Computer Architecture

A Generic CPU - 5

Dr. Masood Mir

Week 8

The Dathpath with Register File and Function Unit

- A typical datapath has more than four registers. (32 or more registers are common).
- The construction of a bus system with a large number of registers requires different techniques. A set of registers having common microoperations performed on them may be organized into a register file. The typical register file is a special type of fast memory that permits one or more words to be read and one or more words to be written, all simultaneously.
- Functionally, a simple register file has memory-like nature. Thus, the A select, B select, and Destination select inputs in the figure become three addresses.
- In Figure 8-10, the A address accesses a word to be read onto A data, the B address accesses a second word to be read onto B data, and the D address accesses a word to be written into from D data. All of these accesses occur in the same clock cycle.

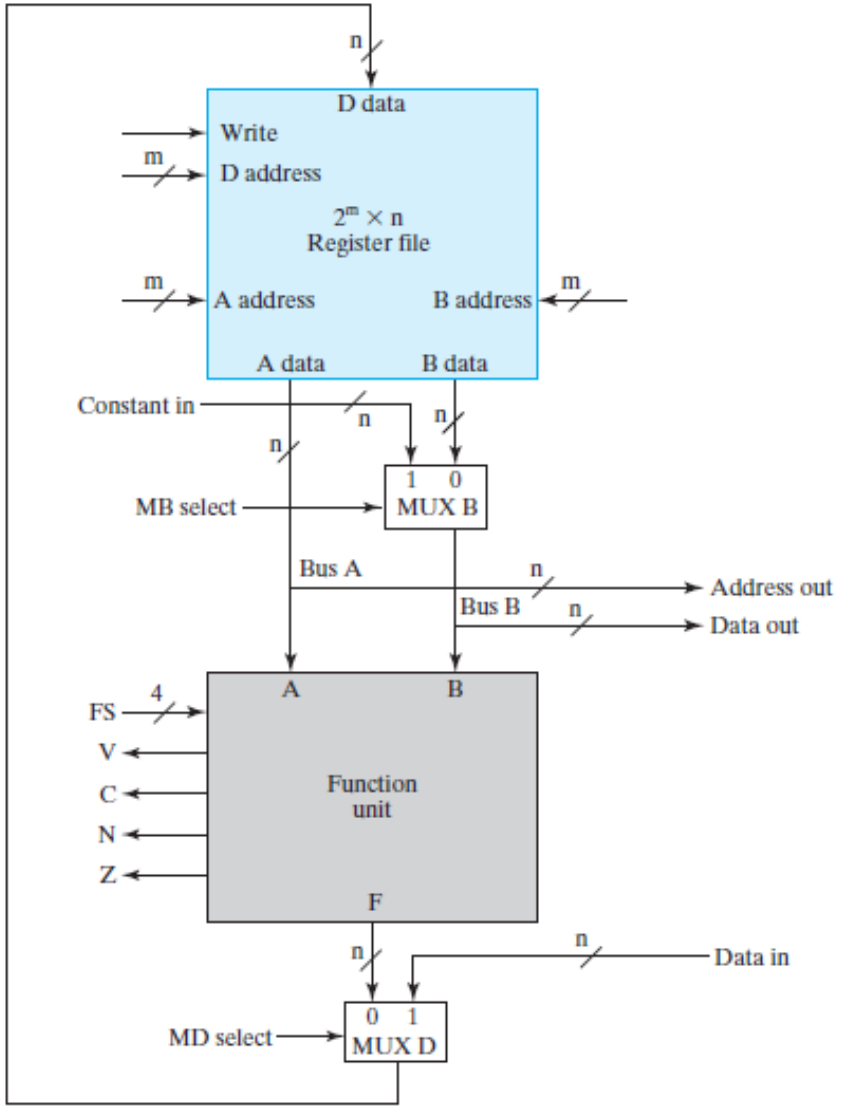


FIGURE 8-10
Block Diagram of Datapath Using the Register File and Function Unit

The Dathpath with Register File and Function Unit

- A Write input corresponding to the Load Enable signal is also provided.
- When at 1, the Write signal permits registers to be loaded during the current clock cycle, and when at 0, prevents register loading.
- The size of the register file is 2^m * n, where m is the number of register address bits and n is the number of bits per register.
- Since the ALU and the shifter are shared processing units with outputs that are selected by MUX F, it is convenient to group the two units and the MUX together to form a shared function unit.
- The inputs to the function unit are from Bus A and Bus B, and the output of the function unit goes to MUX D.
- The function unit also has the four status bits V, C, N, and Z as added outputs.

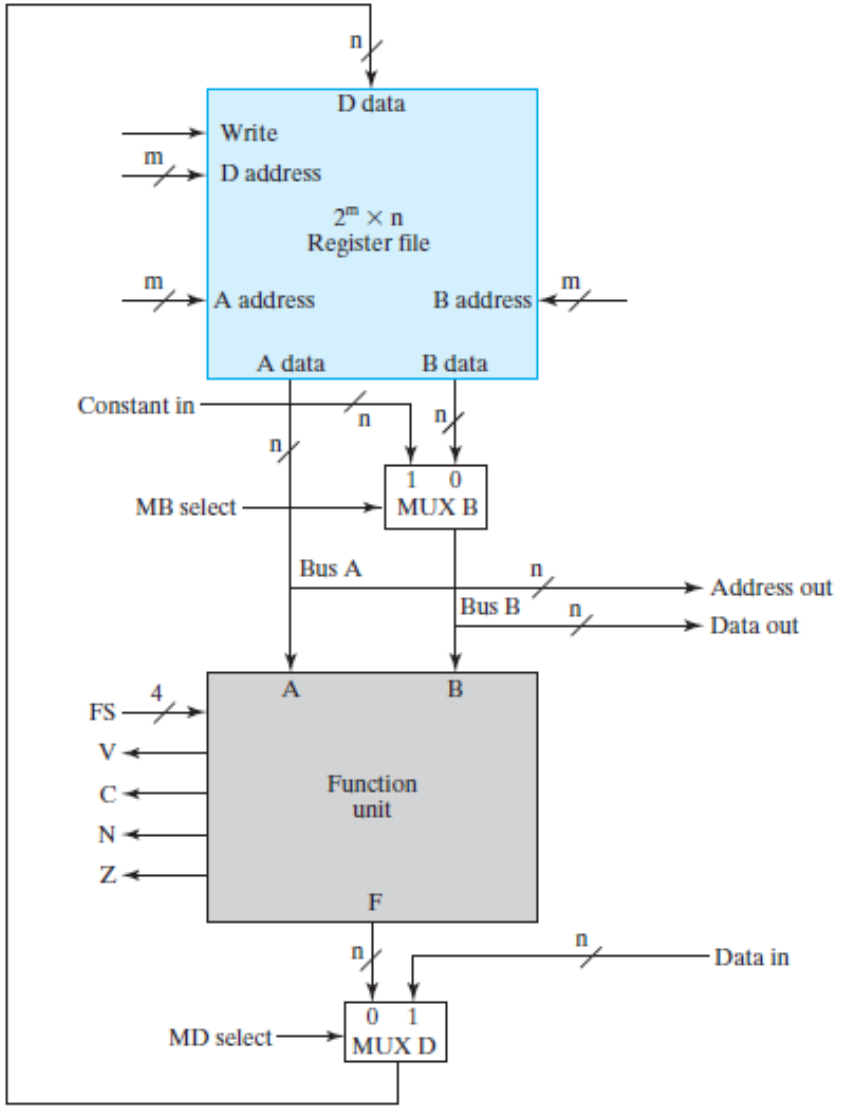


FIGURE 8-10
Block Diagram of Datapath Using the Register File and Function Unit

The Dathpath with Register File and Function Unit

- The select inputs (G select, H select, and MF select have been combined as “function select.”
- All of the codes for MF select, G select, and H select must be defined in terms of the codes for FS as shown in Table 8-4.
- From Table 8-4, it is apparent that MF is 1 for the leftmost two bits of FS both equal to 1.
- If MF select = 0, then the G select codes determine the function on the output of the function unit. If MF select = 1, then the H select codes determine the function on the output of the function unit.
- From Table 8-4, the code transformations can be implemented using the Boolean equations:

  MF = F3 . F2, G3 = F3, G2 = F2, G1 = F1,

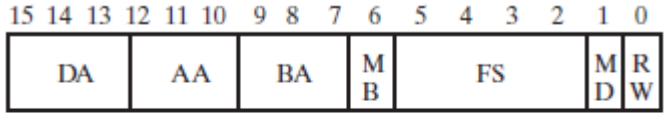
  G0 = F0, H1 = F1, and H0 = F0.

