CSC 220: Computer Organization

Unit 12
CPU Design & Programming

Prepared by:
Md Saiful Islam, PhD

Updated by:
Isra Al-Turaiki, PhD

Department of Computer Science

College of Computer and Information Sciences

Overview

- Simple Computer Architecture
- Single-Cycle Hardwired Control
  - PC Function
  - Instruction Decoder
  - Example Instruction Execution

Chapter-8

M. Morris Mano, Charles R. Kime and Tom Martin, **Logic and Computer Design Fundamentals**, Global (5th) Edition, Pearson Education Limited, 2016. ISBN: 9781292096124

Introduction

- Systems are:
  - Non-programmable:
    - sequence of fixed operations.
    - execute fixed operations sequenced by inputs and status signals only.
  - Programmable:
    - user inputs a program (loaded into memory).
    - system decodes and executes each instruction in the program.
    - example: the simple computer in this chapter.
- A programmable system uses a sequence of instructions to control its operation.

Introduction

- A typical instruction specifies:
  - Operation to be performed.
  - Operands to use.
  - Where to place the result.
  - Sometimes, which instruction to execute next.
- Instructions are stored in RAM or ROM as a program.
- SC needs to know the address in RAM\ROM of the instruction to be executed.
  - stored in PC (program counter) register.

Introduction

- The PC and associated control logic are part of the *Control Unit (CU)*.
- 3 steps performed by the CU:
  - Fetch instruction from memory into IR (Instruction Register).
  - Decode the instruction.
  - Execute the instruction: sequence of micro-operations.

Some Basic Terminology

- a *program*: a list of instructions
  - specifies the operations to be performed by the processor and,
  - their sequence
- an *instruction* is a collection of bits that instructs the processor to perform a specific operation.
- the collection of all instructions for a processor is called *instruction set*.
- A thorough description of the instruction set for a processor is called *instruction set architecture (ISA).*

Instruction Set Architecture (ISA)

- Any *instruction set architecture* has the following three major components.
  - Storage resources: the resources the user sees available for storing information.
  - Instruction formats: determine the meaning of the bits used to encode each instruction.
  - Instruction specifications: describe each of the distinct instructions that can be executed by a processor.

ISA: Storage Resources

- Resources available for storing information
  - Register file
  - Program counter (PC)
  - Instruction memory (program memory)
  - Data memory

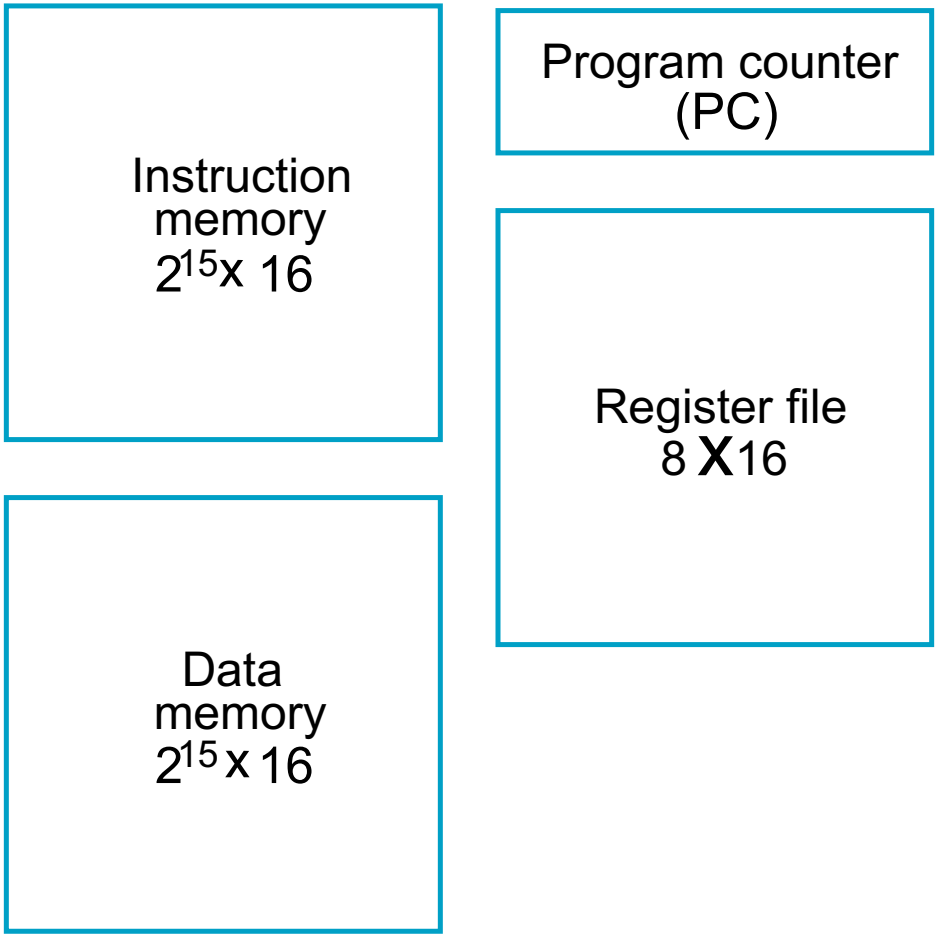
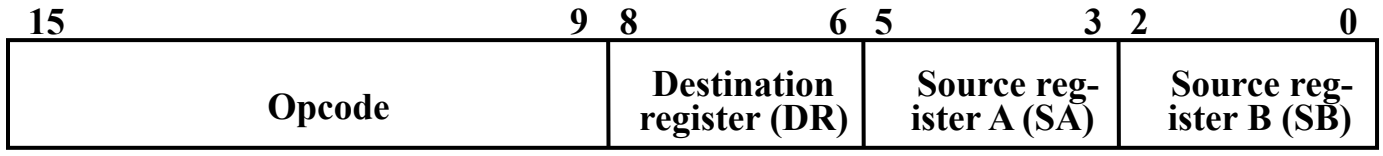


FIGURE 8-13 Storage Resource Diagram for a Simple Computer

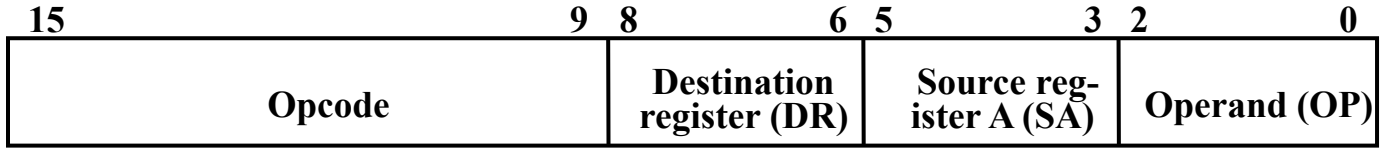
ISA: Instruction Format

- An instruction consists of a bit vector.
- The fields of an instruction are subvectors representing specific functions and having specific binary codes defined.
- An ISA usually contains multiple formats.
- The SC ISA contains the three formats presented on the next slide.

