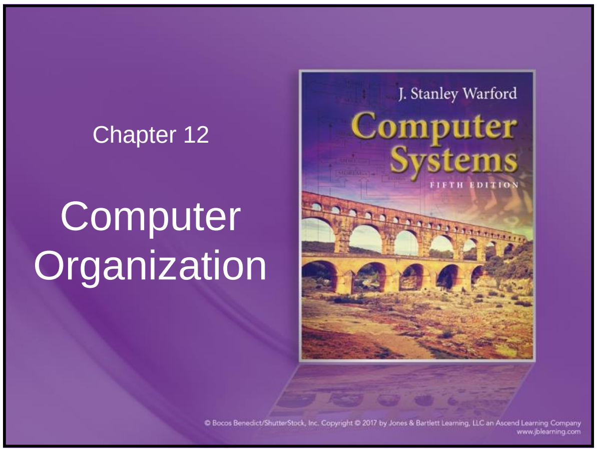
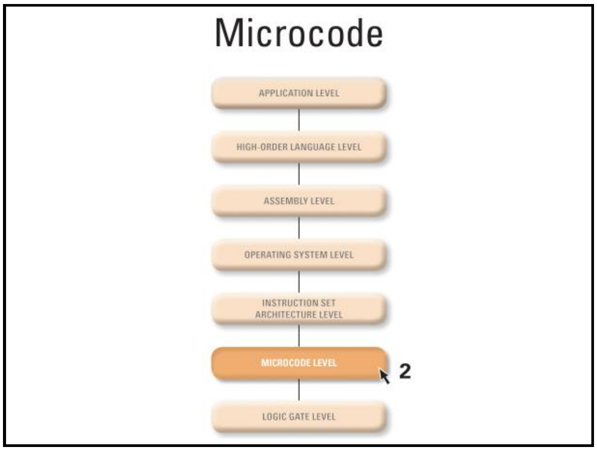
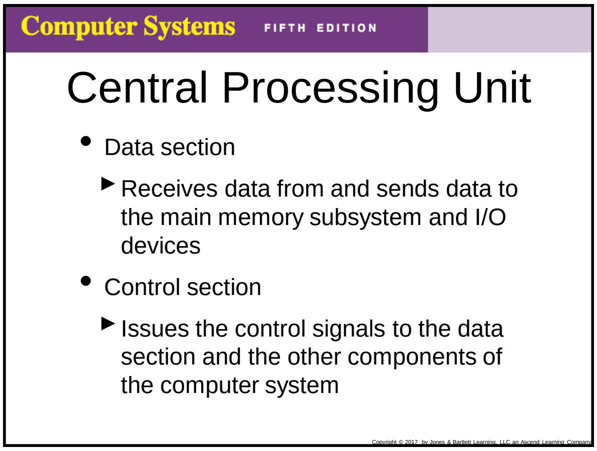


Chapter 12

Computer Organization



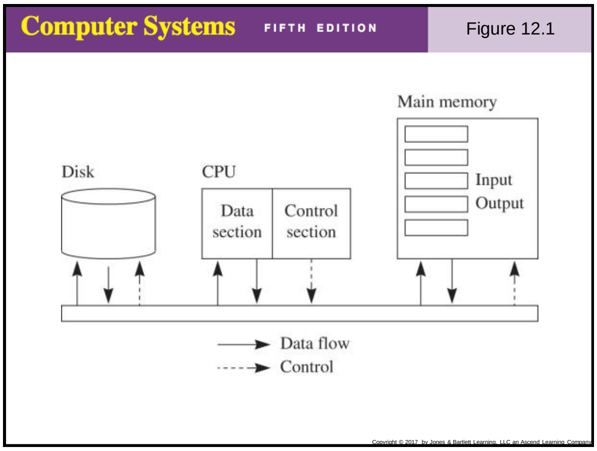
Microcode



Computer Systems FIFTH EDITION

Central Processing Unit

- Data section
  - Receives data from and sends data to the main memory subsystem and I/O devices
- Control section
  - Issues the control signals to the data section and the other components of the computer system