TABLE 8-4
***G* Select, *H* Select, and *MF* Select Codes Defined in Terms of *FS* Codes**

FS(3:0)	MF Select	G Select(3:0)	H Select(3:0)	Microoperation
0000	0	0000	XX	$F = A$
0001	0	0001	XX	$F = A + 1$
0010	0	0010	XX	$F = A + B$
0011	0	0011	XX	$F = A + B + 1$
0100	0	0100	XX	$F = A + \overline{B}$
0101	0	0101	XX	$F = A + \overline{B} + 1$
0110	0	0110	XX	$F = A - 1$
0111	0	0111	XX	$F = A$
1000	0	1X00	XX	$F = A \wedge B$
1001	0	1X01	XX	$F = A \vee B$
1010	0	1X10	XX	$F = A \oplus B$
1011	0	1X11	XX	$F = \overline{A}$
1100	1	XXXX	00	$F = B$
1101	1	XXXX	01	$F = \text{sr } B$
1110	1	XXXX	10	$F = \text{sl } B$

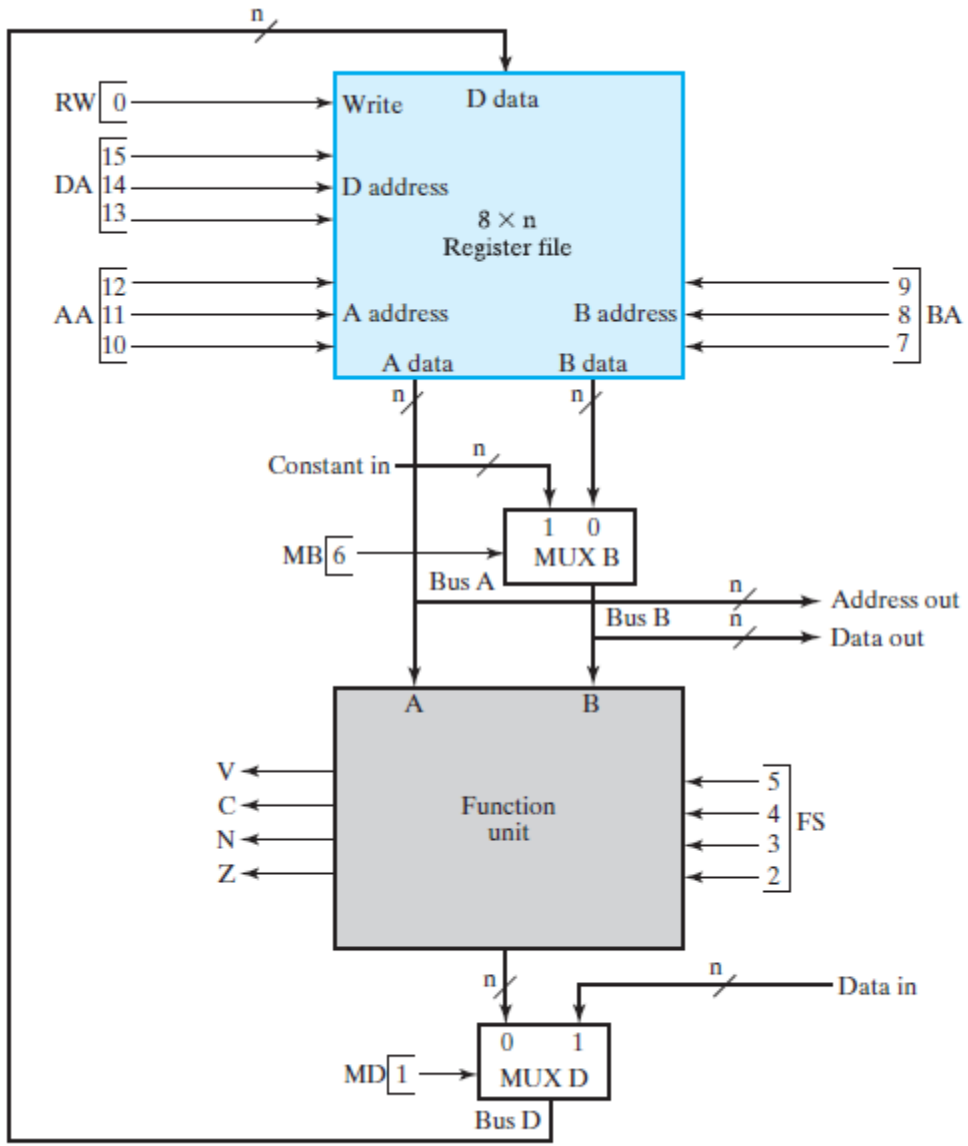
The Control Word

- The selection variables for the datapath control the microoperations executed within the datapath for any given clock pulse.
- For the datapath, the selection variables control the addresses for the data read from the register file, the function performed by the function unit, and the data loaded into the register file, as well as the selection of external data.
- We will now demonstrate how these control variables select the microoperations for the datapath. The choice of control variable values for typical microoperations will be discussed, and a simulation of the datapath will be illustrated.



(b) Control word

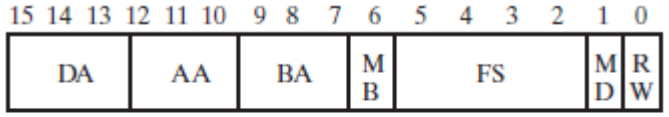
FIGURE 8-11
Datapath with Control Variables