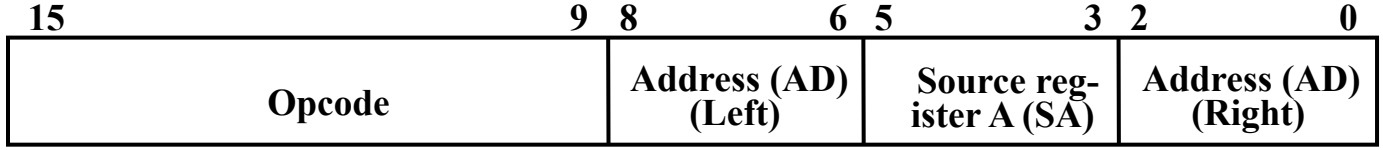
Instruction Format



(a) Register



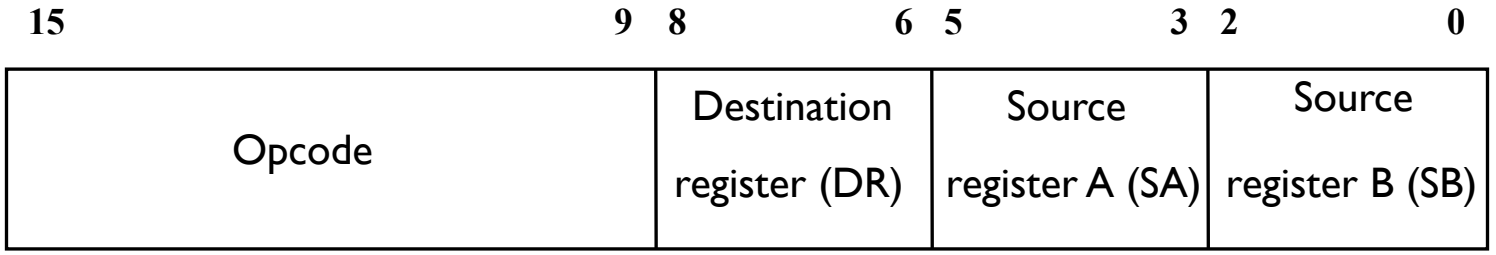
(b) Immediate



(c) Jump and Branch

- The three formats are: Register, Immediate, and Jump and Branch.
- All formats contain an Opcode field in bits 9 through 15.
- The Opcode specifies the operation to be performed.
- More details on each format are provided on the next three slides.

Instruction Format (Register)



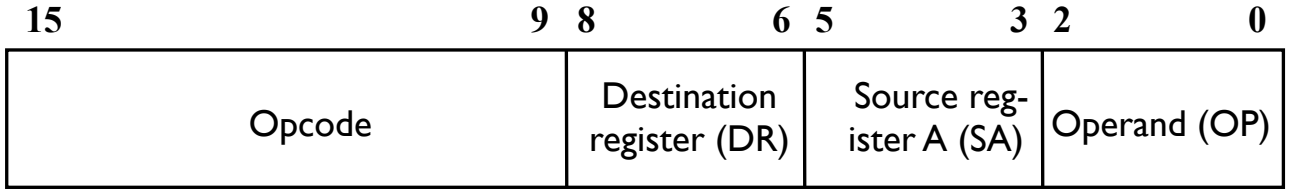
(a) Register

- This format supports instructions represented by:
  - R1 ← R2 + R3
  - R1 ← sl R2
- There are three 3-bit register fields:
  - DR - specifies destination register (R1 in the examples)
  - SA - specifies the A source register (R2 in the first example)
  - SB - specifies the B source register (R3 in the first example and R2 in the second example)

- R1 ← R2 + R3: 0000010 001 010 011
- R1 ← sl R2: 0001110 001 XXX 010

- Why is R2 in the second example SB instead of SA?
  - The source for the shifter in our datapath to be used in implementation is Bus B rather than Bus A.

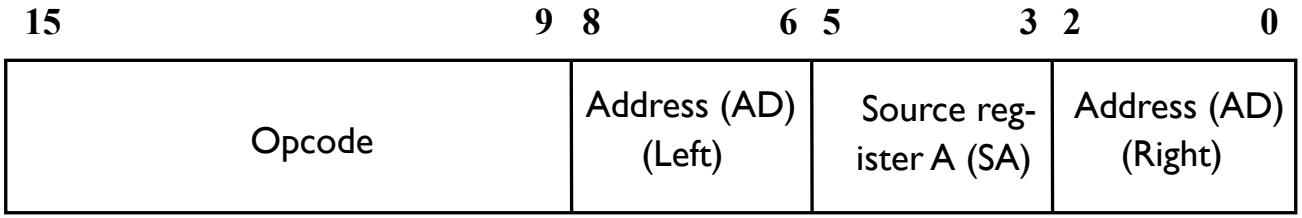
Instruction Format (Immediate)



(b) Immediate

- This format supports instructions described by:
  - R1 ← R2 + 3
- The B Source Register field is replaced by an Operand field OP which specifies a constant.
- The Operand:
  - 3-bit constant.
  - Values from 0 to 7.
- The constant:
  - Zero-fill (on the left of) the Operand to form 16-bit constant.
  - 16-bit representation for values 0 through 7.
- R1 ← R2 + 3: 0000010 001 010 011

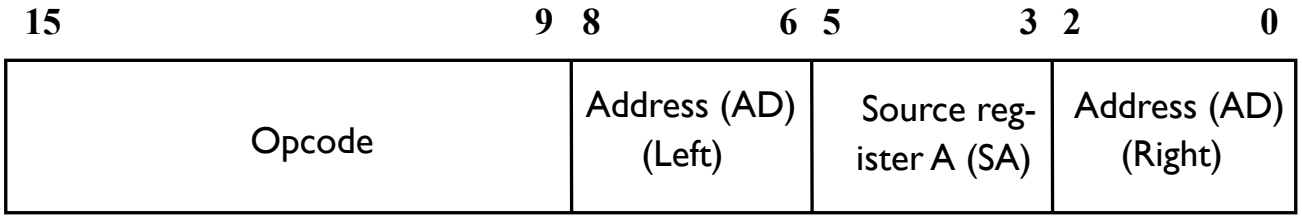
Instruction Format (Jump and Branch)



(c) Jump and Branch

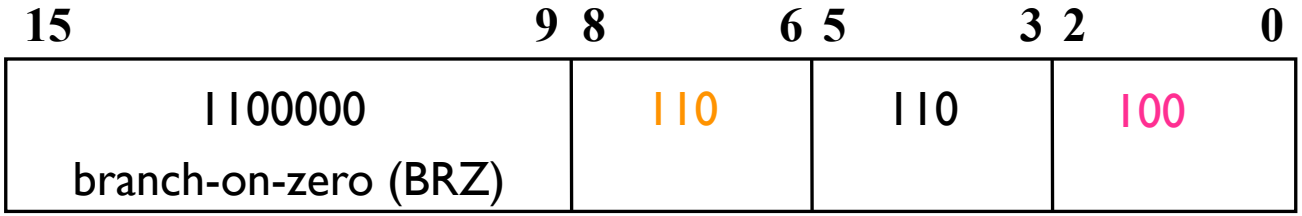
- This instruction supports changes in the sequence of instruction execution
  - by adding an extended, 6 bit, signed 2's complement address offset to the PC value.
  - sign extension is applied to the 6-bit address to form a 16-bit offset before the addition

