جامعة الملك سعود
King Saud University

CSC 220: Computer Organization

Unit 12
CPU Design & Programming

Prepared by:
Md Saiful Islam, PhD

Department of Computer Science

College of Computer and Information Sciences

Overview

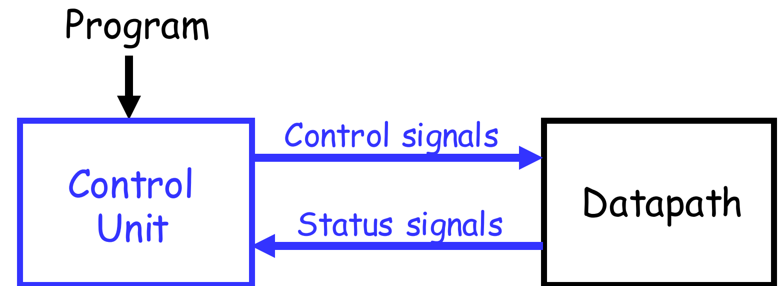
- Overview of CPU Design
- Simple Computer Architecture
- Control Unit Design
    - Machine languages
    - Assembly languages

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

An overview of CPU design

We can divide the design of our CPU into three parts:



- The datapath does all of the actual data processing (Unit-11).
- Instruction Set Architecture: An instruction set is the programmer's interface to CPU.
- A control unit uses the programmer's instructions to tell the datapath what to do.
  - In real computers, the datapath actions are determined by the program that's loaded and running
  - It converts program instructions into control words for the datapath, including **signals WR, DA, AA, BA, MB, FS, MW, MD**.
  - It executes program instructions in the correct sequence.
  - It generates the "constant" input for the datapath.
- The datapath also sends information back to the control unit. For instance, the ALU status bits **V, C, N, Z** can be inspected by branch instructions to alter a program's control flow.

Instruction Set Architecture (ISA)

- A computer is controlled by a series of ***instructions***. Instructions are binary words that are used to determine both the datapath processing and the control behavior.
- A ***program*** is a collection of instructions that are used to accomplish some task(s).
- A ***program counter*** (PC) is used to keep track of the address of the next instruction in memory. The PC has counting logic, as well as parallel load and other logic to permit changes in the instruction sequence.
- Changes in the instruction sequence can be either *conditional* or *unconditional*.
- Instructions are *executed* by activating the necessary microoperations to perform the specified task.

Instruction Set Architecture (ISA) …

- The instruction set architecture (ISA): a comprehensive description of the instructions a computer can execute.
- The ISA has three major components
  - Storage Resources
  - Instruction Formats
  - Instruction Specification

ISA: Storage Resources ..