(a) Block diagram

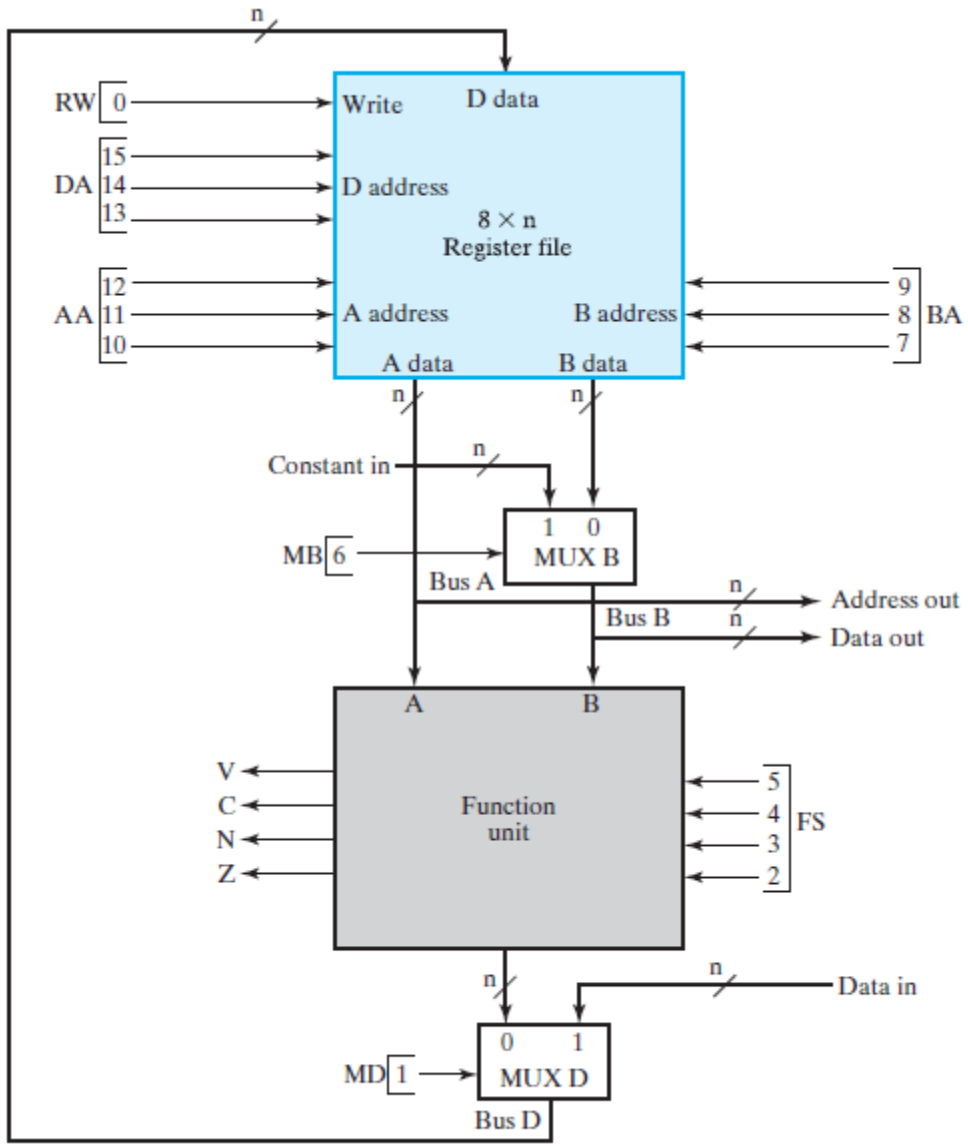
The Control Word

- A block diagram of a datapath that is a specific version of the datapath in Figure 8-10 is shown in Figure 8-11(a).
- It has a register file with eight registers, R0 through R7. The register file provides the inputs to the function unit through Bus A and Bus B.
- MUX B selects between constant values on Constant in and register values on B data.
- The ALU and zero-detection logic within the function unit generate the binary data for the four status bits: V (overflow), C (carry), Z (Zero) and N (Negative).



(b) Control word

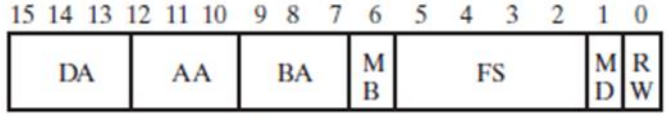
FIGURE 8-11
Datapath with Control Variables



(a) Block diagram

TABLE 8-5
Encoding of Control Word for the Datapath

DA, AA, BA		MB		FS		MD		RW	
Function	Code	Function	Code	Function	Code	Function	Code	Function	Code
$R0$	000	Register	0	$F = A$	0000	Function	0	No Write	0
$R1$	001	Constant	1	$F = A + 1$	0001	Data in	1	Write	1
$R2$	010			$F = A + B$	0010				
$R3$	011			$F = A + B + 1$	0011				
$R4$	100			$F = A + \overline{B}$	0100				
$R5$	101			$F = A + \overline{B} + 1$	0101				
$R6$	110			$F = A - 1$	0110				
$R7$	111			$F = A$	0111				
				$F = A \wedge B$	1000				
				$F = A \vee B$	1001				
				$F = A \oplus B$	1010				
				$F = \overline{A}$	1011				
				$F = B$	1100				
				$F = \text{sr } B$	1101				
				$F = \text{sl } B$	1110				