Instruction Format (Jump and Branch)



(c) Jump and Branch

Example 1: suppose that PC = 45_{10} $(0\ldots0101101)_2$



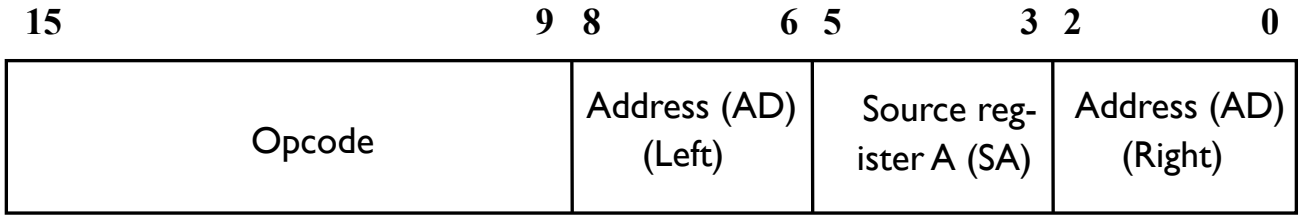
Instruction Description

$$\text{if } (R[SA] = 0)\ PC \leftarrow PC + \text{se AD},$$
$$\text{if } (R[SA] \neq 0)\ PC \leftarrow PC + 1$$

If R6 contains 0 Then
PC = 0…0101101 + (1…110100) =
(45 + (– 12) = 33).

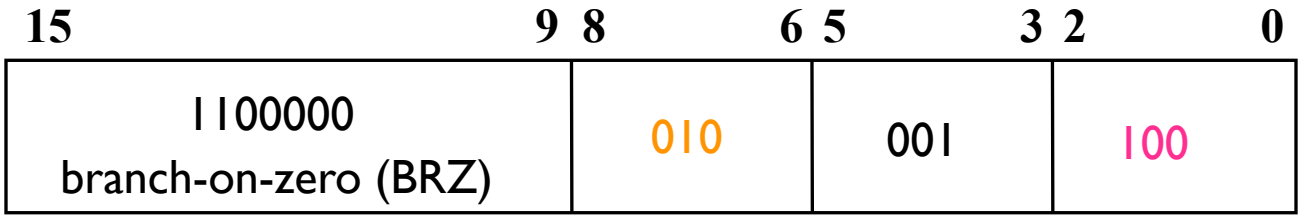
Else
PC = 45+1 = 46

Instruction Format (Jump and Branch)



(c) Jump and Branch

Example 2: suppose that PC = 35_{10}



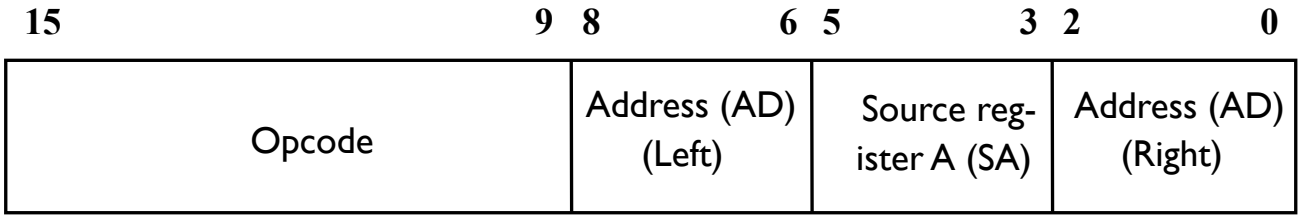
Instruction Description

$$\text{if } (R[SA] = 0)\ PC \leftarrow PC + \text{se AD},$$
$$\text{if } (R[SA] \neq 0)\ PC \leftarrow PC + 1$$

If R1 contains 0 Then
PC = 35 + 20 = 55

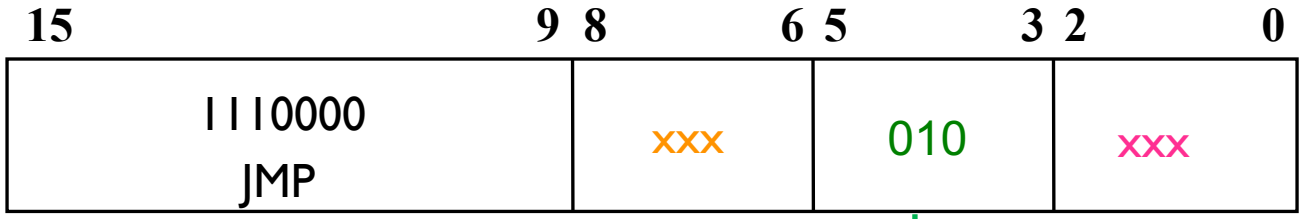
Else
PC = 36

Instruction Format (Jump and Branch)



(c) Jump and Branch

Example 3: R2= 70



jump target.

Instruction Description

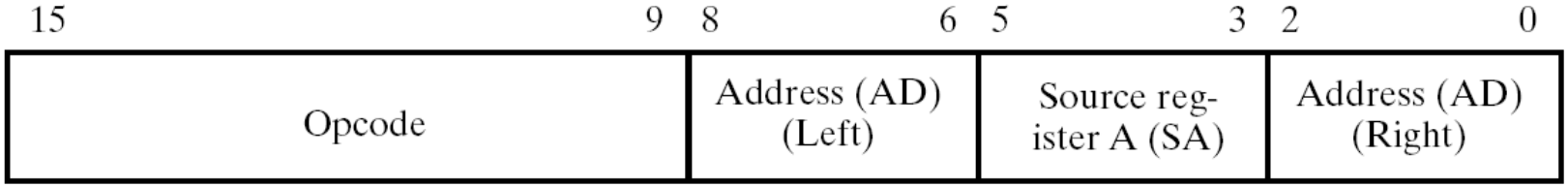
$PC \leftarrow R[SA]$

After the instruction is executed: PC=70

Instruction Format (Jump and Branch)

Summary:

1. Affects the PC.
2. Can load the PC from source SA.
3. Can add the sign-extended 6-bit offset (AD) to the PC.
4. Can be either unconditional, or conditional based on some flag value (i.e. Z, N, C, V).