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

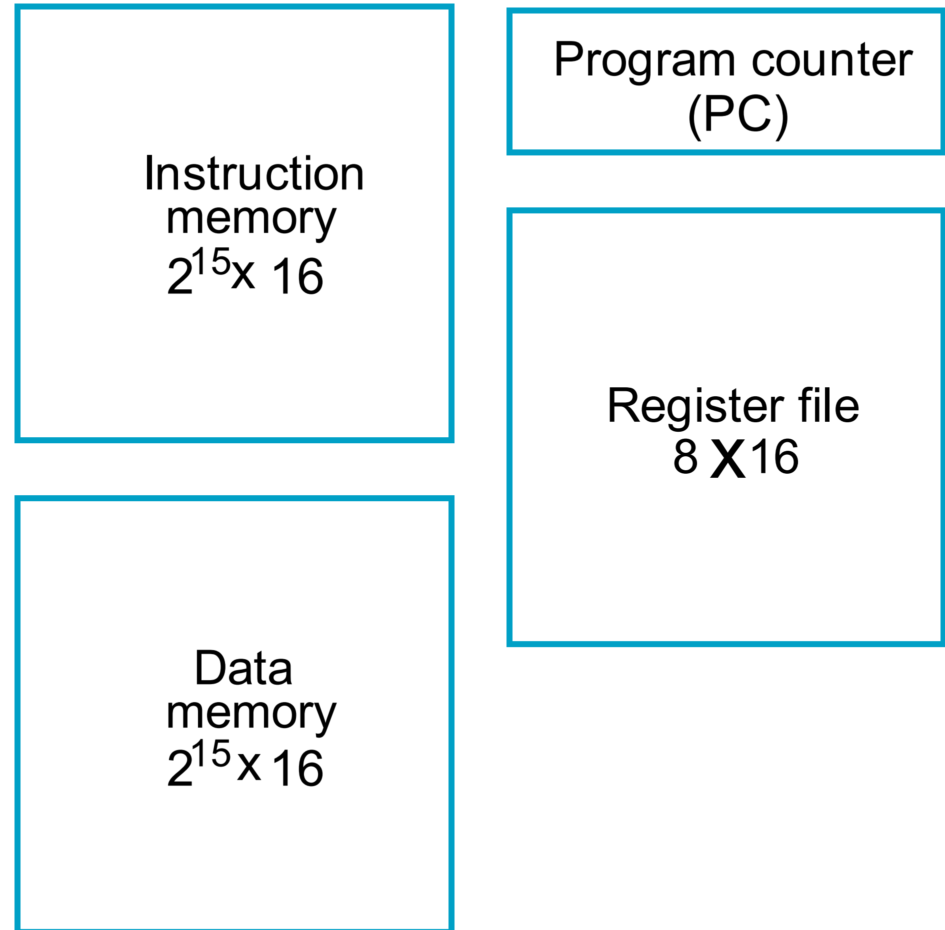
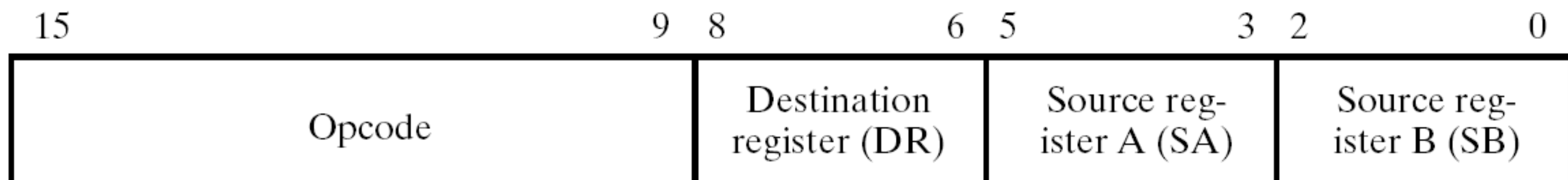


FIGURE 8-13 Storage Resource Diagram for a Simple Computer

ISA: Instruction Formats

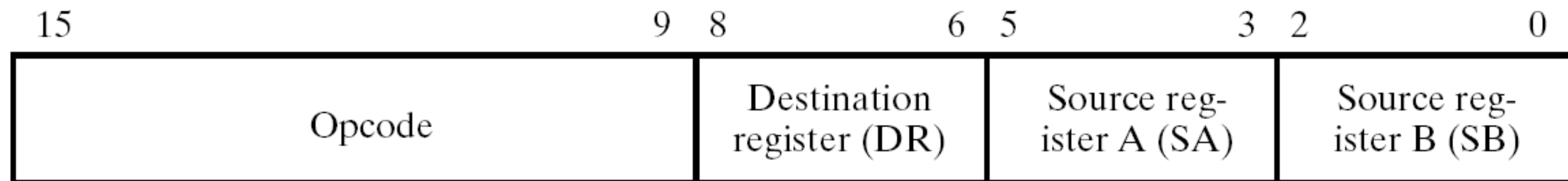
- Instructions are divided into bit groupings called *fields*. Each field will contain a specific part of the instruction.
  - Operation code (opcode)
    - The opcode field determines what the actual operation will be.
    - An n-bit opcode can specify up to 2^n different operations.
  - Operand(s)
    - Registers
    - Addresses of Operands
    - Constant data
  - Operands may be specified implicitly as well.