Computer Systems FIFTH EDITION

Figure 12.1

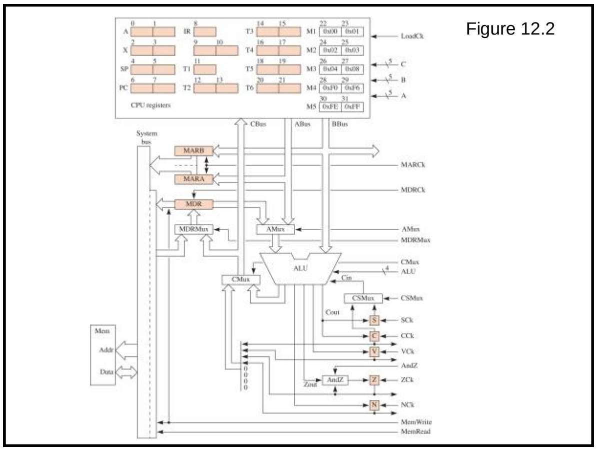
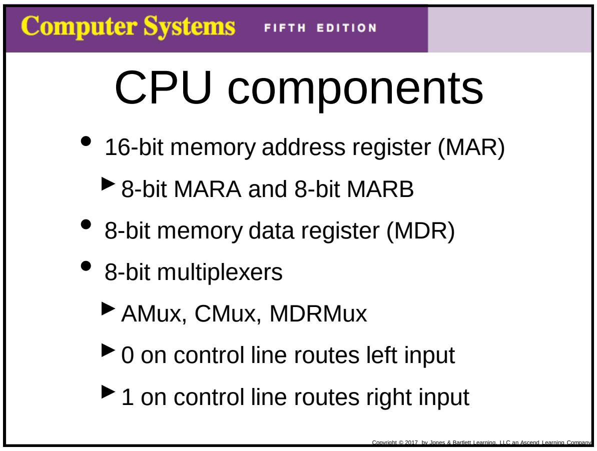
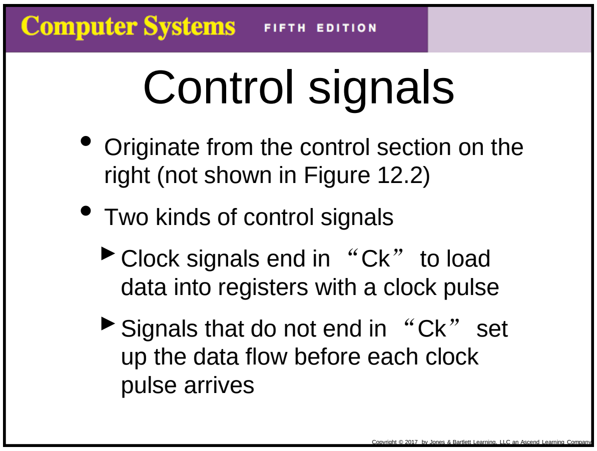


Figure 12.2


Computer Systems FIFTH EDITION

CPU components

- 16-bit memory address register (MAR)
  - 8-bit MARA and 8-bit MARB
- 8-bit memory data register (MDR)
- 8-bit multiplexers
  - AMux, CMux, MDRMux
  - 0 on control line routes left input
  - 1 on control line routes right input


Computer Systems FIFTH EDITION

Control signals

- Originate from the control section on the right (not shown in Figure 12.2)
- Two kinds of control signals
  - Clock signals end in “Ck” to load data into registers with a clock pulse
  - Signals that do not end in “Ck” set up the data flow before each clock pulse arrives

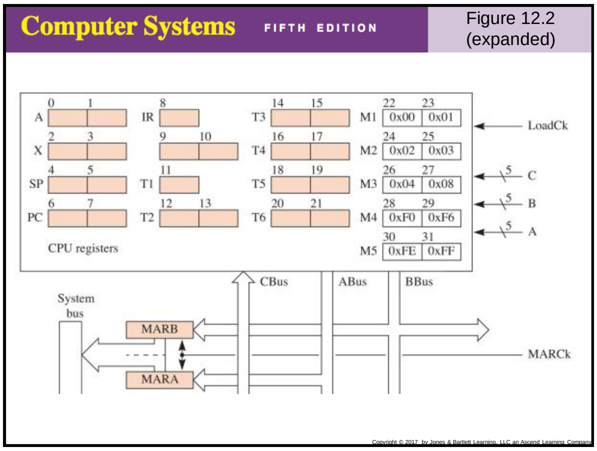
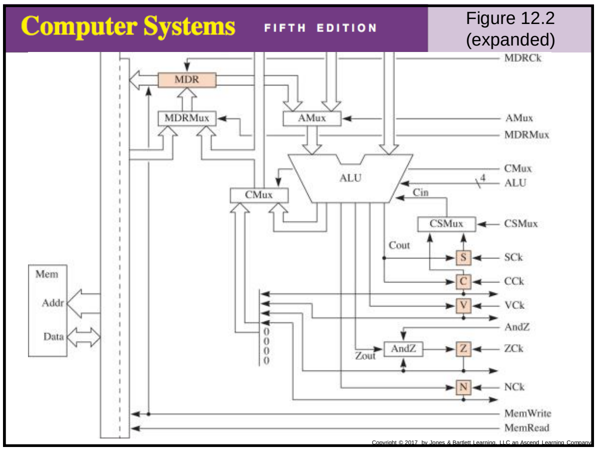

Computer Systems FIFTH EDITION

Figure 12.2 (expanded)

Computer Systems FIFTH EDITION

Figure 12.2 (expanded)