ISA: Instruction Specifications

- The instruction specifications describe in detail each instruction the system can execute.
- A mnemonic is written (instead of binary opcode) by the programmer to represent the opcode in text.
- This representation is converted to the binary representation by a program called an *assembler*.
  - Example assemblers for Intel x86 processor: NASM, YASM, MASM
- Not every instruction sets every flag
  - Refer to Table 8-8

Instruction Specifications for SC -1

TABLE 8-8
Instruction Specifications for the Simple Computer

Instruction	Opcode	Mnemonic	Format	Description	Status Bits
Move A	0000000	MOVA	RD, RA	R[DR] ← R[SA]*	N, Z
Increment	0000001	INC	RD, RA	R[DR] ← R[SA] + 1*	N, Z
Add	0000010	ADD	RD, RA, RB	R[DR] ← R[SA] + R[SB]*	N, Z
Subtract	0000101	SUB	RD, RA, RB	R[DR] ← R[SA] − R[SB]*	N, Z
Decrement	0000110	DEC	RD, RA	R[DR] ← R[SA] − 1*	N, Z
AND	0001000	AND	RD, RA, RB	R[DR] ← R[SA] ∧ R[SB]*	N, Z
OR	0001001	OR	RD, RA, RB	R[DR] ← R[SA] ∨ R[SB]*	N, Z
Exclusive OR	0001010	XOR	RD, RA, RB	R[DR] ← R[SA] ⊕ R[SB]*	N, Z
NOT	0001011	NOT	RD, RA	R[DR] ← $\overline{\text{R[SA]}}$*	N, Z
Move B	0001100	MOVB	RD, RB	R[DR] ← R[SB]*	
Shift Right	0001101	SHR	RD, RB	R[DR] ← sr R[SB]*	
Shift Left	0001110	SHL	RD, RB	R[DR] ← sl R[SB]*	

Instruction Specifications for SC -2

TABLE 8-8
Instruction Specifications for the Simple Computer

Instruction	Opcode	Mnemonic	Format	Description	Status Bits
Load Immediate	1001100	LDI	RD, OP	R[DR] ← zf OP*	
Add Immediate	1000010	ADI	RD, RA, OP	R[DR] ← R[SA] + zf OP*	N, Z
Load	0010000	LD	RD, RA	R[DR] ← M[SA]*	
Store	0100000	ST	RA, RB	M[SA] ← R[SB]*	
Branch on Zero	1100000	BRZ	RA, AD	if (R[SA] = 0) PC ← PC + se AD, if (R[SA] ≠ 0) PC ← PC + 1	N, Z
Branch on Negative	1100001	BRN	RA, AD	if (R[SA] < 0) PC ← PC + se AD, if (R[SA] ≥ 0) PC ← PC + 1	N, Z
Jump	1110000	JMP	RA	PC ← R[SA]*	

* For all of these instructions, PC ← PC + 1 is also executed to prepare for the next cycle.

ISA: Instruction Specifications

- The specifications provide:
  - The name of the instruction.
  - The instruction's opcode.
  - A shorthand name for the opcode called a mnemonic.
  - A specification for the instruction format.
  - A register transfer description of the instruction.
  - Status bits important to this operation.

ISA: Instruction Specifications

Example:

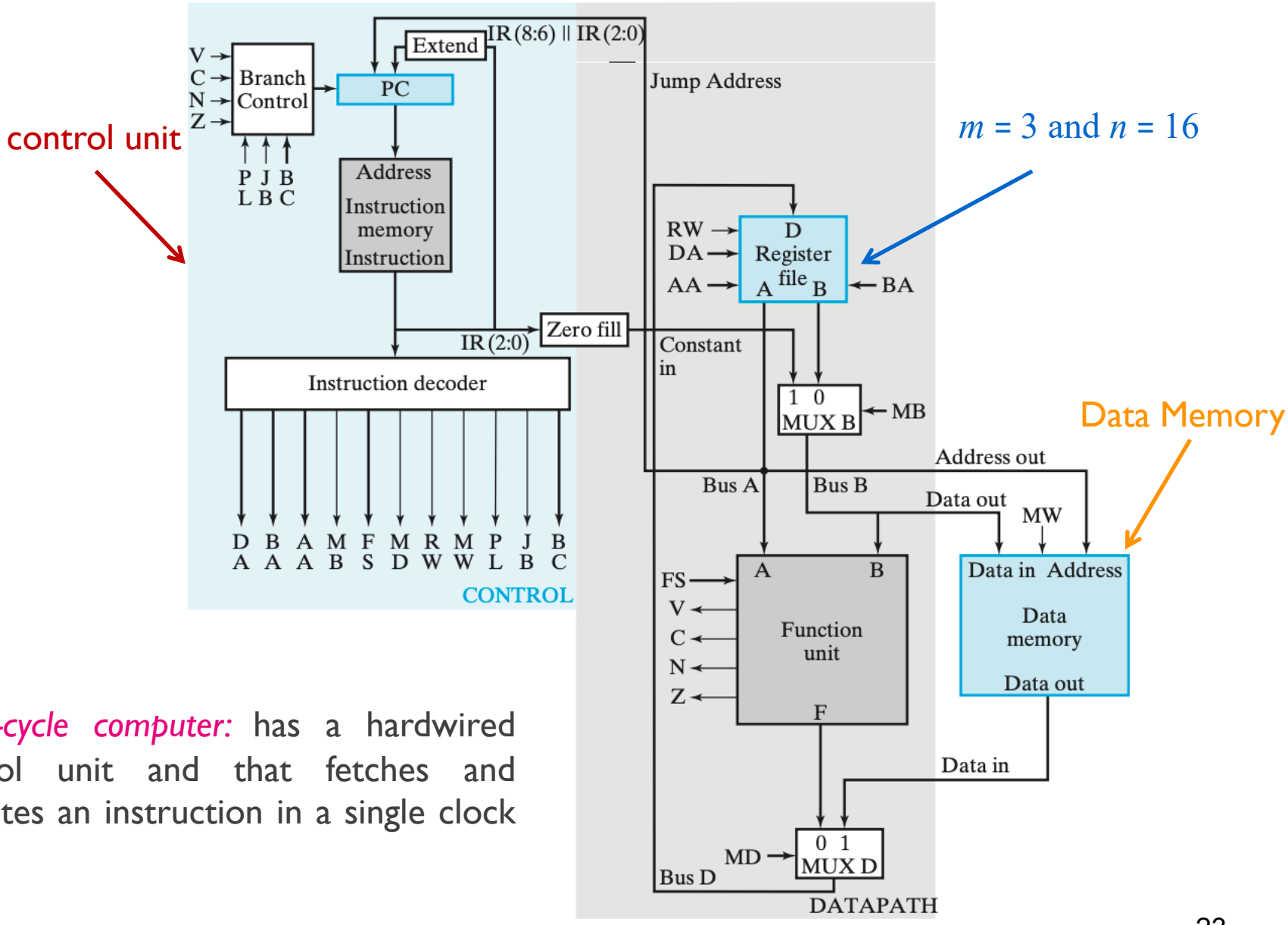
Suppose R4 = 70, R5= 80, and memory location 70=192

M[70]←80

TABLE 8-9
Memory Representation of Instructions and Data

Decimal Address	Memory Contents	Decimal Opcode	Other Fields	Operation
25	0000101 001 010 011	5 (Subtract)	DR:1, SA:2, SB:3	R1 ← R2 – R3
35	0100000 000 100 101	32 (Store)	SA:4, SB:5	M[R4] ← R5
45	1000010 010 111 011	66 (Add Immediate)	DR:2, SA:7, OP:3	R2 ← R7 + 3
55	1100000 101 110 100	96 (Branch on Zero)	AD: 44, SA:6	If R6 = 0, PC ← PC – 20
70	00000000011000000	Data = 192. After execution of instruction in 35, Data = 80.		

