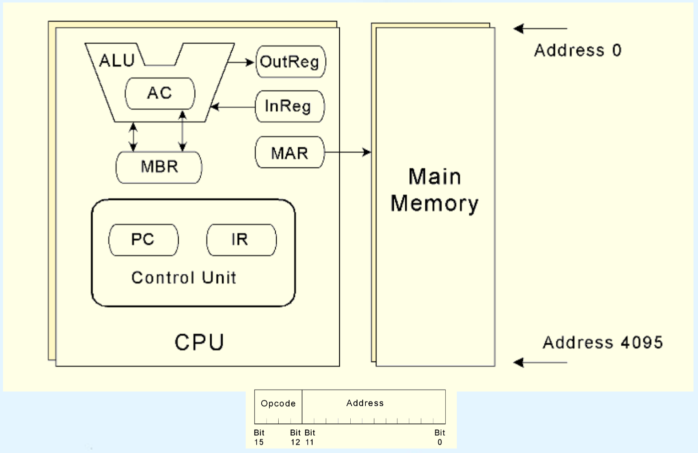
Chapter 4

MARIE: An Introduction to a Simple Computer


THE ESSENTIALS OF
Computer Organization and Architecture
FOURTH EDITION
Linda Null
Julia Lobur
INCLUDES ONLINE ACCESS CODE

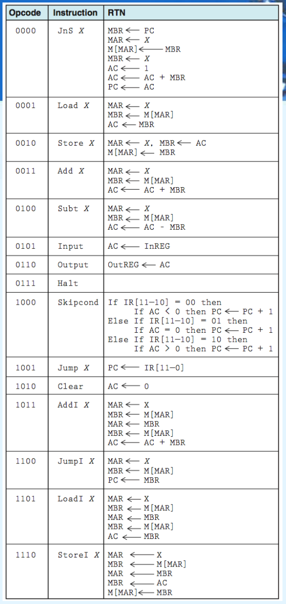
4.8 MARIE

This is the MARIE architecture shown graphically.



4.8 MARIE

MARIE's Full Instruction Set



Opcode	Instruction	RTN

0000	JnS *X*	MBR ← PC
		MAR ← *X*
		M[MAR] ← MBR
		MBR ← *X*
		AC ← 1
		AC ← AC + MBR
		PC ← AC

0001	Load *X*	MAR ← *X*
		MBR ← M[MAR]
		AC ← MBR

0010	Store *X*	MAR ← *X*, MBR ← AC
		M[MAR] ← MBR

0011	Add *X*	MAR ← *X*
		MBR ← M[MAR]
		AC ← AC + MBR

0100	Subt *X*	MAR ← *X*
		MBR ← M[MAR]
		AC ← AC - MBR

0101	Input	AC ← InREG
0110	Output	OutREG ← AC
0111	Halt	
1000	Skipcond	If IR[11–10] = 00 then If AC < 0 then PC ← PC + 1 Else If IR[11–10] = 01 then If AC = 0 then PC ← PC + 1 Else If IR[11–10] = 10 then If AC > 0 then PC ← PC + 1
1001	Jump *X*	PC ← IR[11–0]
1010	Clear	AC ← 0

1011	AddI *X*	MAR ← X
		MBR ← M[MAR]
		MAR ← MBR
		MBR ← M[MAR]
		AC ← AC + MBR

1100	JumpI *X*	MAR ← *X*
		MBR ← M[MAR]
		PC ← MBR

1101	LoadI *X*	MAR ← X
		MBR ← M[MAR]
		MAR ← MBR
		MBR ← M[MAR]
		AC ← MBR

1110	StoreI *X*	MAR ← X
		MBR ← M[MAR]
		MAR ← MBR
		MBR ← AC
		M[MAR] ← MBR

4.13 A Discussion on Decoding

- A computer's control unit keeps things synchronized, making sure that bits flow to the correct components as the bits are needed.
- There are two general ways in which a control unit can be implemented: *microprogrammed control* and *hardwired control*.
  - With microprogrammed control, a small program is placed into read-only memory.
  - Hardwired controllers implement this program using digital logic components.