Computer Systems FIFTH EDITION

The status bits NZVC

- Each status bit is a one-bit D flip-flop
- Each status bit is available to the control section
- The status bits can be sent as the low-order nybble to the left input of CMux and from there to the register bank

Computer Systems FIFTH EDITION

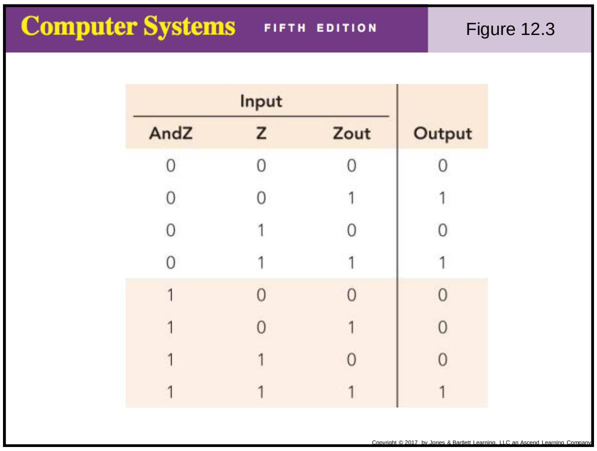
The shadow carry bit S

- Invisible at level ISA3
- Visible at level Mc2
- Used for internal arithmetic operations that should not affect the C bit
- Example: PC ← PC + 1
- Selected by CSMux = 1

Computer Systems FIFTH EDITION

Setting the status bits

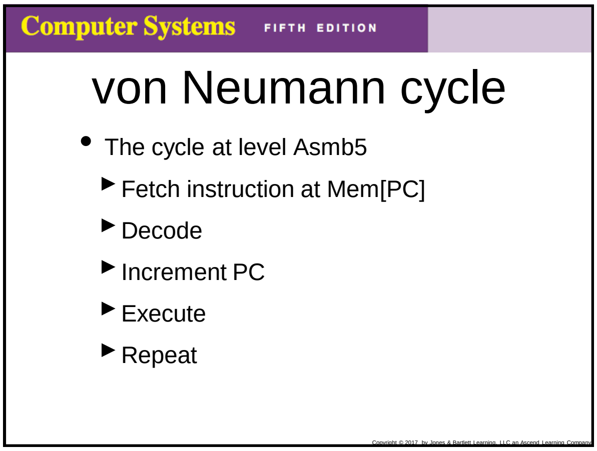
- C can be set directly from ALU Cout, and can be the Cin input to the ALU
- V and N can be set directly from ALU
- Z can be set in one of two ways
  - If AndZ control signal is 0, Z is set directly from ALU Zout
  - If AndZ control signal is 1, Z is set as the AND of ALU Zout and Z



Computer Systems FIFTH EDITION

Figure 12.3

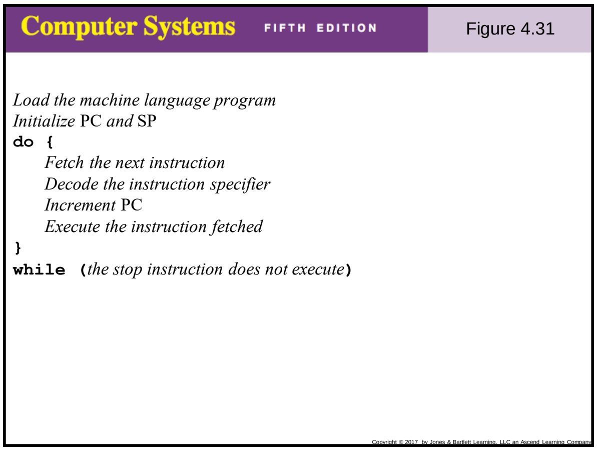
Input			
AndZ	Z	Zout	Output
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1



Computer Systems FIFTH EDITION

von Neumann cycle

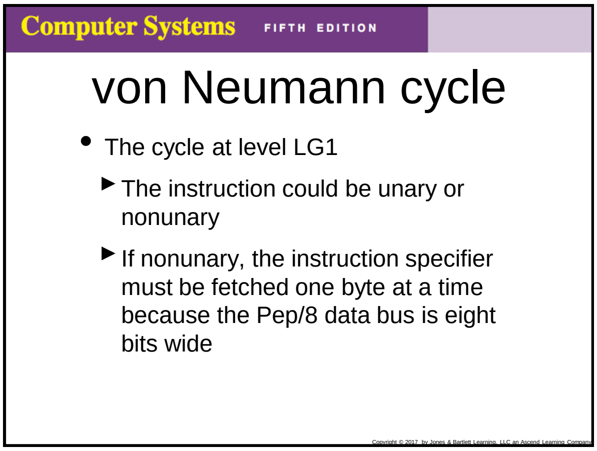
- The cycle at level Asmb5
  - Fetch instruction at Mem[PC]
  - Decode
  - Increment PC
  - Execute
  - Repeat