Single-cycle hardwired control Unit



Single-cycle computer: has a hardwired control unit and that fetches and executes an instruction in a single clock cycle.

FIGURE 8-15
Block Diagram for a Single-Cycle Computer

Single-cycle hardwired control Unit

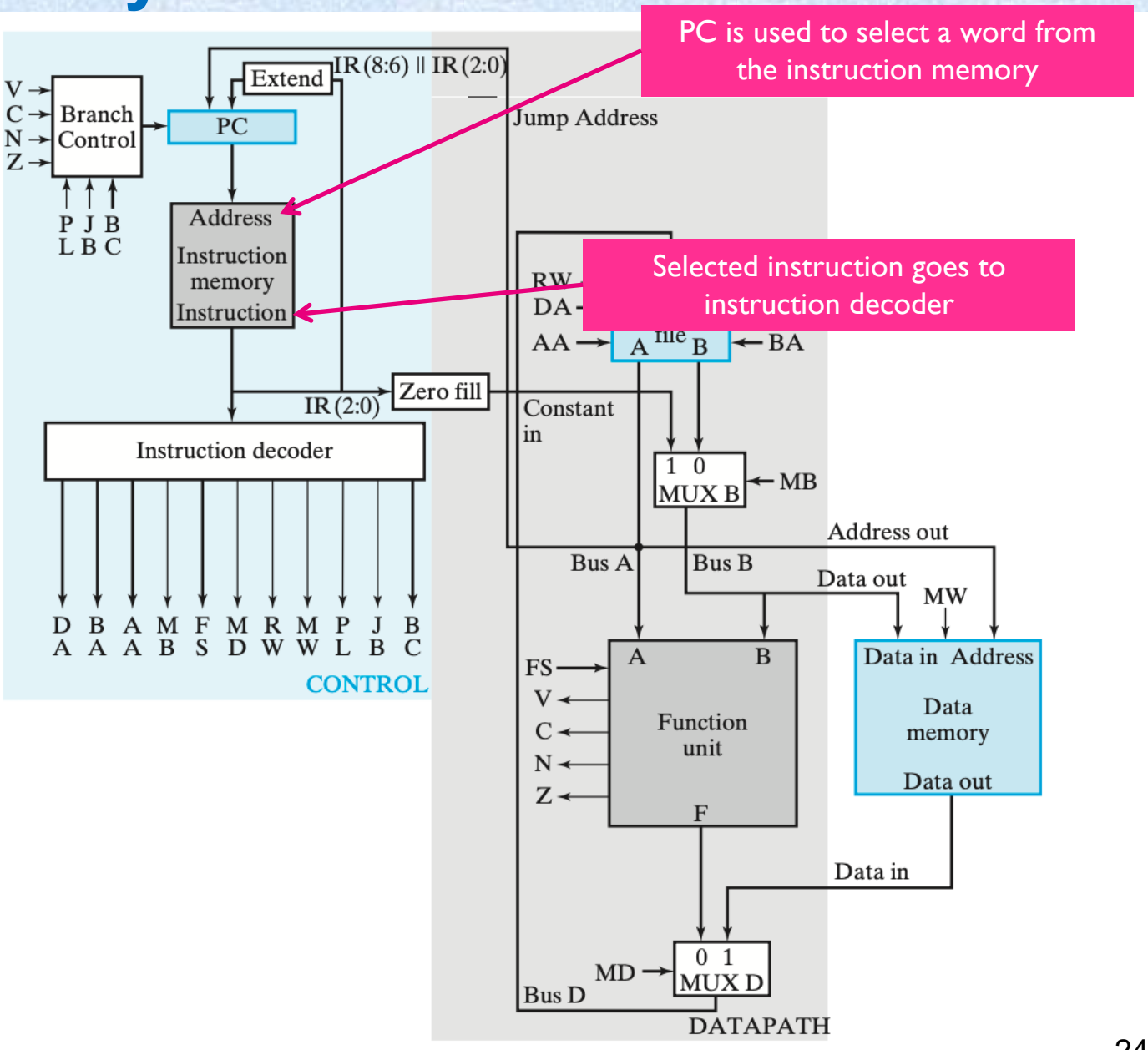


FIGURE 8-15
Block Diagram for a Single-Cycle Computer

Single-cycle hardwired control Unit

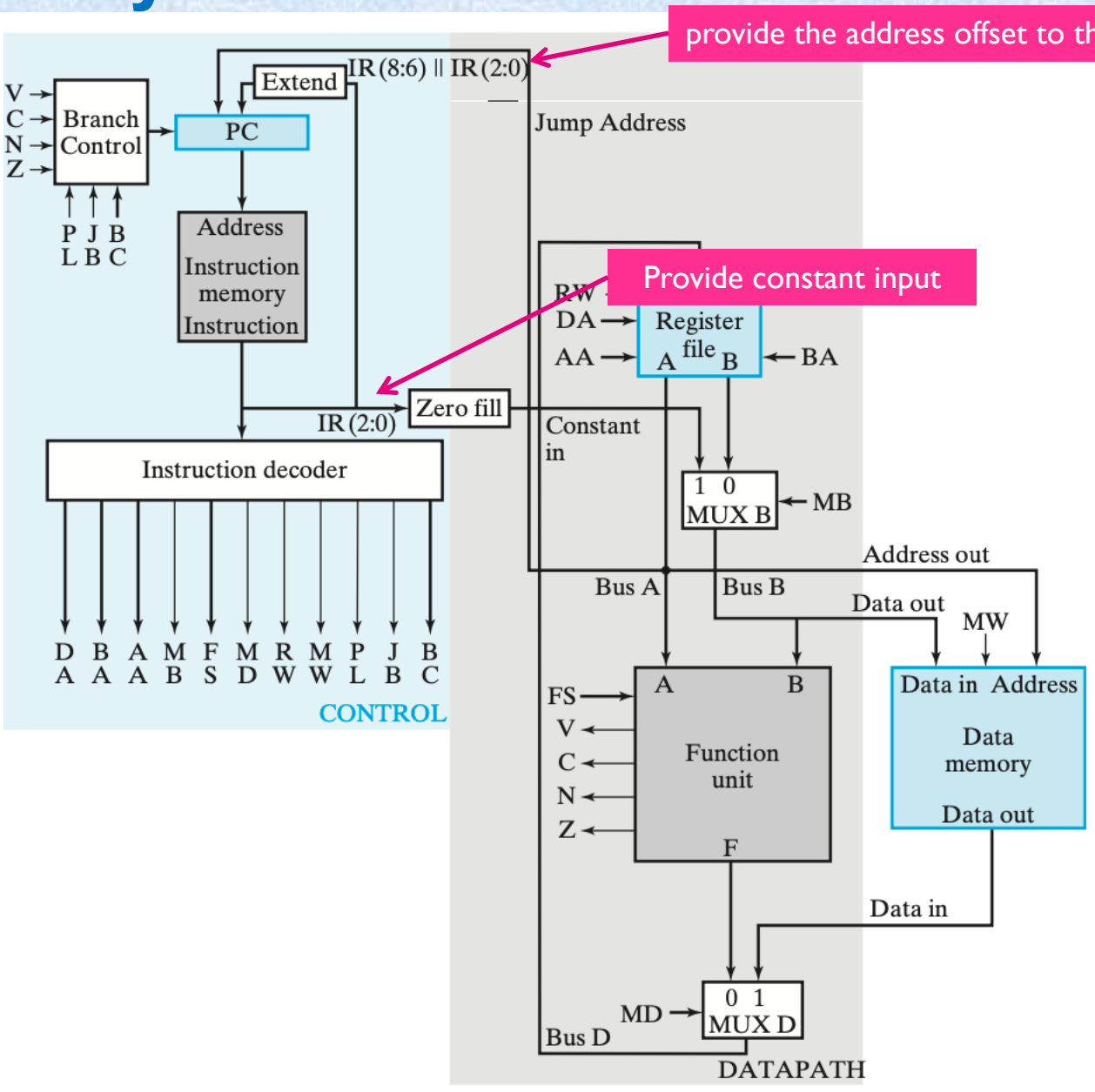
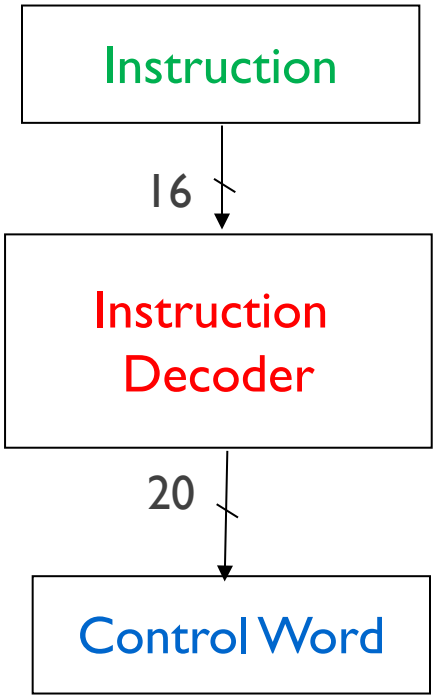


FIGURE 8-15
Block Diagram for a Single-Cycle Computer

Instruction Decoder

- The combinational *instruction decoder* converts the instruction into the signals necessary to control all parts of the computer during the single cycle execution.
- The input is the 16-bit Instruction.
- The outputs are control signals:
  - Register file addresses DA, AA, and BA.
  - Function Unit Select FS.
  - Multiplexer Select Controls MB and MD.
  - Register file and Data Memory Write Controls RW and MW.
  - PC Controls PL, JB, and BC.



PC Function

- Branch Control determines the PC transfers based on five of its inputs defined as follows:
  - N,Z – negative and zero status bits
  - PL – load enable for the PC
  - JB – Jump/Branch select: If JB = 1, Jump, else Branch
  - BC – Branch Condition select: If BC = 1, branch for N = 1, else branch for Z = 1.
- The above is summarized by the following table:

PC Operation	PL	JB	BC
Count Up	0	X	X
Jump	1	1	X
Branch on Negative (else Count Up)	1	0	1
Branch on Zero (else Count Up)	1	0	0