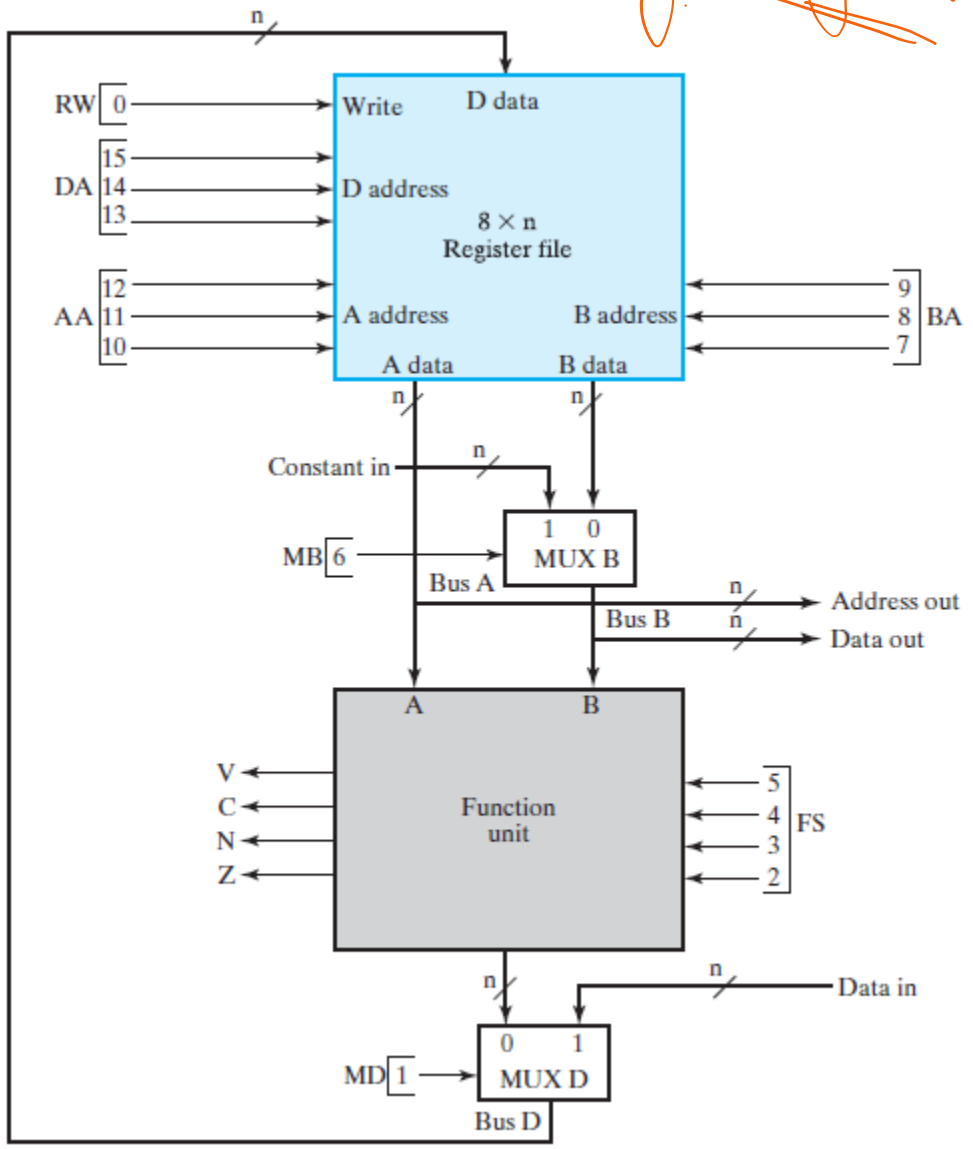


(b) Control word

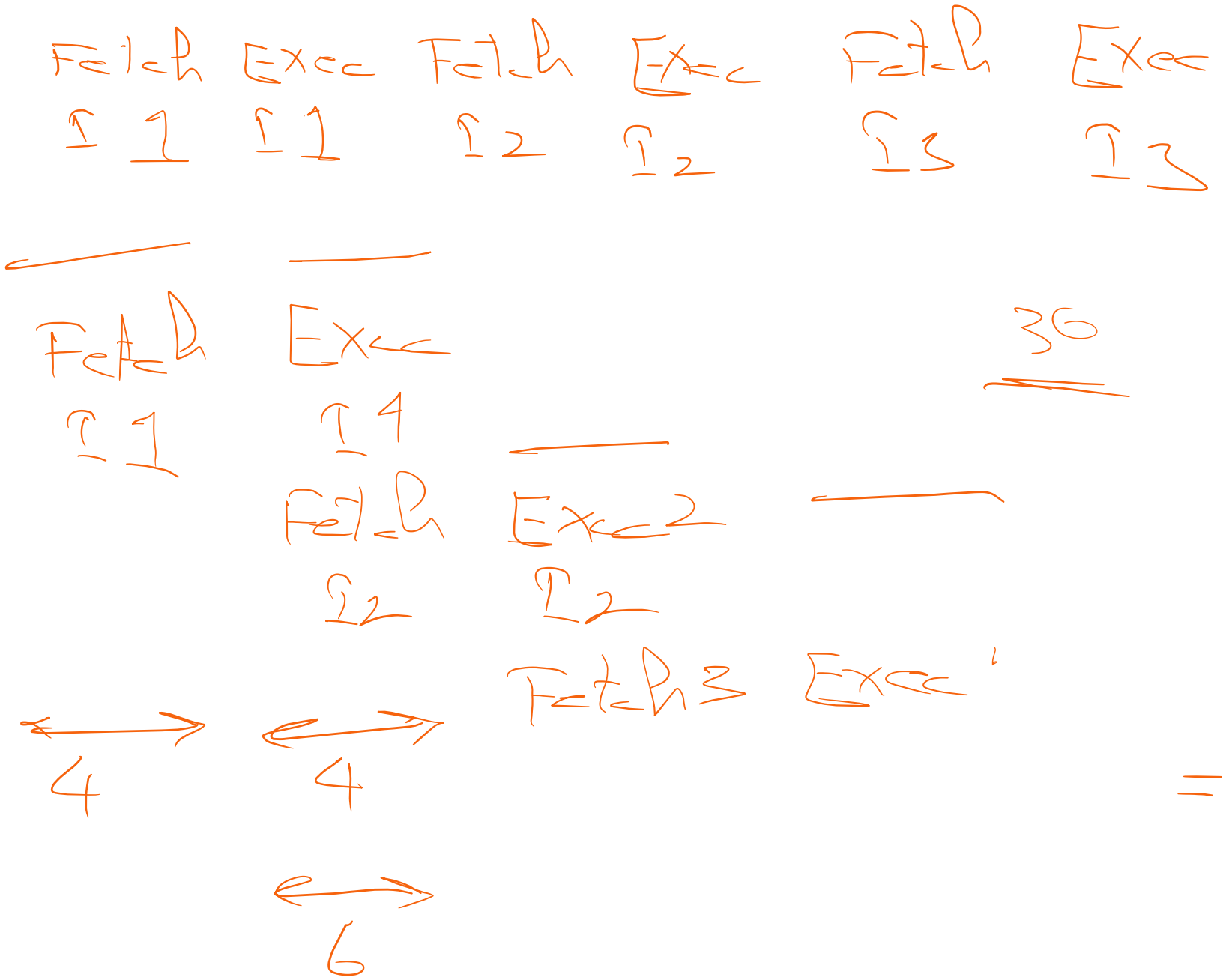
FIGURE 8-11
Datapath with Control Variables

A Simple Computer Architecture

- In a Simple Computer, a portion of **the input** to the processor consists of a **sequence of instructions**. Each instruction specifies an operation the system is to perform, which operands to use for the operation, where to place the results of the operation, and/or, in some cases, which instruction to execute next.
- These **instructions are usually stored in memory**, which is either RAM or ROM. To execute the instructions in sequence, it is **necessary to provide the address** in memory of the instruction to be executed.
- In a computer, this address comes from a register called the **program counter (PC)**. As the name implies, the PC has logic that permits it to count.
- To change the sequence of operations using decisions based on status information, the PC has **parallel load** capability, as well.



(a) Block diagram


Fetch Exec Fetch Exec Fetch Exec.
I1 I1 I2 I2 I3 I3
Fetch
I1
Exec
I4
Fetch
I2
Exec2
I2
Fetch3 Exec'
4
4
6
30
Program
Counter.
= Points to addr.
of next instr.