Computer Systems FIFTH EDITION

Figure 4.31

```
Load the machine language program
Initialize PC and SP
do {
    Fetch the next instruction
    Decode the instruction specifier
    Increment PC
    Execute the instruction fetched
}
while (the stop instruction does not execute)
```



Computer Systems FIFTH EDITION

von Neumann cycle

- The cycle at level LG1
  - The instruction could be unary or nonunary
  - If nonunary, the instruction specifier must be fetched one byte at a time because the Pep/8 data bus is eight bits wide

Computer Systems FIFTH EDITION

Figure 12.4

```
do {
    Fetch the instruction specifier at address in PC
    PC ← PC + 1
    Decode the instruction specifier
    if (the instruction is not unary) {
        Fetch the high-order byte of the operand specifier at address in PC
        PC ← PC + 1
        Fetch the low-order byte of the operand specifier at address in PC
        PC ← PC + 1
    }
    Execute the instruction fetched
}
while ((the stop instruction does not execute) &&
       (the instruction is legal))
```

Computer Systems FIFTH EDITION

Control sequences

- Each line is a clock cycle
- Comma is the parallel separator
- Semicolon is the sequential separator
- Control signals before the semicolon are combinational signals set for the duration of the cycle
- Control signals after the the semicolon are clock pulses at the end of the cycle

Computer Systems FIFTH EDITION

Control signals for the von Neumann cycle

- To fetch from memory
  - Put the address in the MAR (MARA and MARB)
  - Assert MemRead for three consecutive cycles
  - At the end of the third cycle, clock the data from the bus into the MDR

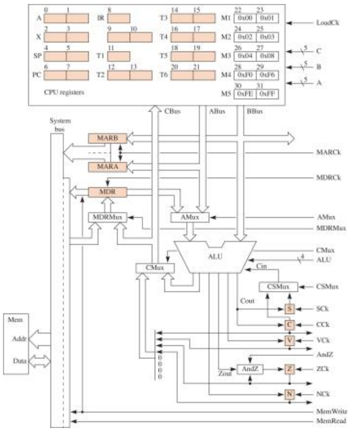
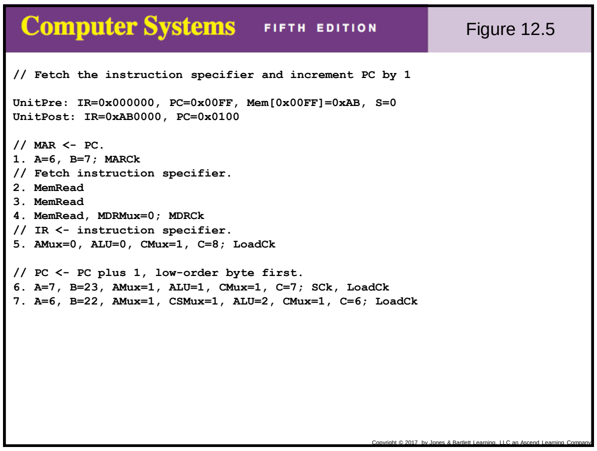