Instruction Formats -1 …

Register: The source(s) and destination of an instruction can all be registers.

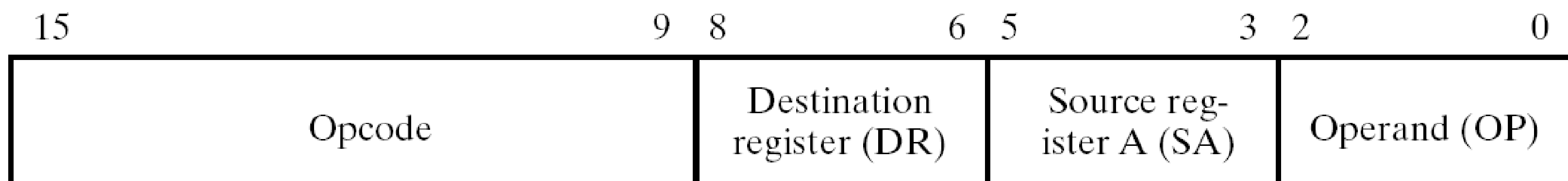
- The source registers are read and the new information is written to the destination register.
- All registers are explicitly identified (usually).



Instruction Formats – 2 …

Immediate operands: data constants are contained within the instruction.

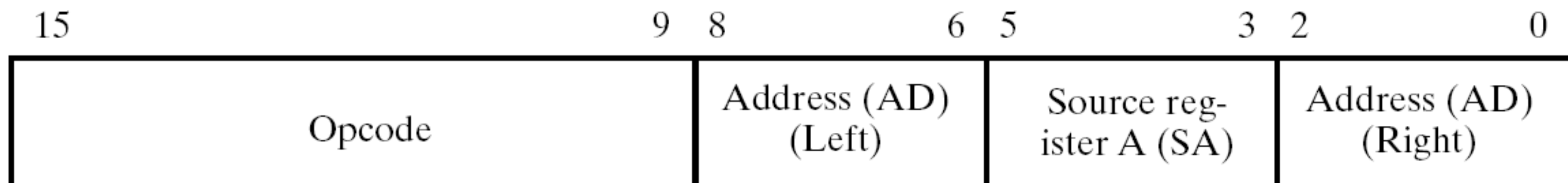
- Note that only three bits of data can be specified here, so
- It must be extended to the register length by zero-fill or sign-extension.
- Clever encodings enable wider immediates
  - RR format, RI format, B format



Instruction Formats – 3 …

Jump and Branch:

- Changes in program flow are caused by jump or branch instructions
- **Affecting only the PC**.
  1. Can load the PC from source SA.
  2. Can add the sign-extended 6-bit offset (AD) to the PC.
  3. 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 by the programmer to represent the opcode in text.
- ***assembler*** generates the actual binary instruction
  - **To make things simpler, people typically use assembly language**
  - **Example assemblers for Intel x86 processor: NASM, YASM, MASM**
- Not every instruction sets every flag
  - Refer to Table 8-8

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, and
  - A listing of the status bits that are meaningful during an instruction's execution (not used in the architectures defined in this chapter)

ISA: Instruction Specifications …

TABLE 8-8
Instruction Specifications for the Simple Computer

Instruction	Opcode	Mne-monic	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] ← $\overline{R[SA]}$*	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]*	

- Here we have arithmetic, logic and shift **instruction sets** correspond to **micro-operations** discussed in Unit 8-11. The **opcode** (7 bit) and **FS** (4 bit) have one-to-one correspondence – first three bits of an opcode are 000.
- RTL instruction (in description) is written in a slightly different way. For example, R[DR] <- R[SA] is a RTL instruction where DR and SA are destination and source register addresses respectively.