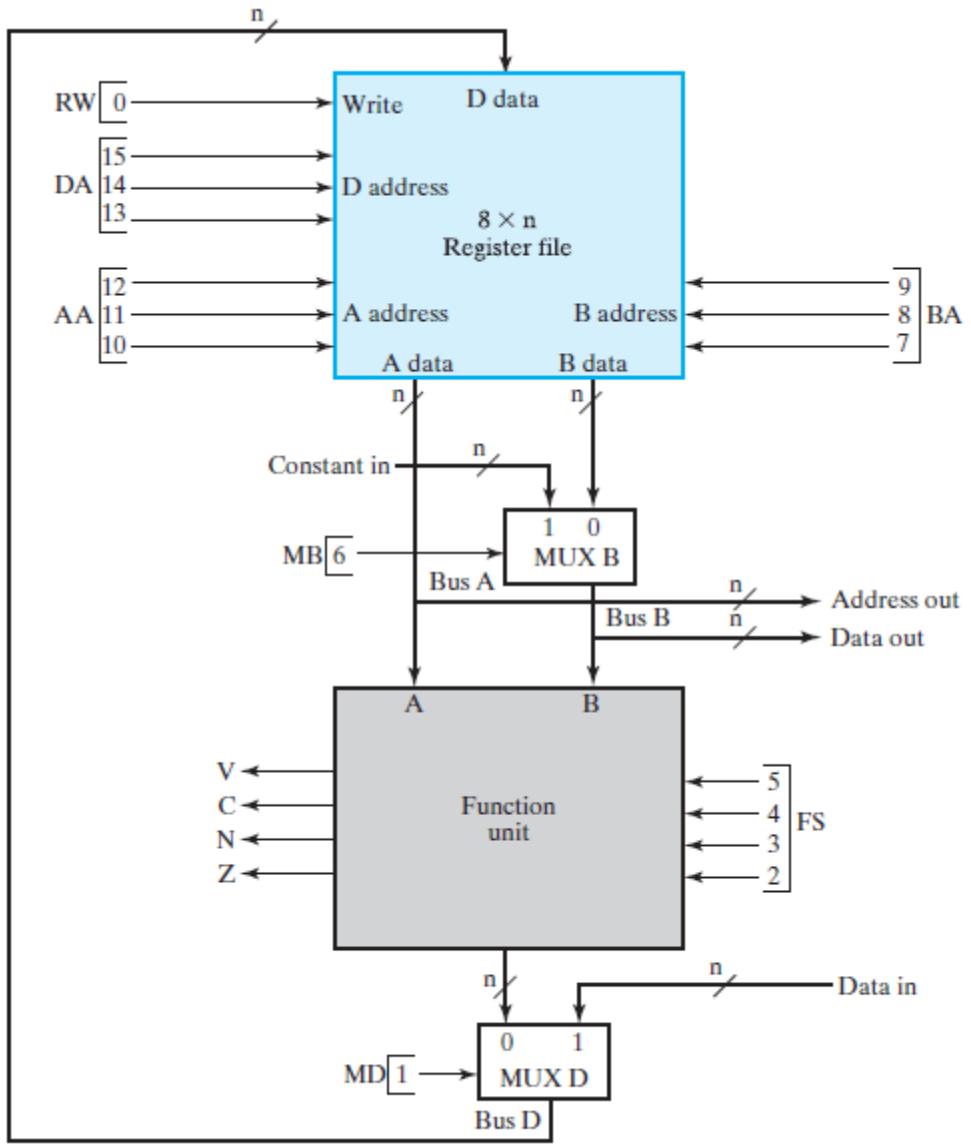
PC = not a simple register

auto increment

parallel load

A Simple Computer Architecture

- The **control unit contains a PC** and associated decision logic, as well as the necessary logic to interpret the instruction in order to execute it.
- **Executing an instruction** means activating the necessary sequence of microoperations in the datapath (and elsewhere) required to perform the operation specified by the instruction.
- There is **no PC** or similar register in a **non-programmable** system. Instead, the control unit determines the operations to be performed and the sequence of those operations, based on **only its inputs and the status bits**.



(a) Block diagram

Storage Resources

- The storage resources for the simple computer are represented by the diagram in Figure 8-13.
- The diagram depicts the **computer structure as viewed by a programmer**.
- Note that the architecture includes two memories, one for storage of instructions and the other for storage of data. These may actually be different memories, or they may be the same memory, but viewed as different from the standpoint of the CPU.
- Also visible to the programmer in the diagram is a register file with eight 16-bit registers and the 16-bit program counter.

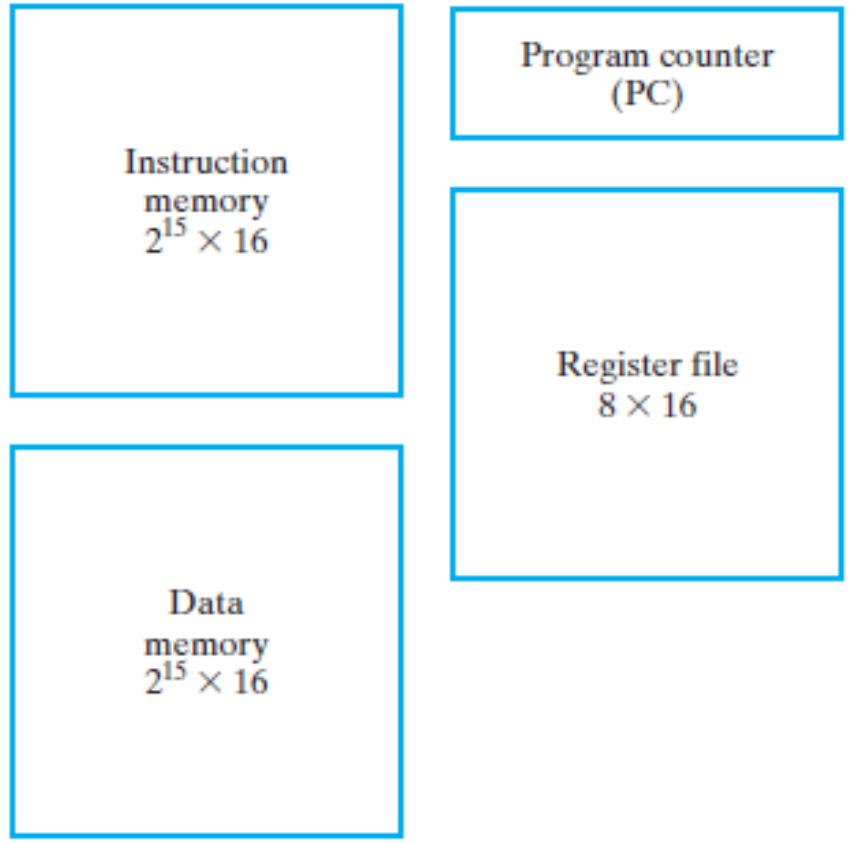


FIGURE 8-13
Storage Resource Diagram for a Simple Computer

Instruction Formats

Fields

- The format of an instruction can be understood by looking at the purpose of bits in it, combined into **groups called fields**.
- Each field is assigned a **specific item**, such as the operation code, a constant value, or a register file address.
- The various fields specify different functions for the instruction and, when shown together, constitute an instruction format.
- The operation code of an instruction, often shortened to "**opcode**," is a group of bits in the instruction that specifies an operation, such as add, subtract, shift, or complement.
- The **number of bits required for the opcode** of an instruction is a function of the total number of operations in the instruction set.

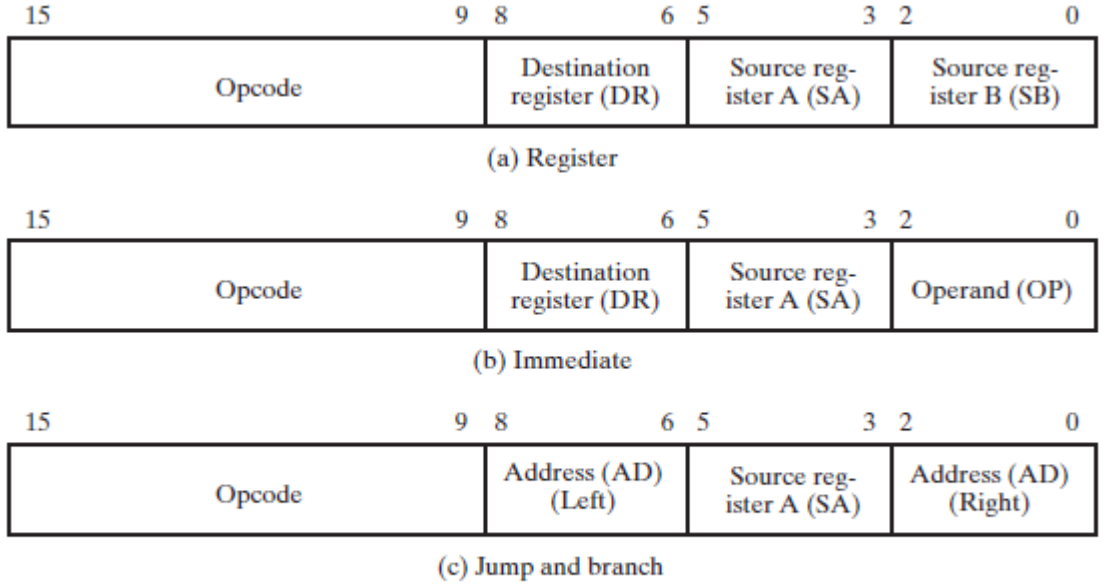


FIGURE 8-14
Three Instruction Formats

Opcode = Operation Code

100101

bits

Assembler

mnemonic = SUB

Instruction Formats

- It must consist of at least m bits for up to 2^m distinct operations.
- The **designer assigns a bit combination** (a code) to each operation. The computer is designed to accept this bit configuration at the proper time in the sequence of activities and to supply the proper control word sequence to execute the specified operation.
- As a specific example, consider a computer with a maximum of **128 distinct operations**, one of them an addition operation. The opcode assigned to this operation consists of seven bits 0000010. When the opcode 0000010 is detected by the control unit, a sequence of operations is applied to the datapath to perform the intended addition.
- The **opcode** of an instruction **specifies the operation** to be performed. Operation must be performed using data stored in the contents of the storage resources