Instruction Decoder

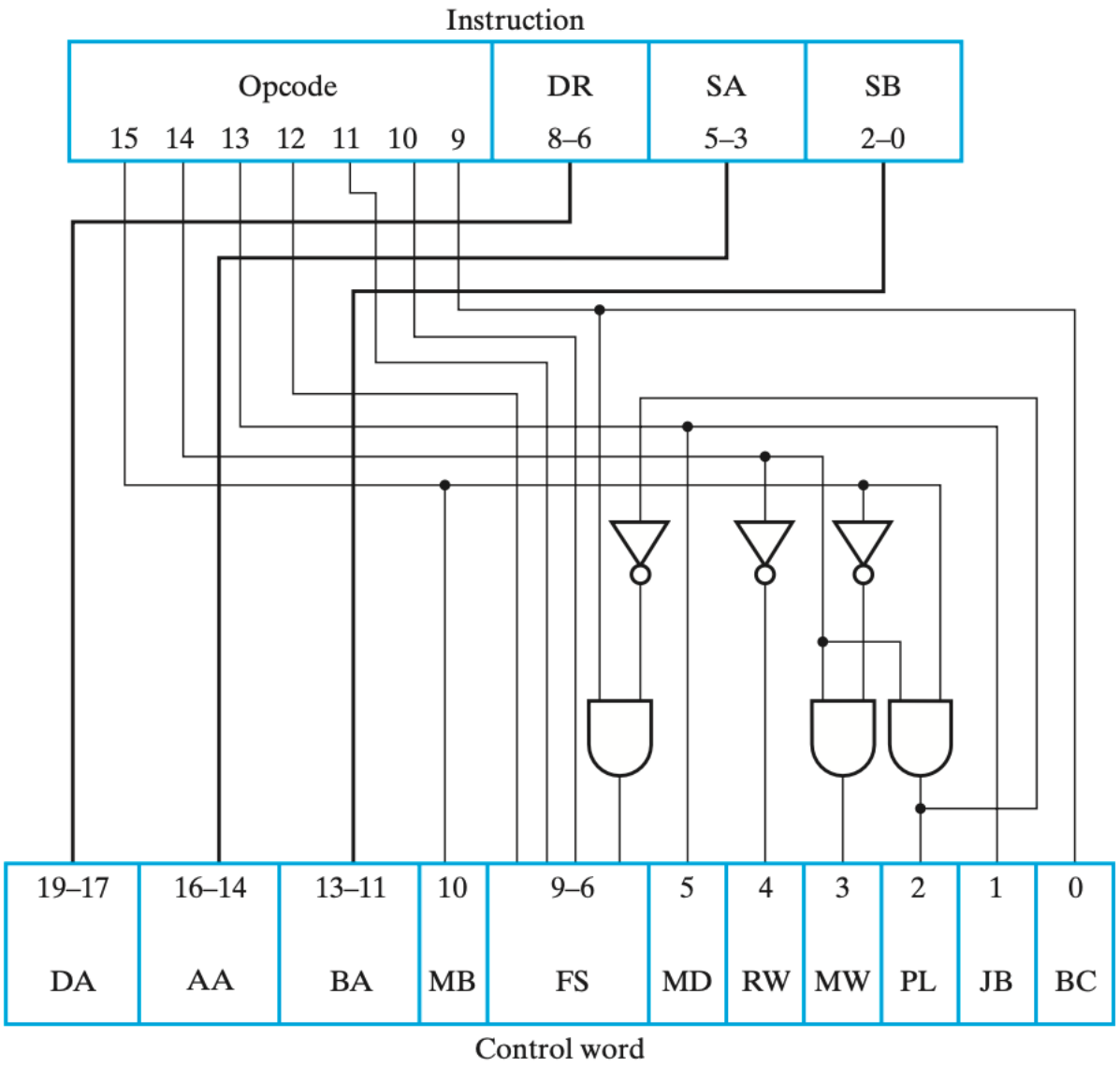
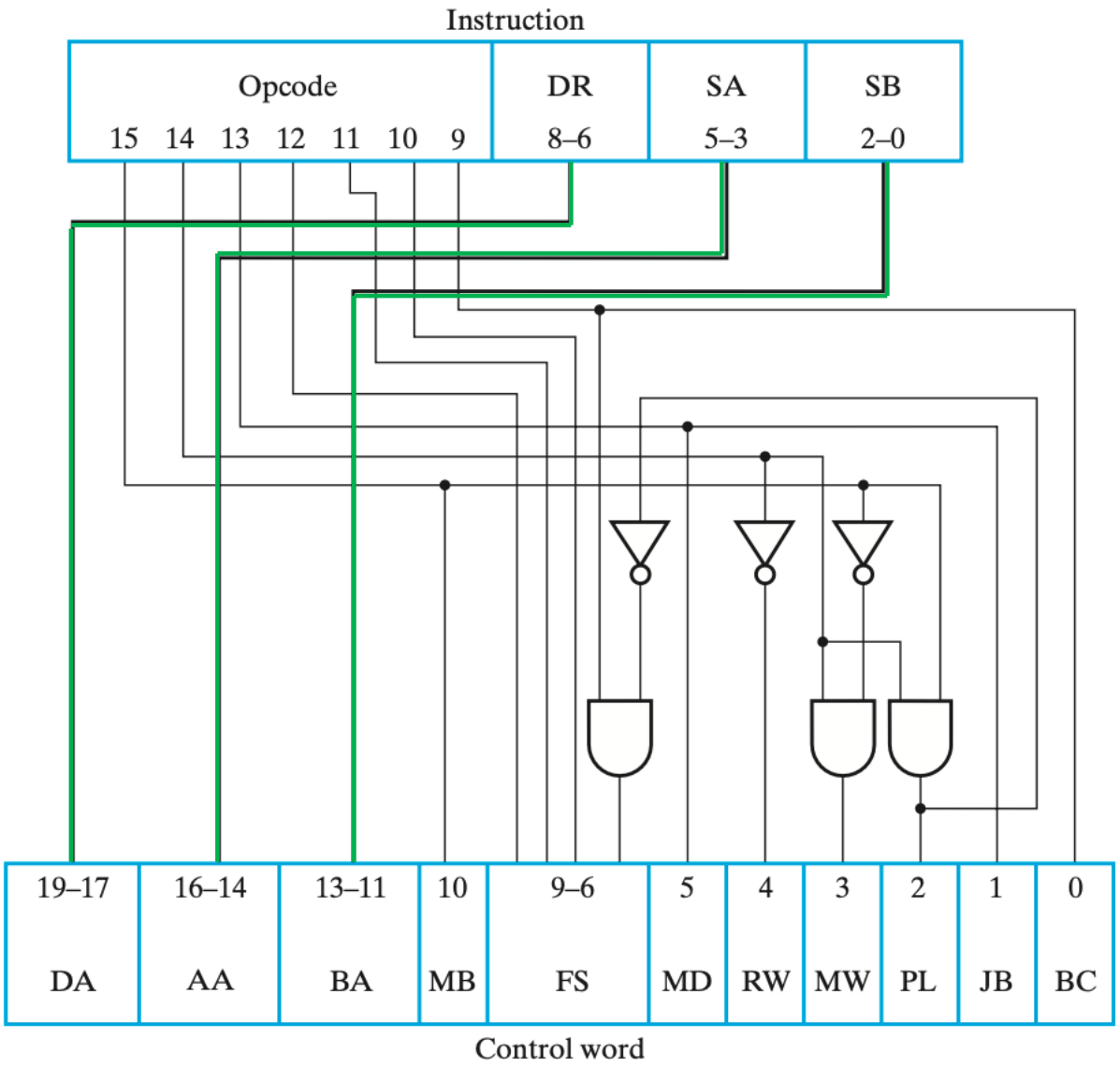