4.13 A Discussion on Decoding

- Your text provides a complete list of the register transfer language for each of MARIE's instructions.
- The microoperations given by each RTL define the operation of MARIE's control unit.
- Each microoperation consists of a distinctive signal pattern that is interpreted by the control unit.
  - Recall, the RTL for the **Add** instruction is:

```
MAR ← X
MBR ← M[MAR]
AC ← AC + MBR
```

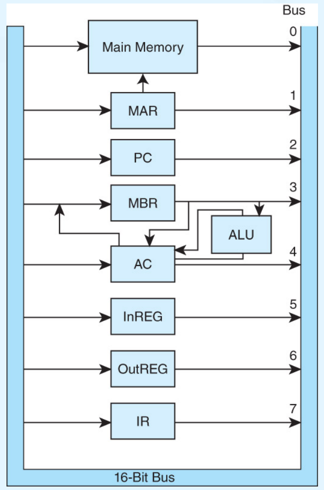
4.13 A Discussion on Decoding

This is the MARIE data path shown graphically.

Data and instructions are transferred using a common bus.

Some additional pathways speed up computation.

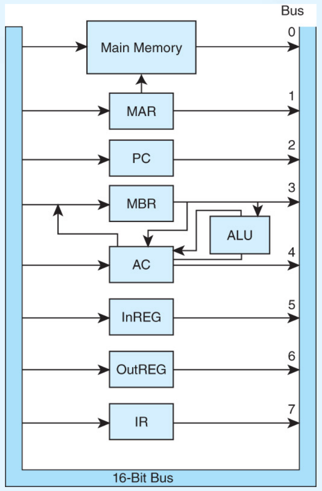
Data can be put on the common bus in the same clock cycle in which data can be put on these other pathways (allowing these events to take place in parallel).



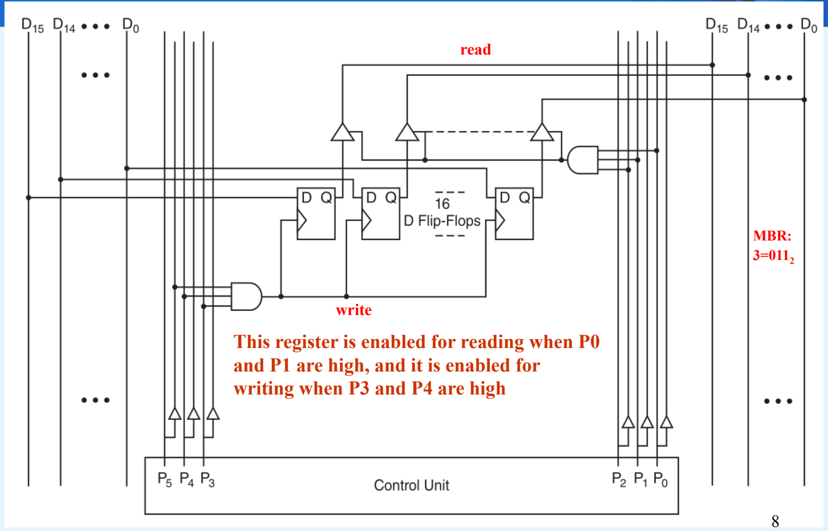
4.13 A Discussion on Decoding

- Let us define two sets of three signals.
- One set, P_2, P_1, P_0, controls reading from memory or a register, and the other set consisting of P_5, P_4, P_3, controls writing to memory or a register.

The next slide shows a close up view of MARIE's MBR.



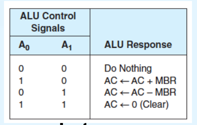
4.13 A Discussion on Decoding



This register is enabled for reading when P0 and P1 are high, and it is enabled for writing when P3 and P4 are high

4.13 A Discussion on Decoding

- Careful inspection of MARIE's RTL reveals that the ALU has only three operations: add, subtract, and clear.
  - We will also define a fourth "do nothing" state.