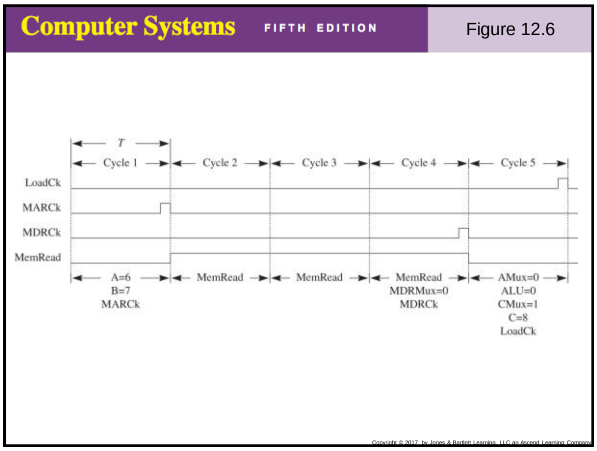
Figure 12.2



Computer Systems FIFTH EDITION

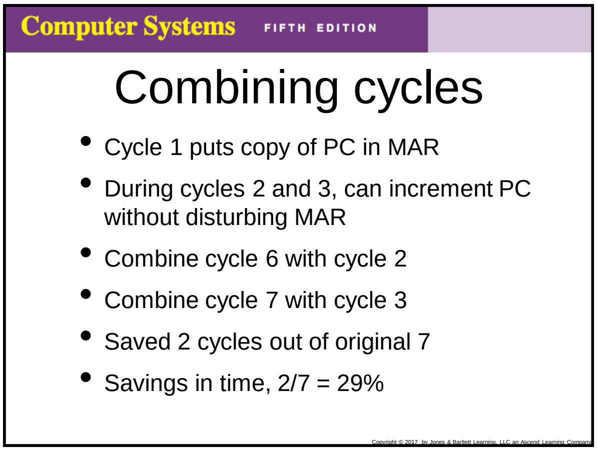
Figure 12.5

```
// Fetch the instruction specifier and increment PC by 1

UnitPre: IR=0x000000, PC=0x00FF, Mem[0x00FF]=0xAB, S=0
UnitPost: IR=0xAB0000, PC=0x0100

// MAR <- PC.
1. A=6, B=7; MARCk
// Fetch instruction specifier.
2. MemRead
3. MemRead
4. MemRead, MDRMux=0; MDRCk
// IR <- instruction specifier.
5. AMux=0, ALU=0, CMux=1, C=8; LoadCk

// PC <- PC plus 1, low-order byte first.
6. A=7, B=23, AMux=1, ALU=1, CMux=1, C=7; SCk, LoadCk
7. A=6, B=22, AMux=1, CSMux=1, ALU=2, CMux=1, C=6; LoadCk
```



Computer Systems FIFTH EDITION

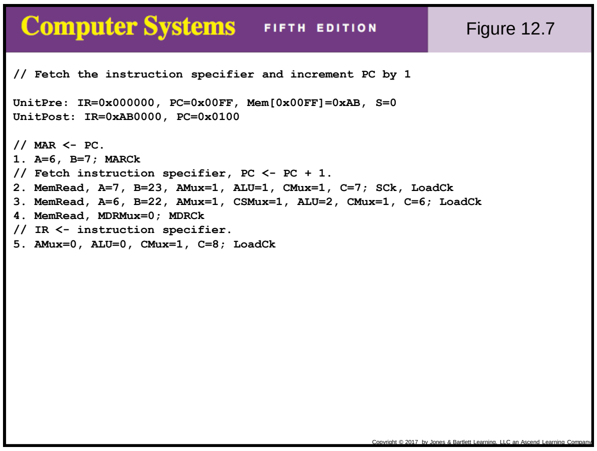
Figure 12.6



Computer Systems FIFTH EDITION

Combining cycles

- Cycle 1 puts copy of PC in MAR
- During cycles 2 and 3, can increment PC without disturbing MAR
- Combine cycle 6 with cycle 2
- Combine cycle 7 with cycle 3
- Saved 2 cycles out of original 7
- Savings in time, 2/7 = 29%



Computer Systems FIFTH EDITION

Figure 12.7

```
// Fetch the instruction specifier and increment PC by 1

UnitPre: IR=0x000000, PC=0x00FF, Mem[0x00FF]=0xAB, S=0
UnitPost: IR=0xAB0000, PC=0x0100

// MAR <- PC.
1. A=6, B=7; MARCk
// Fetch instruction specifier, PC <- PC + 1.
2. MemRead, A=7, B=23, AMux=1, ALU=1, CMux=1, C=7; SCk, LoadCk
3. MemRead, A=6, B=22, AMux=1, CSMux=1, ALU=2, CMux=1, C=6; LoadCk
4. MemRead, MDRMux=0; MDRCk
// IR <- instruction specifier.
5. AMux=0, ALU=0, CMux=1, C=8; LoadCk
```

Computer Systems FIFTH EDITION

Figure 12.8

Addressing mode	Operand
Immediate	OprndSpec
Direct	Mem[OprndSpec]
Indirect	Mem[Mem[OprndSpec]]
Stack-relative	Mem[SP + OprndSpec]
Stack-relative deferred	Mem[Mem[SP + OprndSpec]]
Indexed	Mem[OprndSpec + X]
Stack-indexed	Mem[SP + OprndSpec + X]
Stack-deferred indexed	Mem[Mem[SP + OprndSpec] + X]

Computer Systems FIFTH EDITION

Store byte

- Assume operand specifier has been fetched and contains the address of the operand
- To write byte to memory
  - Put the address in the MAR and the data to to be written in the MDR
  - Assert MemWrite for three consecutive cycles

$$\text{byte Oprnd} \leftarrow r\langle 8..15\rangle$$

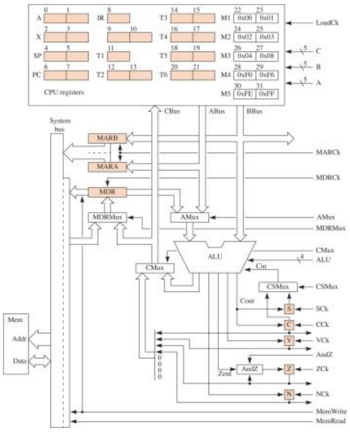


Figure 12.2

Computer Systems FIFTH EDITION

Figure 12.9

```
// STBA there,d
// RTL: byteOprnd <- A<8..15>
// Direct addressing: Oprnd = Mem[OprndSpec]

UnitPre: IR=0xF1000F, A=0x00AB
UnitPost: Mem[0x000F]=0xAB

// MAR <- OprndSpec.
1. A=9, B=10; MARCk
// Initiate write, MBR <- A<low>.
2. MemWrite, A=1, AMux=1, ALU=0, CMux=1, MDRMux=1; MDRCk
3. MemWrite
4. MemWrite
```