ISA: Instruction Specifications …

TABLE 8-8
Instruction Specifications for the Simple Computer

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

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

- These are additional instructions having no corresponding **micro-operations** – for them first three bits have **non-zero values.**
- For instruction LDI, RTL is R[DR] <- zf OP where **zf** means zero filled and OP is a **3 bit constant** (that is used for the simple computer).
- In description of BRZ and BRN, **se** mean sign extended. This is done to make it 16 bit because the RAM has $2^{16} \times 16$ configuration (refer to slide 8, figure 8-13).

Machine Language …

- Table 8-9 shows Instructions and data, in binary, are placed in memory. Here, we assume that we have a program (in machine language) in the RAM and we show instructions at locations 25, 35, 45, and 55 (as examples).
- Corresponding assembly codes for four instructions are as follows.

25: SUB R1, R2, R3
35: ST (R4), R5 [suppose R4=70, R5=80]
45: ADI R2, R7, 3
55: BRZ R6, AD [suppose AD(left)=101, AD(right)=100, se AD = 1111111111101100 = -20]

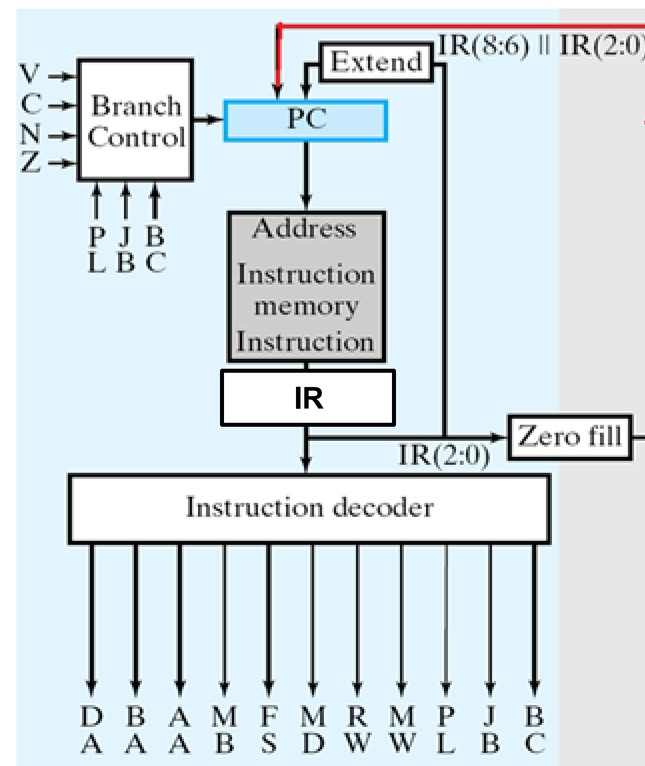
TABLE 8-9
Memory Representation of Instructions and Data

Decimal Address	Memory Contents	Decimal Opcode	Other Fields	Operation
	Opcode DR SA SB			
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 (Immediate operand)	R2 ← R7 + 3
	15 9 8 6 5 3 2 0 — Opcode / Address (AD) (Left) / Source register A (SA) / Address (AD) (Right) — Jump and Branch			
55	1100000 101 110 100 (101100)	96 (Branch on Zero)	AD: 44, SA:6	If R6 = 0, PC ← PC – 20
70	0000000011000000 (before) 0000000001010000 (After)	Data = 192. After execution of instruction in 35, Data = 80.		

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.
- Note that the computer uses programmable control, but we will use a hardwired control unit to actually implement the programmable control.



Control Unit

Control Unit …

The instruction decoder maps the instruction word to a control word.

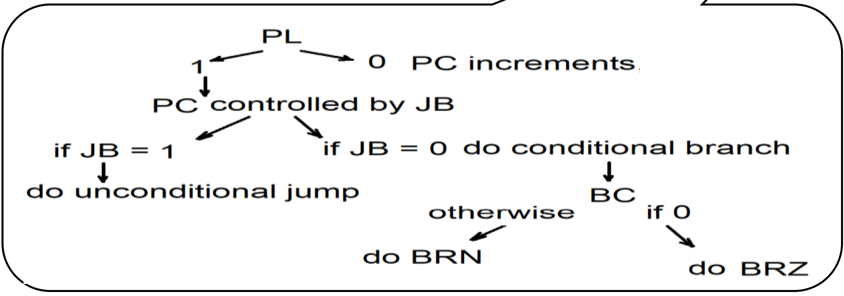
The concatenated offset field is sign-extended for PC-relative addressing.