ALU Control Signals		
A_0	A_1	ALU Response
0	0	Do Nothing
1	0	AC ← AC + MBR
0	1	AC ← AC – MBR
1	1	AC ← 0 (Clear)

- The entire set of MARIE's control signals consists of:
  - Register controls: P_0 through P_5, M_R and M_W.
  - ALU controls: A_0, A_1, and L_{ALT} to control the ALU's data source.
  - Timing: T_0 through T_7 and counter reset C_r

The counter is reset if fewer than eight clock cycles are needed for an instruction

4.13 A Discussion on Decoding

- Consider MARIE's **Add** instruction. Its RTL is:

```
MAR ← X
MBR ← M[MAR]
AC ← AC + MBR
```

- After an **Add** instruction is fetched, the address, X, is in the rightmost 12 bits of the IR, which has a datapath address of 7 ($=111_2$).
- X is copied to the MAR, which has a datapath address of 1 ($=001_2$).
- Thus we need to raise signals P_2, P_1, and P_0 to read from the IR, and raise signal P_3 to write to the MAR.

4.13 A Discussion on Decoding

- Here is the complete signal sequence for MARIE's **Add** instruction (the fetch uses clock cycles T_0, T_1, and T_2):

$P_3\ P_2\ P_1\ P_0\ T_3$: MAR ← X
$P_4\ P_3\ T_4\ M_R$: MBR ← M[MAR]
$C_r\ A_0\ P_5\ T_5\ L_{ALT}$: AC ← AC + MBR
[Reset counter]

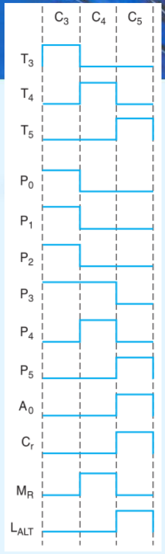
- These signals are ANDed with combinational logic to bring about the desired machine behavior.
- The next slide shows the timing diagram for this instruction.

IR: 111, MAR: 001, Memory: 000, MBR: 011, AC: 100

4.13 A Discussion on Decoding

- Notice the concurrent signal states during each machine cycle: C_3 through C_5.

$P_3\ P_2\ P_1\ P_0\ T_3$: MAR ← X
$P_4\ P_3\ T_4\ M_R$: MBR ← M[MAR]
$C_r\ A_0\ P_5\ T_5\ L_{ALT}$: AC ← AC + MBR
[Reset counter]



4.13 A Discussion on Decoding

To increment the PC we introduce a new control signal **incrPC**.

Then the signal states for the fetch are:

$P_3\ P_1\ T_0$: $MAR \leftarrow PC$

$P_5\ P_4\ P_3\ T_1\ M_R$: $IR \leftarrow M[MAR]$

$incrPC\ T_2$: $PC \leftarrow PC + 1$

[Reset counter]

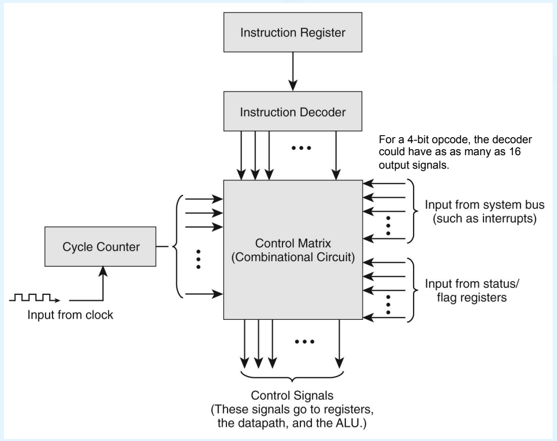
IR: 111, MAR: 001, Memory: 000, PC: 010

4.13 A Discussion on Decoding

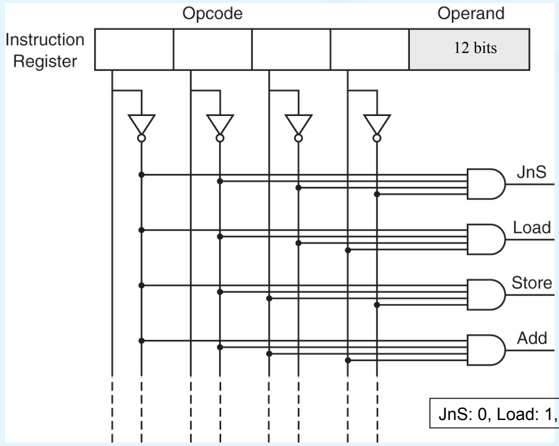
- We note that the signal pattern just described is the same whether our machine used hardwired or microprogrammed control.
- In *hardwired control*, the bit pattern of machine instruction in the IR is decoded by combinational logic.
- The decoder output works with the control signals of the current system state to produce a new set of control signals.