Computer Systems FIFTH EDITION

Memory read protocol

- Address in MAR before first MemRead cycle
- Data in MDR on or before third MemRead cycle
- Cannot change MAR on third MemRead cycle

Computer Systems FIFTH EDITION

Memory write protocol

- Address in MAR before first MemWrite cycle
- Data in MDR on or before second MemWrite cycle
- Can put next data in MDR on third MemWrite cycle
- Cannot change MAR on third MemWrite cycle

Computer Systems FIFTH EDITION

Store word

- Assume operand specifier has been fetched and contains the address of the operand
- To write word to memory
  - Write first byte to memory
  - Increment OprndSpec by 1
  - Write second byte to memory

Oprnd ← r

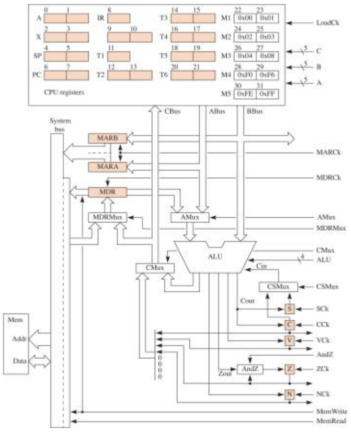
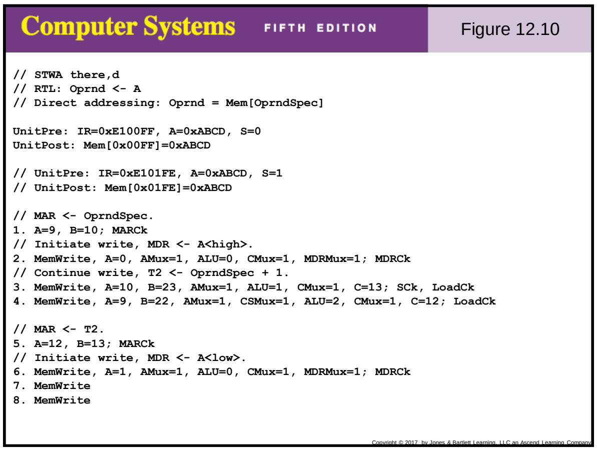


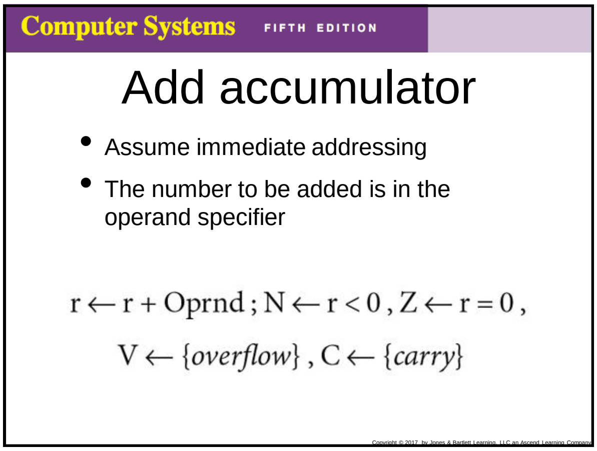
Figure 12.2



Computer Systems FIFTH EDITION

Figure 12.10

```
// STWA there,d
// RTL: Oprnd <- A
// Direct addressing: Oprnd = Mem[OprndSpec]

UnitPre: IR=0xE100FF, A=0xABCD, S=0
UnitPost: Mem[0x00FF]=0xABCD

// UnitPre: IR=0xE101FE, A=0xABCD, S=1
// UnitPost: Mem[0x01FE]=0xABCD

// MAR <- OprndSpec.
1. A=9, B=10; MARCk
// Initiate write, MDR <- A<high>.
2. MemWrite, A=0, AMux=1, ALU=0, CMux=1, MDRMux=1; MDRCk
// Continue write, T2 <- OprndSpec + 1.
3. MemWrite, A=10, B=23, AMux=1, ALU=1, CMux=1, C=13; SCk, LoadCk
4. MemWrite, A=9, B=22, AMux=1, CSMux=1, ALU=2, CMux=1, C=12; LoadCk

// MAR <- T2.
5. A=12, B=13; MARCk
// Initiate write, MDR <- A<low>.
6. MemWrite, A=1, AMux=1, ALU=0, CMux=1, MDRMux=1; MDRCk
7. MemWrite
8. MemWrite
```