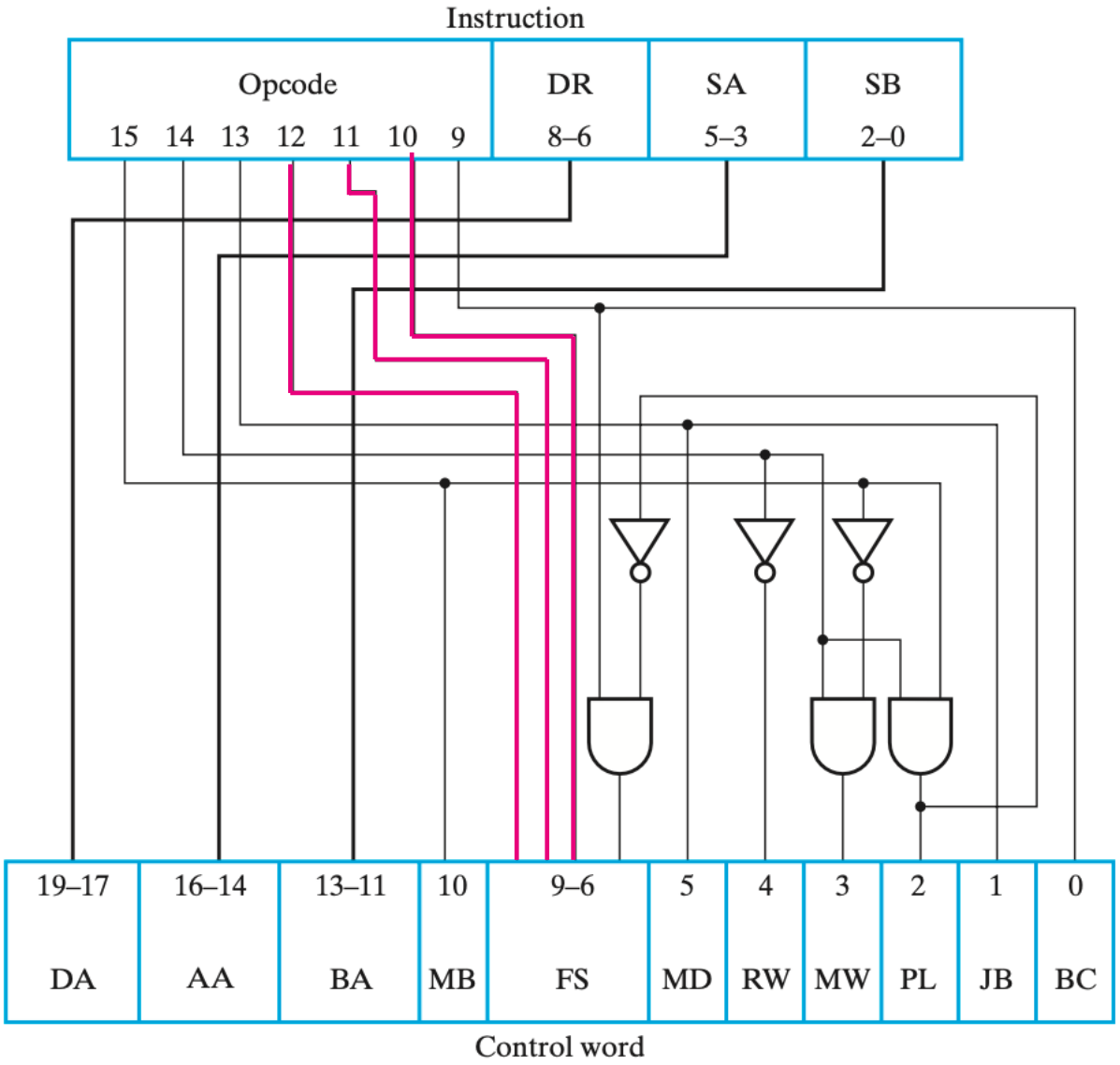
FIGURE 8-16
Diagram of Instruction Decoder