A block diagram of a hardwired control unit is shown on the following slide.

4.13 A Discussion on Decoding


Instruction Register
Instruction Decoder
For a 4-bit opcode, the decoder could have as as many as 16 output signals.
Input from system bus (such as interrupts)
Cycle Counter
Control Matrix (Combinational Circuit)
Input from status/ flag registers
Input from clock
Control Signals (These signals go to registers, the datapath, and the ALU.)

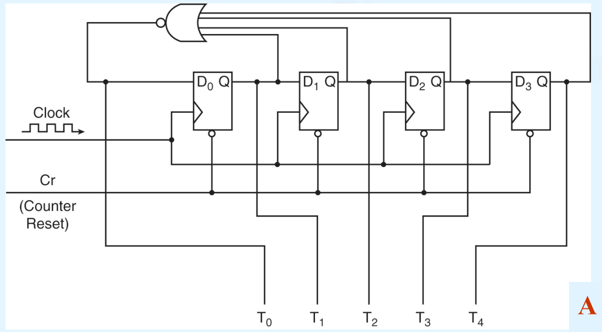
4.13 A Discussion on Decoding



JnS: 0, Load: 1, Store: 2, Add: 3

MARIE's instruction decoder. (Partial)

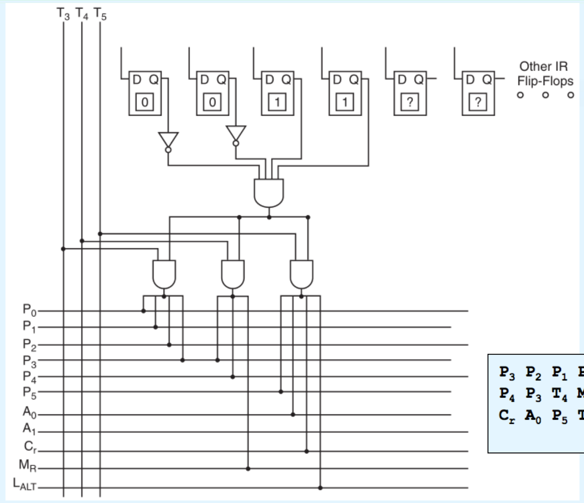
4.13 A Discussion on Decoding



A ring counter that counts from 0 to 5

Initially, all of the flip-flop inputs are low except from the input to D_0 (because of the inverted OR gate on the other outputs). This bit circulates through the ring.

4.13 A Discussion on Decoding



This is the hardwired logic for MARIE's Add = 0011 instruction.

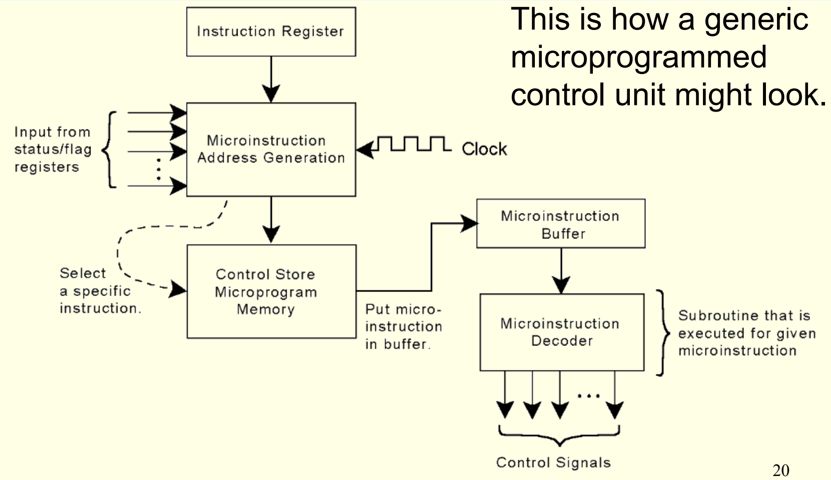
```
P3 P2 P1 P0 T3   : MAR ← X
P4 P3 T4 MR      : MBR ← M[MAR]
Cr A0 P5 T5 LALT : AC ← AC + MBR
                     [Reset counter]
```

4.13 A Discussion on Decoding

- In microprogrammed control, microcode instructions produces control signal changes.
- Machine instructions are the input for a microprogram that converts the 1s and 0s of an instruction into control signals.
- The microprogram is stored in firmware (ROM, PROM, or EPROM), which is also called the control store.
- A microcode instruction is retrieved during each clock cycle.