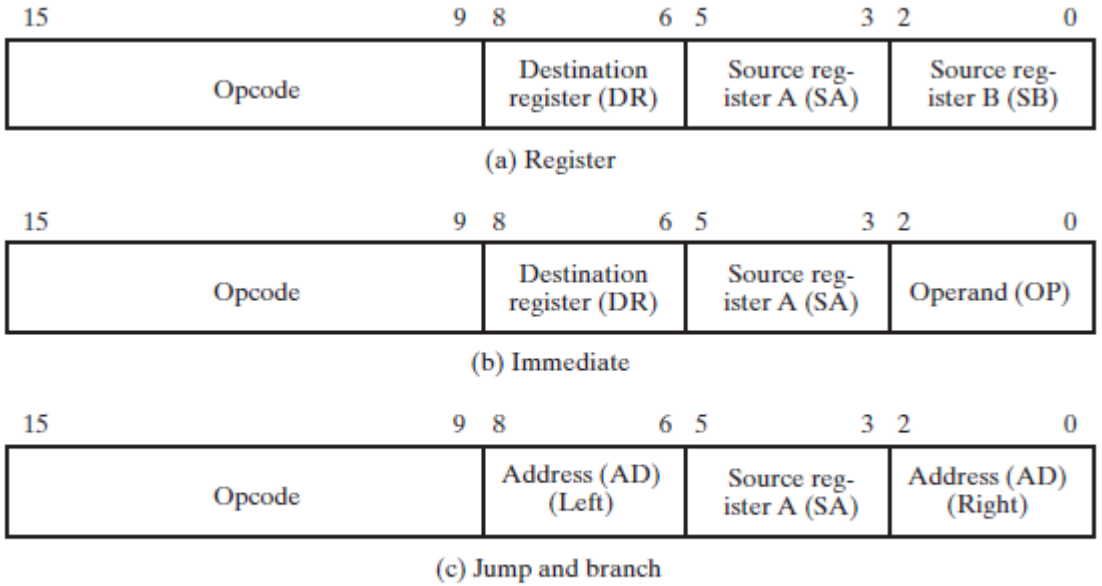


FIGURE 8-14
Three Instruction Formats

Instruction Formats

- An instruction, therefore, must specify not only the operation, but also the **registers** or **memory** words in which the operands are to be found and the result is to be placed.
- The operands may be specified by an instruction in two ways. An operand is said to be specified **explicitly** if the instruction contains special **bits for its identification**. For example, the instruction performing an addition may contain three binary numbers specifying the registers containing the two operands and the register that receives the result.
- An operand is said to be defined **implicitly** if it is included **as a part of** the definition of the operation itself, as represented by the **opcode**, rather than being given in the instruction. For example, in an Increment Register operation, one of the operands is implicitly +1.

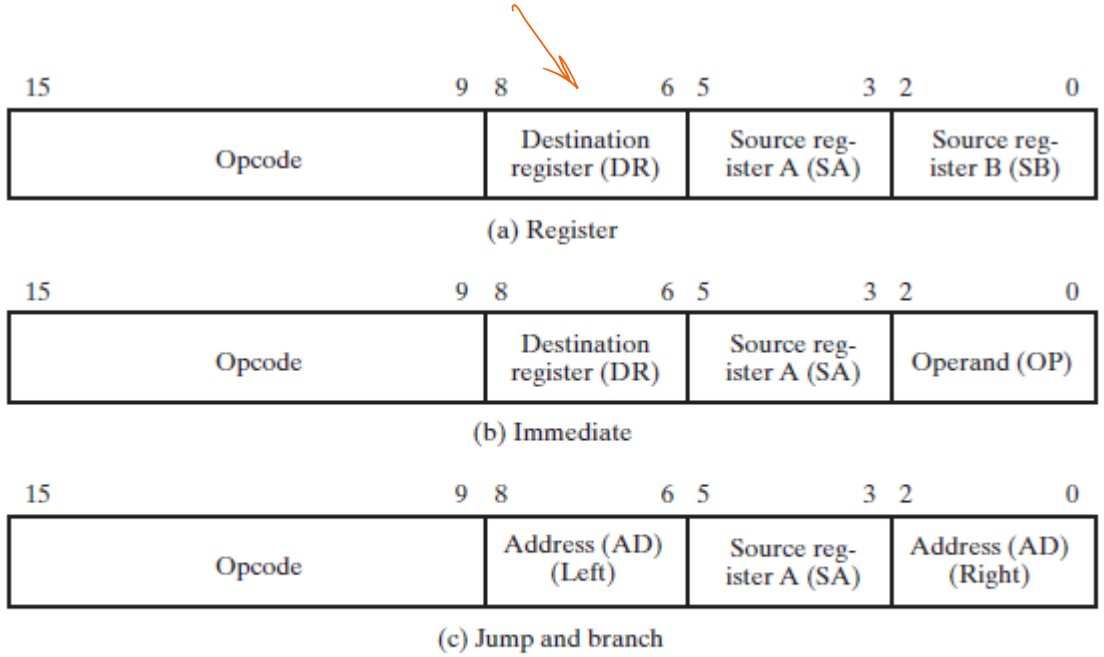


FIGURE 8-14
Three Instruction Formats

Instruction Formats

- The **three instruction formats** for the simple computer are illustrated in Figure 8-14.
- Considering a computer having a register file consisting of **eight registers, R0 through R7**. And for that instruction format contains 3 bits of address for each register.
- One of the registers is designated a destination for the result and two of the registers sources for operands. (DR for "Destination Register," SA for "Source Register A," and SB for "Source Register B.")
- The numbers of register fields and registers actually used are determined by the specific opcode. The opcode also specifies the use of the registers.
- For example, for a subtraction operation, suppose that the three bits in SA are 010, specifying R2, the three bits in SB are 011, specifying R3, and the three bits in DR are 001, specifying R1. (R1 <- R2 – R3)

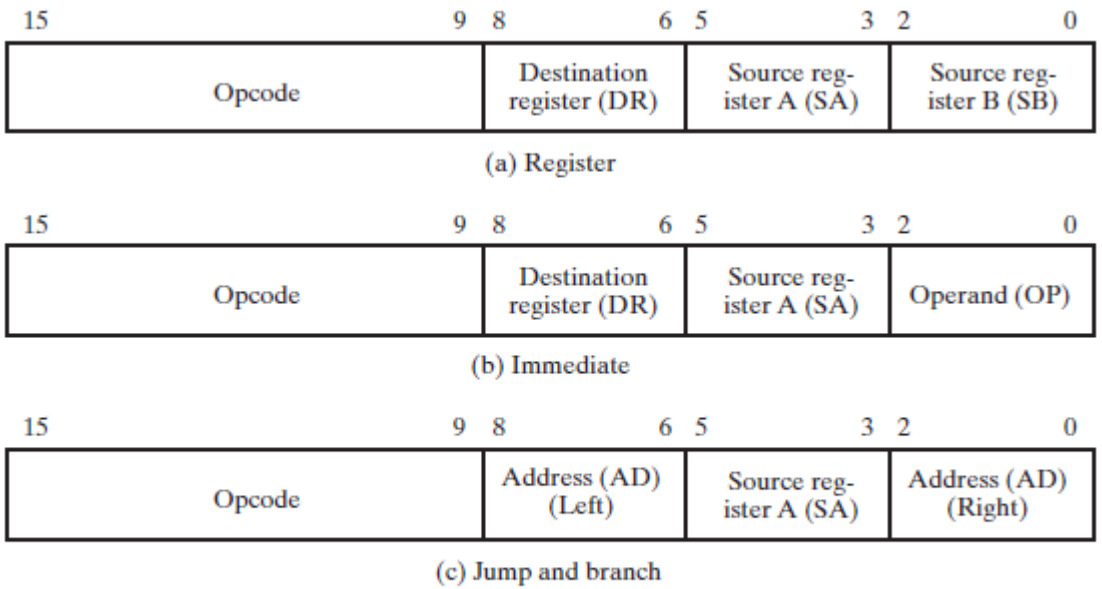


FIGURE 8-14
Three Instruction Formats

Instruction Formats

- As an example, suppose that the operation is a **store** (to memory). Suppose further, that the **three bits in SA specify R4** and the **three bits in SB specify R5**. For this particular operation, it is assumed that the register specified in SA contains the address and the register specified in SB contains the operand to be stored. So the **value in R5 is stored in the memory location given by the value in R4**. The **DR field has no effect**, since the store operation prevents the register file from being written.
- The **instruction format in Figure 8-14(b)** has an **opcode, two register fields, and an operand**. The **operand is a constant called an immediate operand**, since it is immediately available in the instruction. For example, for an add immediate operation with **SA specified as R7, DR specified as R2, and operand OP equal to 011**, the value 3 is added to the contents of R7, and the result of the addition is placed in R2.