Instruction Decoder



DA, AA, and BA are equal to the instruction fields DR, SA, and SB, respectively.

Instruction Decoder



FS(7:9) = I(10:12)

FIGURE 8-16
Diagram of Instruction Decoder

Instruction Decoder

TABLE 8-10
Truth Table for Instruction Decoder Logic

	Instruction Bits				Control-Word Bits						
Instruction Function Type	**15**	**14**	**13**	**9**	**MB**	**MD**	**RW**	**MW**	**PL**	**JB**	**BC**
Function-unit operations using registers	0	0	0	X	0	0	1	0	0	X	X
Memory read	0	0	1	X	0	1	1	0	0	X	X
Memory write	0	1	0	X	0	X	0	1	0	X	X
Function-unit operations using register and constant	1	0	0	X	1	0	1	0	0	X	X
Conditional branch on zero (Z)	1	1	0	0	X	X	0	0	1	0	0
Conditional branch on negative (N)	1	1	0	1	X	X	0	0	1	0	1
Unconditional jump	1	1	1	X	X	X	0	0	1	1	X

- The remaining control-word fields: MB, MD, RW, and MW
  - MB = I(15), MD = I(13)
  - RW = I(14)′, MW = I(14).I(15)′
- There are two added bits for the control of the PC: PL and JB
  - PL = I(14).I(15)
  - JB = I(13), BC = I(9)

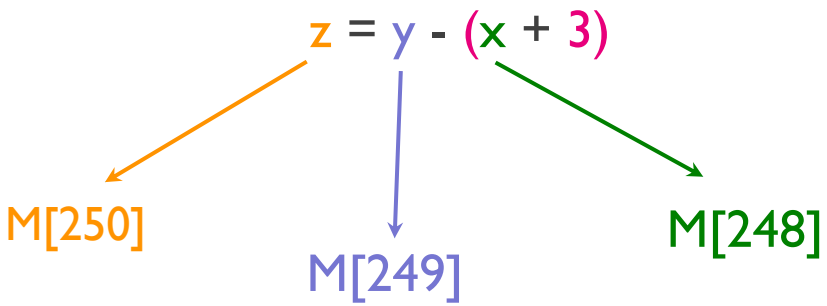
Assembly Language Programming

Example Program:

- Write an assembly language program to evaluate the equation z = y - (x + 3)
- Assume that:
  - x is located at the address 248
  - y is located at the address 249
  - z is located at the address 250
  - R3 = 248

address	Data memory
…	…
248	2
249	83
250	
…	…
…	…

Solution:



Assembly Language Programming

Example Program:

- Write an assembly language program to evaluate the equation z = y - (x + 3)
- Assume that:
  - x is located at the address 248
  - y is located at the address 249.
  - z is located at the address 250
  - R3 = 248

address	Data memory
…	…
248	2
249	83
250	
…	…
…	…

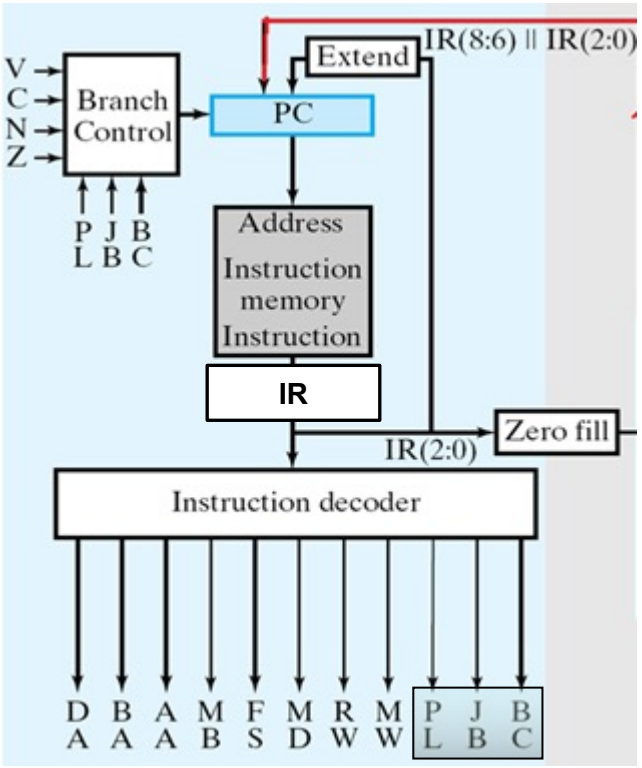
Solution:

```
LD      R1, R3      Load R1 with contents of location 248 in memory (R1 = 2)
ADI     R1, R1, 3   Add 3 to R1 (R1=5)
INC     R3, R3      Increment the contents of R3 (R3= 249)
LD      R2, R3      Load R2 with contents of location 249 in memory (R2 = 83)
SUB     R2, R2, R1  Subtract contents of R1 from contents of R2 (R2 = 78)
INC     R3, R3      Increment the contents of R3 (R3 = 250)
ST      R3, R2      Store R2 in memory location 250 (M[250] = 78)
```

Control Unit Design

Single-cycle hardwired control Unit

- the PC is updated on each clock cycle. Each instruction is completed in a single cycle.
- The PC is used to select a word from the instruction memory:
  - load the instruction to Instruction Register (IR)
  - which is driven to the instruction decoder
- The instruction decoder then provides:
  - the control word to the datapath to activate the desired functionality,
  - determines how the PC is updated.



Control Unit

Programming and CPUs

- Programs written in a high-level language like C++ must be compiled to produce an executable program.
- The result is a CPU-specific machine language program. This can be loaded into memory and executed by the processor.
- CSC 220 focuses on stuff below the dotted blue line, but machine language serves as the interface between hardware and software.
- Machine language instructions are sequences of bits in a specific order.