ROM: Read-Only Memory PROM: Programmable ROM EPROM: Erasable PROM

4.13 A Discussion on Decoding


Instruction Register
Input from status/flag registers
Microinstruction Address Generation
Clock
Select a specific instruction.
Control Store Microprogram Memory
Put micro-instruction in buffer.
Microinstruction Buffer
Microinstruction Decoder
Subroutine that is executed for given microinstruction
Control Signals

This is how a generic microprogrammed control unit might look.

4.13 A Discussion on Decoding

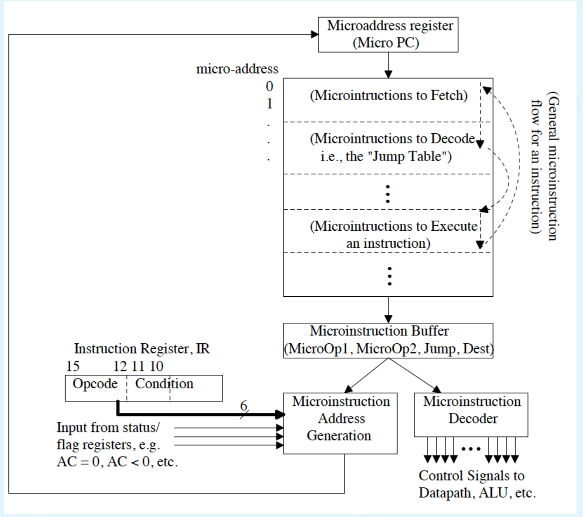
- If MARIE were microprogrammed, the microinstruction format might look like this:

MicroOp1	MicroOp2	Jump	Dest
17 13	12 8	7	6 0

- **`MicroOp1`** and **`MicroOp2`** contain binary codes for each instruction. **`Jump`** is a single bit indicating that the value in the **`Dest`** field is a valid address and should be placed in the microsequencer (circuitry that that serves as the program counter).

Because some micro-instructions can be done in parallel (on the same tick of the clock), we will allow for two micro-operations to be carried out on each tick of the clock.

4.13 A Discussion on Decoding


Microaddress register (Micro PC)
micro-address
0
1
(Microintructions to Fetch)
(Microintructions to Decode i.e., the "Jump Table")
(Microintructions to Execute an instruction)
(General microinstruction flow for an instruction)
Microinstruction Buffer (MicroOp1, MicroOp2, Jump, Dest)
Instruction Register, IR
15 12 11 10
Opcode
Condition
6
Input from status/ flag registers, e.g. AC = 0, AC < 0, etc.
Microinstruction Address Generation
Microinstruction Decoder
Control Signals to Datapath, ALU, etc.

4.13 A Discussion on Decoding

- The table below contains MARIE's microoperation codes along with the corresponding RTL:

MicroOp Code	Microoperation	MicroOp Code	Microoperation
00000	NOP	01101	MBR ← M[MAR]
00001	AC ← 0	01110	OutREG ← AC
00010	AC ← MBR	01111	PC ← IR[11-0]
00011	AC ← AC - MBR	10000	PC ← MBR
00100	AC ← AC + MBR	10001	PC ← PC + 1
00101	AC ← InREG	10010	If AC = 0
00110	IR ← M[MAR]	10011	If AC > 0
00111	M[MAR] ← MBR	10100	If AC < 0
01000	MAR ← IR[11-0]	10101	If IR[11-10] = 00
01001	MAR ← MBR	10110	If IR[11-10] = 01
01010	MAR ← PC	10111	If IR[11-10] = 10
01011	MAR ← X	11000	If IR[15-12] = MicroOp2[4-1]
01100	MBR ← AC		

4.13 A Discussion on Decoding

- The first nine lines of MARIE's microprogram are given below (using RTL for clarity):