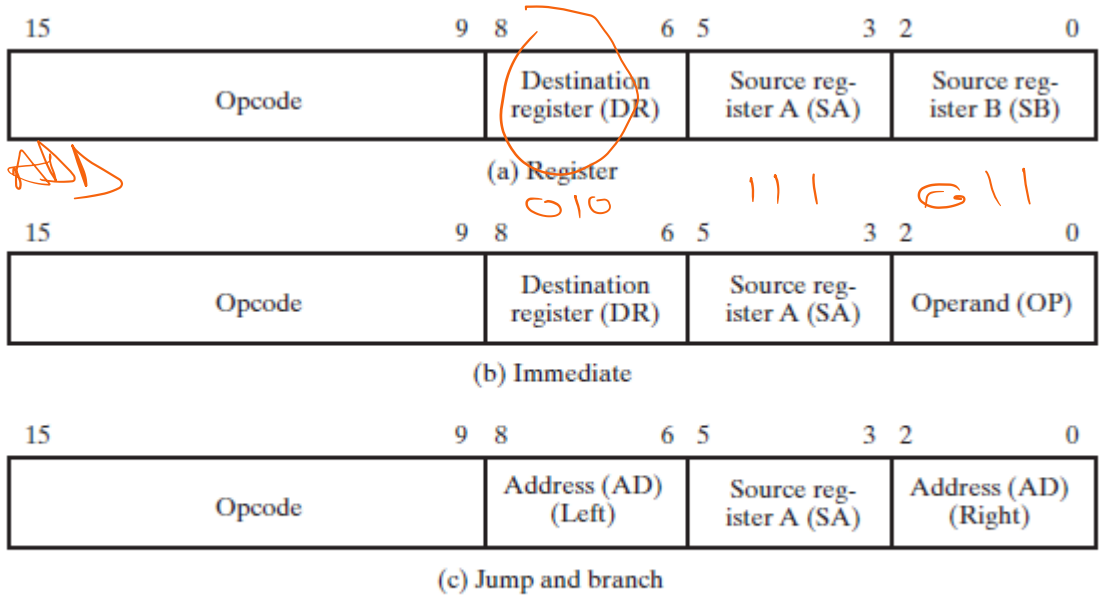


FIGURE 8-14
Three Instruction Formats

Instruction Formats

- The **instruction format in Figure 8-14(c),** in contrast to the other two formats, does not change any register file or memory contents. Instead, **it affects the order** in which the instructions are fetched from memory by providing a method for as **jumps and branches**.
- Normally, the PC fetches the instructions from sequential addresses in memory as the program is executed. But much of the power of a CPU comes from its ability **to change the order of execution** of the instructions **based on results of the processing performed**. These changes in the order of instruction execution are based on the use of instructions referred to as jumps and branches.
- The example format given in Figure 8-14(c) for jump and branch instructions has an **operation code, one register field SA, and a split address field AD**.

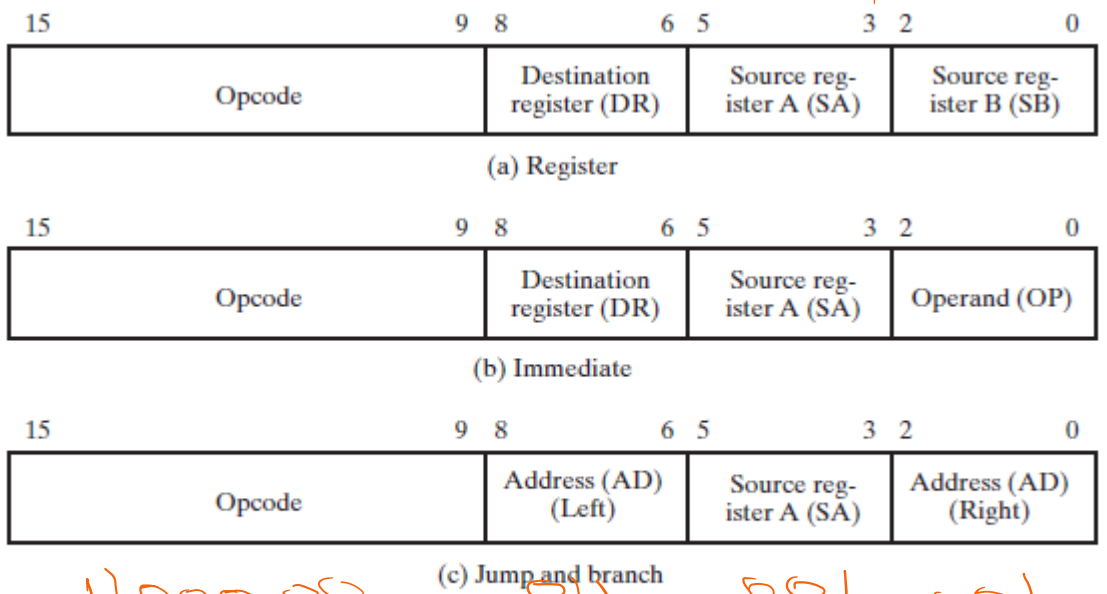


FIGURE 8-14
Three Instruction Formats

Instruction Formats

- If a branch (possibly based on the contents of the register specified) is to occur, **the new address is formed by adding the current PC contents and the contents of the 6-bit address field**.
- This addressing method is called **PC relative** and the 6-bit address field, referred to as an **address offset**, is treated as a **signed 2s complement number**.
- To preserve the 2s complement representation, sign extension is applied to the 6-bit address to form a 16-bit offset before the addition. If the leftmost bit of the address field AD is a 1, then the 10 bits to its left are filled with 1s to give a negative 2s complement offset. If the leftmost bit of the address field is 0, then the 10 bits to its left are filled with 0s to give a positive 2s complement offset.
- The offset is added to the PC to form the location from which the next instruction is to be fetched.