The constant field is zero-filled and made available to the datapath

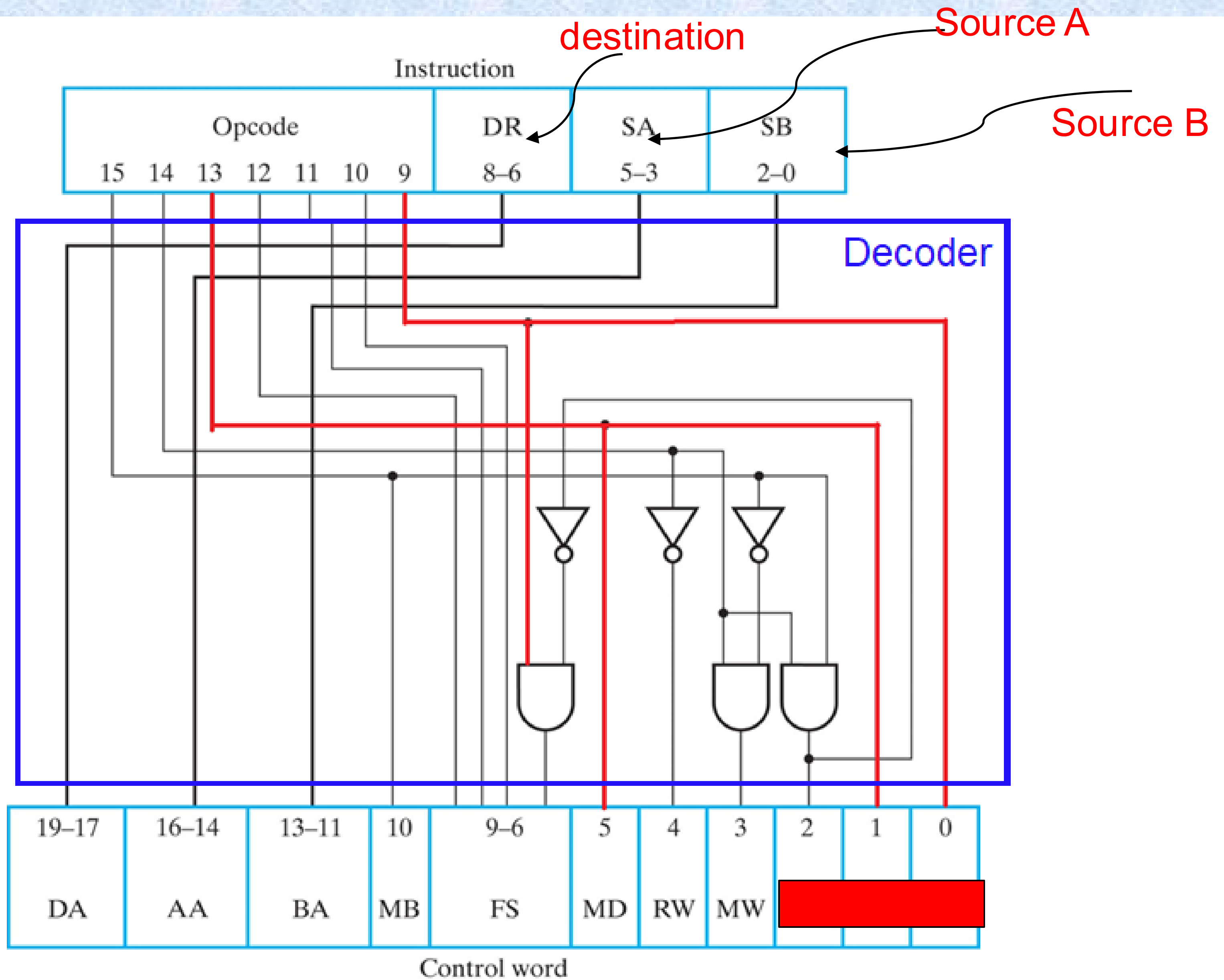
Control Unit

V C N Z
Branch Control
PL JB BC
Extend
IR(8:6) || IR(2:0)
PC
Address
Instruction memory
Instruction
IR
IR(2:0)
Zero fill
Instruction decoder
DA BA AA MB FS MD RW MW
CONTROL

RW DA AA
D Register file
Constant in
MUX B
MB
Address out
Bus A
Bus B
Data out
MW
FS V C N Z
A B
Function unit
F
Data in Address
Data memory
Data out
Data in
MD
0 1 MUX D
Bus D
DATAPATH



Instruction Decoder …



Copyright ©2016 Pearson Education, All Rights Reserved

Instruction Decoder …

- Control-word fields DA, AA, and BA are equal to the instruction fields DR, SA, and SB, respectively
- Control field BC for selection of the branch condition status bits is taken directly from the last bit of Opcode.
- The remaining control-word fields: MB, MD, RW, and MW
- There are two added bits for the control of the PC: PL and JB