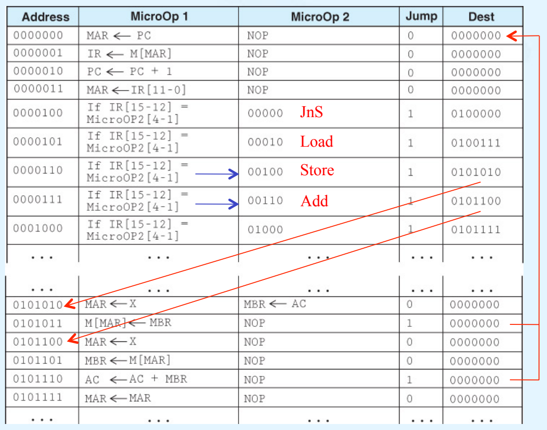
Address	MicroOp 1	MicroOp 2	Jump	Dest
0000000	MAR ← PC	NOP	0	0000000
0000001	IR ← M[MAR]	NOP	0	0000000
0000010	PC ← PC + 1	NOP	0	0000000
0000011	MAR ← IR[11-0]	NOP	0	0000000
0000100	If IR[15-12] = MicroOp2[4-1]	00000	1	0100000
0000101	If IR[15-12] = MicroOp2[4-1]	00010	1	0100111
0000110	If IR[15-12] = MicroOp2[4-1]	00100	1	0101010
0000111	If IR[15-12] = MicroOp2[4-1]	00110	1	0101100
0001000	If IR[15-12] = MicroOp2[4-1]	01000	1	0101111
...	...	...	...	...

4.13 A Discussion on Decoding

- The first four lines initiates the fetch-decode-execute cycle.
- The remaining lines are the beginning of a jump table.

Address	MicroOp 1	MicroOp 2	Jump	Dest
0000000	MAR ← PC	NOP	0	0000000
0000001	IR ← M[MAR]	NOP	0	0000000
0000010	PC ← PC + 1	NOP	0	0000000
0000011	MAR ← IR[11-0]	NOP	0	0000000
0000100	If IR[15-12] = MicroOp2[4-1]	00000	1	0100000
0000101	If IR[15-12] = MicroOp2[4-1]	00010	1	0100111
0000110	If IR[15-12] = MicroOp2[4-1]	00100	1	0101010
0000111	If IR[15-12] = MicroOp2[4-1]	00110	1	0101100
0001000	If IR[15-12] = MicroOp2[4-1]	01000	1	0101111
...	...	...	...	...

4.13 A Discussion on Decoding



Address	MicroOp 1	MicroOp 2	Jump	Dest
0000000	MAR ← PC	NOP	0	0000000
0000001	IR ← M[MAR]	NOP	0	0000000
0000010	PC ← PC + 1	NOP	0	0000000
0000011	MAR ← IR[11-0]	NOP	0	0000000
0000100	If IR[15-12] = MicroOP2[4-1]	00000 JnS	1	0100000
0000101	If IR[15-12] = MicroOP2[4-1]	00010 Load	1	0100111
0000110	If IR[15-12] = MicroOP2[4-1]	00100 Store	1	0101010
0000111	If IR[15-12] = MicroOP2[4-1]	00110 Add	1	0101100
0001000	If IR[15-12] = MicroOP2[4-1]	01000	1	0101111
...	...	...	...	...
...	...	...	...	...
0101010	MAR ← X	MBR ← AC	0	0000000
0101011	M[MAR] ← MBR	NOP	1	0000000
0101100	MAR ← X	NOP	0	0000000
0101101	MBR ← M[MAR]	NOP	0	0000000
0101110	AC ← AC + MBR	NOP	1	0000000
0101111	MAR ← MAR	NOP	0	0000000
...	...	...	...	...

4.13 A Discussion on Decoding

- The microinstructions for the **Store** and **Add** instructions.

	Address	MicroOp 1	MicroOp 2	Jump	Dest
Store	0101010	MAR ← X	MBR ← AC	0	0000000
	0101011	M[MAR] ← MBR	NOP	1	0000000
Add	0101100	MAR ← X	NOP	0	0000000
	0101101	MBR ← M[MAR]	NOP	0	0000000
	0101110	AC ← AC + MBR	NOP	1	0000000

4.13 A Discussion on Decoding

- It's important to remember that a microprogrammed control unit works like a system in miniature.
- Microinstructions are fetched, decoded, and executed in the same manner as regular instructions.
- This extra level of instruction interpretation is what makes microprogrammed control slower than hardwired control.
- The advantages of microprogrammed control are that it can support very complicated instructions and only the microprogram needs to be changed if the instruction set changes (or an error is found).