Computer Systems FIFTH EDITION

Add accumulator

- Assume immediate addressing
- The number to be added is in the operand specifier

$$r \leftarrow r + \mathrm{Oprnd}\,;\, N \leftarrow r < 0\,,\, Z \leftarrow r = 0\,,$$

$$V \leftarrow \{\mathit{overflow}\}\,,\, C \leftarrow \{\mathit{carry}\}$$

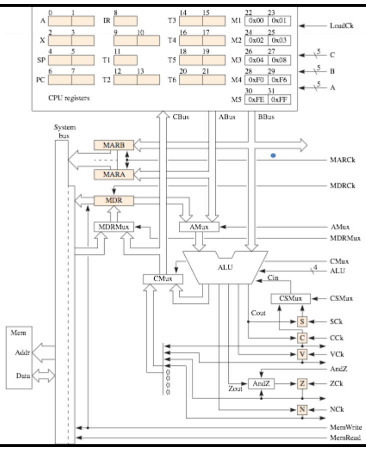
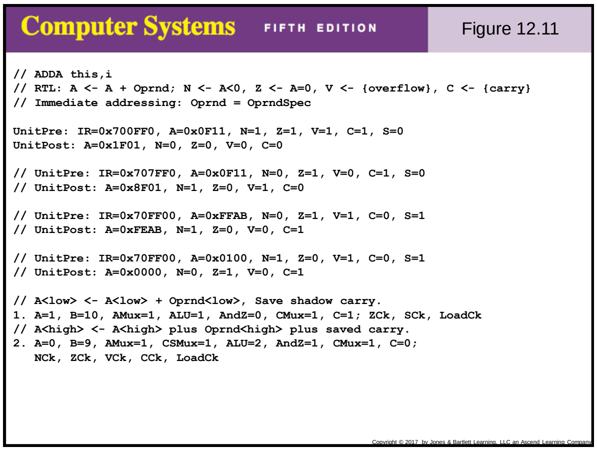


Figure 12.2



Computer Systems FIFTH EDITION

Figure 12.11

```
// ADDA this,i
// RTL: A <- A + Oprnd; N <- A<0, Z <- A=0, V <- {overflow}, C <- {carry}
// Immediate addressing: Oprnd = OprndSpec

UnitPre: IR=0x700FF0, A=0x0F11, N=1, Z=1, V=1, C=1, S=0
UnitPost: A=0x1F01, N=0, Z=0, V=0, C=0

// UnitPre: IR=0x707FF0, A=0x0F11, N=0, Z=1, V=0, C=1, S=0
// UnitPost: A=0x8F01, N=1, Z=0, V=1, C=0

// UnitPre: IR=0x70FF00, A=0xFFAB, N=0, Z=1, V=1, C=0, S=1
// UnitPost: A=0xFEAB, N=1, Z=0, V=0, C=1

// UnitPre: IR=0x70FF00, A=0x0100, N=1, Z=0, V=1, C=0, S=1
// UnitPost: A=0x0000, N=0, Z=1, V=0, C=1

// A<low> <- A<low> + Oprnd<low>, Save shadow carry.
1. A=1, B=10, AMux=1, ALU=1, AndZ=0, CMux=1, C=1; ZCk, SCk, LoadCk
// A<high> <- A<high> plus Oprnd<high> plus saved carry.
2. A=0, B=9, AMux=1, CSMux=1, ALU=2, AndZ=1, CMux=1, C=0;
   NCk, ZCk, VCk, CCk, LoadCk
```

Load index register

- Assume indirect addressing
- The operand specifier contains the address of the address of the operand

$$r \leftarrow \text{Oprnd}\,;\, N \leftarrow r < 0\,,\, Z \leftarrow r = 0$$

Figure 12.13

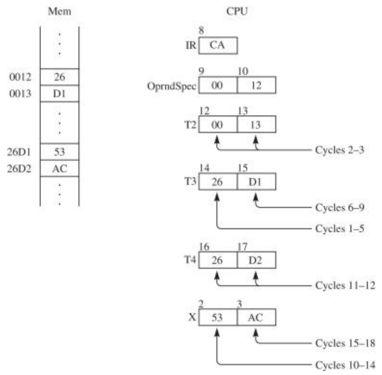


Figure 12.12

```
// Figure 12.12
// LDWX this,n
// RTL: X <- Oprnd; N <- X<0, Z <- X=0
// Indirect addressing: Oprnd = Mem[Mem[OprndSpec]]

UnitPre: IR=0xCA0012, Mem[0x0012]=0x26D1, Mem[0x26D1]=0x53AC
UnitPre: N=1, Z=1, V=0, C=1, S=1
UnitPost: X=0x53AC, N=0, Z=0, V=0, C=1

// UnitPre: IR=0xCA0012, X=0xEEEE, Mem[0x0012]=0x00FF, Mem[0x00FF]=0x0000
// UnitPre: N=1, Z=0, V=1, C=0, S=1
// UnitPost: X=0x0000, N=0, Z=1, V=1, C=0

// T3<high> <- Mem[OprndSpec], T2 <- OprndSpec + 1.
1. A=9, B=10; MARCk
2. MemRead, A=10, B=23, AMux=1, ALU=1, CMux=1, C=13; SCk, LoadCk
3. MemRead, A=9, B=22, AMux=1, CSMux=1, ALU=2, CMux=1, C=12; LoadCk
4. MemRead, MDRMux=0; MDRCk
5. A=12, B=13, AMux=0, ALU=0, CMux=1, C=14; MARCk, LoadCk
```