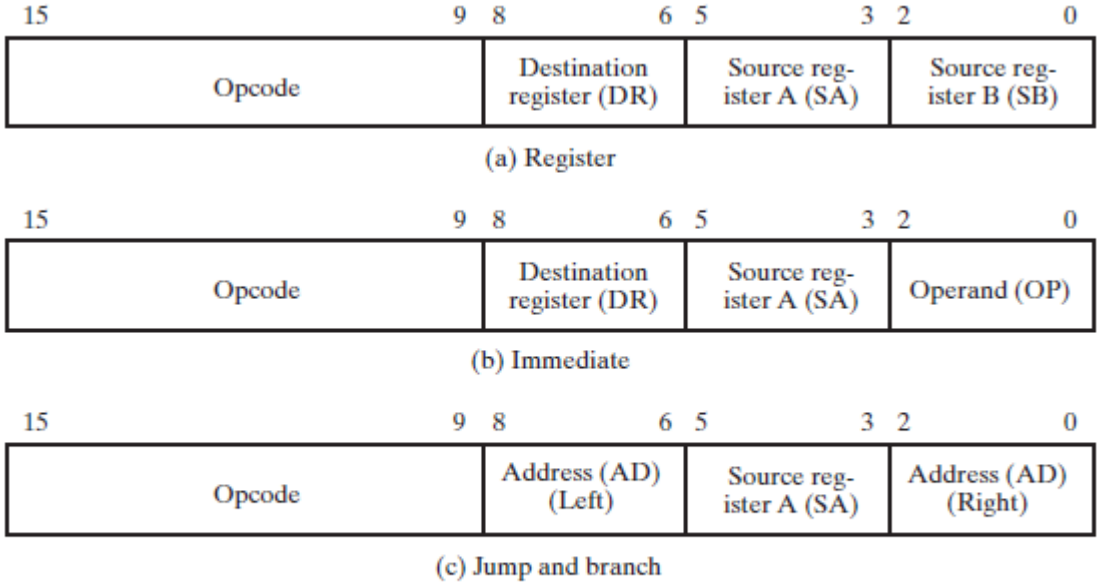


FIGURE 8-14
Three Instruction Formats

Instruction Formats

- For example, with the PC value equal to 55, suppose that a branch is to occur to location 35 if the contents of R6 is equal to zero. The opcode would specify a branch-on-zero instruction, SA would be specified as R6, and AD would be the 6-bit, 2s complement representation of -20. If R6 is zero, then PC contents becomes 55 + (-20) = 35, and the next instruction will be fetched from address 35.
- Otherwise, if R6 is nonzero, the PC will count up to 56, and the next instruction will be fetched from address 56.
- This addressing method alone provides branch addresses within a small range below and above the PC value. The jump provides a broader range of addresses by using the unsigned contents of a 16-bit register as the jump target.

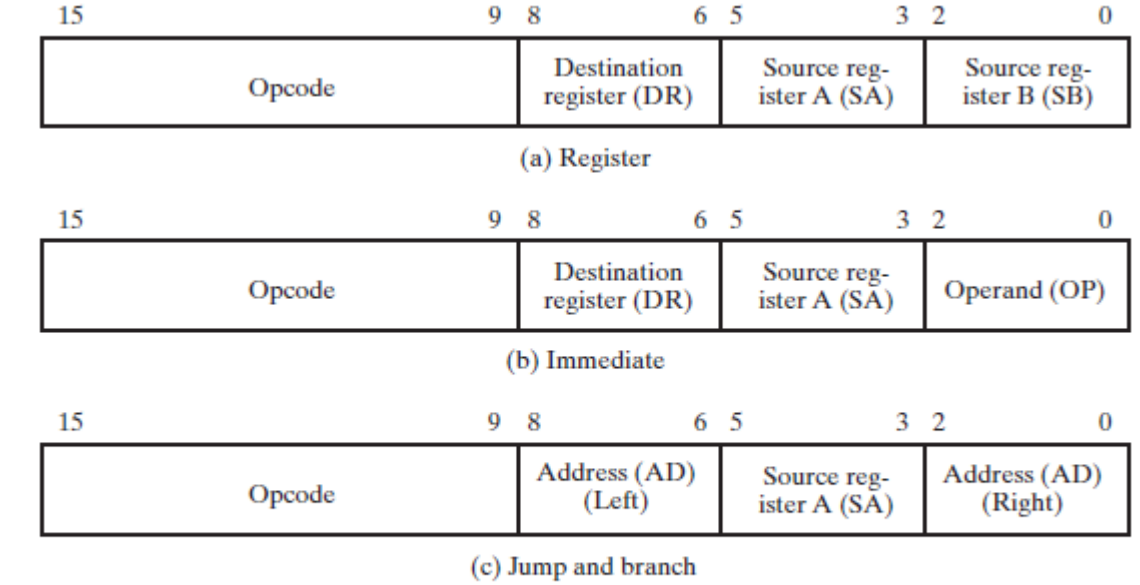


FIGURE 8-14
Three Instruction Formats

Instruction Specifications

- For each instruction, the opcode is given along with a shorthand name called a **mnemonic**, which can be used as a symbolic representation for the opcode.
- This mnemonic represents the notation to be used in specifying all of the fields of the instruction symbolically.
- This symbolic representation is then converted to the binary representation of the instruction by a program called an **assembler**.
- In the table, a description of the operation performed by the instruction execution is given, including the status bits that are affected by the instruction.
- The instruction specifications for the simple computer are given in Table 8-8.

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] ← $\overline{R[SA]}$*	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]*	
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.

BL = 1100 1100 (Addr. bus is 8 bit)

3bit

101 = -3

0	1	1	1	= 7
0	1	0	1	= +5
1	1	0	0	= +12

4bit

0	1 1 1	7
1	1 0 1	-3
8	-4 -1	4

2-4

-19

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] ← $\overline{R[SA]}$*	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]*	
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.

Assignment 3 Due Nov 10

- Using Table 8-8 and Figure 8-14:
- Explain the function of each instruction.
- Write the complete binary code for each instruction.
- Assume RD = R0, RA = R2, RB = R3.
- Assume OP = 100 (binary),

  AD = 100 001 (binary).

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] ← $\overline{R[SA]}$*	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]*	
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.

Instruction Specifications

- Suppose that we have a memory with 16 bits per word with instructions having one of the formats in Figure 8-14.
- Instructions and data, in binary, are placed in memory, as shown in Table 8-9.
- This stored information represents the four instructions illustrating the distinct formats.
- At address 25, we have a register format instruction that specifies an operation to subtract R3 from R2 and load the difference into R1.

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	0000000011000000	Data = 192. After execution of instruction in 35, Data = 80.		

- In memory location 35 is a register format instruction to store the contents of R5 in the memory location specified by R4.
- Suppose R4 contains 70 and R5 contains 80. Then the execution of this instruction will store the value 80 in memory location 70, replacing the original value of 192 stored there.

Instruction Specifications

- At address 45, an immediate format instruction appears that adds 3 to the contents of R7 and loads the result into R2.
- The operand to be added is the value 3 (011) in the OP field, the last three bits of the instruction.
- In location 55, the branch instruction appears. The opcode for this instruction is 96, and source register A is specified as R6.
- Note that AD (Left) contains 101 and AD (Right) contains 100.

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)	,SA:6	If R6 = 0, PC ← PC – 20
70	0000000011000000	Data = 192. After execution of instruction in 35, Data = 80.		

- Putting these two together and applying sign extension, we obtain 1111111111101100, which represents -20 in 2s complement. If register R6 is zero, then -20 is added to the PC to give 35.
- If register R6 is nonzero, the new PC value will be 56.
- We assumed that the addition to the PC content occurs before the PC has been incremented.

Instruction Specifications

- At address 45, an immediate format instruction appears that adds 3 to the contents of R7 and loads the result into R2.
- The operand to be added is the value 3 (011) in the OP field, the last three bits of the instruction.
- In location 55, the branch instruction appears. The opcode for this instruction is 96, and source register A is specified as R6.
- Note that AD (Left) contains 101 and AD (Right) contains 100.

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)	,SA:6	If R6 = 0, PC ← PC – 20
70	0000000011000000	Data = 192. After execution of instruction in 35, Data = 80.		

- Putting these two together and applying sign extension, we obtain 1111111111101100, which represents -20 in 2s complement. If register R6 is zero, then -20 is added to the PC to give 35.
- If register R6 is nonzero, the new PC value will be 56.
- We assumed that the addition to the PC content occurs before the PC has been incremented.