Microprogrammed control dominates the personal computer market.

4.14 Real World Architectures

- MARIE shares many features with modern architectures but it is not an accurate depiction of them.
- In the following slides, we briefly examine two machine architectures.
- We will look at an Intel architecture, which is a CISC machine and MIPS, which is a RISC machine.
  - CISC is an acronym for complex instruction set computer.
  - RISC stands for reduced instruction set computer.

We delve into the "RISC versus CISC" argument in Chapter 9.

4.14 Real World Architectures

- The classic Intel architecture, the 8086, first member of the x86 family, was born in 1979. It is a CISC architecture.
- It was adopted by IBM for its famed PC (later dubbed the XT), which was released in 1981.
- The 8086 operated on 16-bit data words and supported 20-bit memory addresses.
- Later, to lower costs, the 8-bit bus 8088 was introduced. Like the 8086, it used 20-bit memory addresses.

What was the largest memory that the 8086 could address?

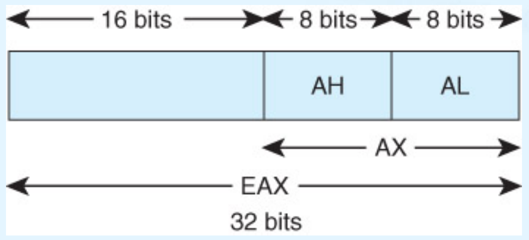
4.14 Real World Architectures

- The 8086 had four 16-bit general-purpose registers (AX, BX, CX, and DX) that could be accessed by the half-word.
- It also had a flags register, an instruction register, and a stack accessed through the values in two other registers, the stack pointer (SP) and the base pointer (BP). SP was used to reference the top of the stack.
  BP was used to reference parameters pushed onto the stack.
- The 8086 had no built in floating-point processing.
- In 1980, Intel released the 8087 numeric coprocessor, but few users elected to install them because of their cost.

4.14 Real World Architectures

- In 1985, Intel introduced the 32-bit 80386.
  An "E" prefix (which stood for "extended") was added to the register names (EAX, EBX, ECX, and EDX).
- It also had no built-in floating-point unit.
- The 80486, introduced in 1989, was an 80386 that had built-in floating-point processing and cache memory.
- The 80386 and 80486 offered downward compatibility with the 8086 and 8088.
- Software written for the smaller word systems was directed to use the lower 16 bits of the 32-bit registers.

4.14 Real World Architectures



EAX register, broken into parts

4.14 Real World Architectures

- Intel's Pentium 4 (2000) introduced a brand new NetBurst architecture.
- Speed enhancing features include:
  - Hyperthreading
  - Hyperpipelining
  - Wider instruction pipeline
  - Execution trace cache (holds decoded instructions for possible reuse) multilevel cache and instruction pipelining.
- Intel, along with many others, is marrying many of the ideas of RISC architectures with microprocessors that are largely CISC.

4.14 Real World Architectures

- The MIPS family of CPUs has been one of the most successful in its class.
- In 1986 the first MIPS CPU was announced.
- It had a 32-bit word size and could address 4GB of memory.
- Over the years, MIPS processors have been used in general purpose computers as well as in games (Nintendo and Sony).
- The MIPS architecture now offers 32- and 64-bit versions.

4.14 Real World Architectures

- MIPS was one of the first RISC microprocessors.
- The original MIPS architecture had only 55 different instructions, as compared with the 8086 which had over 100.
- MIPS was designed with performance in mind: It is a *load/store* architecture, meaning that only the load and store instructions can access memory.
- The large number of registers in the MIPS architecture keeps bus traffic to a minimum.

How does this design affect performance?

Chapter 4 Conclusion

- The major components of a computer system are its control unit, registers, memory, ALU, and data path.
- A built-in clock keeps everything synchronized.
- Control units can be microprogrammed or hardwired.
- Hardwired control units give better performance, while microprogrammed units are more adaptable to changes.

Chapter 4 Conclusion

- Computers run programs through iterative fetch-decode-execute cycles.
- Computers can run programs that are in machine language.
- An assembler converts mnemonic code to machine language.
- The Intel architecture is an example of a CISC architecture; MIPS is an example of a RISC architecture.

End of Chapter 4