Figure 12.12
(continued)

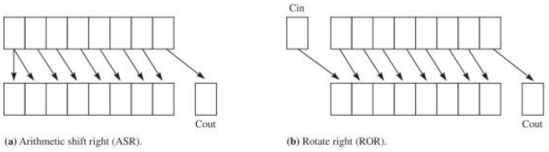
```
// T3<low> <- Mem[T2].
6. MemRead
7. MemRead
8. MemRead, MDRMux=0; MDRCk
9. AMux=0, ALU=0, CMux=1, C=15; LoadCk

// Assert: T3 contains the address of the operand.
// X<high> <- Mem[T3], T4 <- T3 + 1.
10. A=14, B=15; MARCk
11. MemRead, A=15, B=23, AMux=1, ALU=1, CMux=1, C=17; SCk, LoadCk
12. MemRead, A=14, B=22, AMux=1, CSMux=1, ALU=2, CMux=1, C=16; LoadCk
13. MemRead, MDRMux=0; MDRCk
14. A=16, B=17, AMux=0, ALU=0, AndZ=0, CMux=1, C=2; NCk, ZCk, MARCk, LoadCk

// X<low> <- Mem[T4].
15. MemRead
16. MemRead
17. MemRead, MDRMux=0; MDRCk
18. AMux=0, ALU=0, AndZ=1, CMux=1, C=3; ZCk, LoadCk
```

Arithmetic shift right accumulator

- Unary instruction
- No memory read or write

$$C \leftarrow r\langle 15\rangle\,,\ r\langle 1..15\rangle \leftarrow r\langle 0..14\rangle\ ;$$
$$N \leftarrow r < 0\,,\ Z \leftarrow r = 0$$

Figure 10.62



(a) Arithmetic shift right (ASR).

(b) Rotate right (ROR).

Figure 12.14

```
// ASRA
// RTL: C <- A<15>, A<1..15> <- A<0..14>; N <- A<0, Z <- A=0

UnitPre: IR=0x0C0000, A=0xFF01, N=1, Z=1, V=1, C=0, S=0
UnitPost: A=0xFF80, N=1, Z=0, V=1, C=1

// UnitPre: IR=0x0C0000, A=0x7E00, N=1, Z=1, V=0, C=0, S=1
// UnitPost: A=0x3F00, N=0, Z=0, V=0, C=0

// UnitPre: IR=0x0C0000, A=0x0001, N=1, Z=1, V=0, C=0, S=1
// UnitPost: A=0x0000, N=0, Z=1, V=0, C=1

// Arithmetic shift right of high-order byte.
1. A=0, AMux=1, ALU=13, AndZ=0, CMux=1, C=0; NCk, ZCk, SCk, LoadCk
// Rotate right of low-order byte.
2. A=1, AMux=1, CSMux=1, ALU=14, AndZ=1, CMux=1, C=1; ZCk, CCk, LoadCk
```

The Pep/9 control section

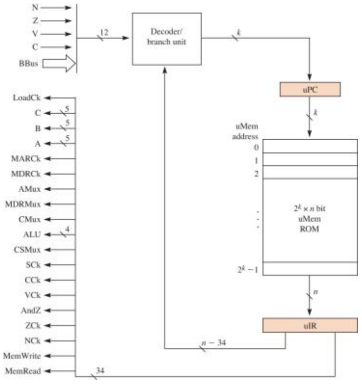
Computer Systems FIFTH EDITION

Level Mc2 for Pep/9

- uMem: Microcode ROM memory
- uPC: Microcode program counter
- uIR: Microcode instruction register
- Micro-von Neumann cycle
  - No increment part of the cycle
  - Each micro-instruction contains the address of the next instruction

Computer Systems FIFTH EDITION

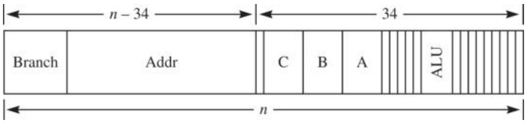
Figure 12.15



Computer Systems FIFTH EDITION

Figure 12.16

Microinstruction format



Computer Systems FIFTH EDITION

Increasing performance

- Increase data bus width
- Insert a cache between CPU and memory
- Increase hardware parallelism with pipelining