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

Assembly Language Programming

Example Program:

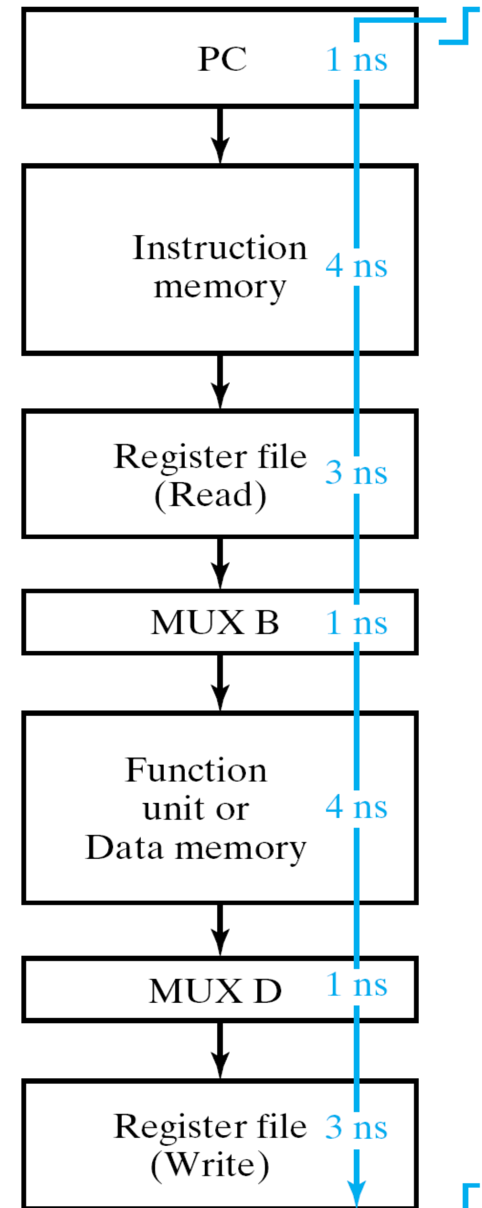
Assume that variable x is located at the address 248 in data memory containing 2, and variable y is located at the address 249 contains 83. Write an assembly language program to evaluate the equation $z = y - (x + 3)$ where variable z is located at the address 250.

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

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

Single-Cycle Computer Issues …

- Complex operations
  - Only combinational logic can be used in data transformation – no sequential logic
    - E.g. no multiplier
- Unified memory
  - If program and data are in one memory, how can you simultaneously access the same memory for an instruction and a data operand?
- Worst-case delay



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.