Revision
Unit – I
Chapter - 1

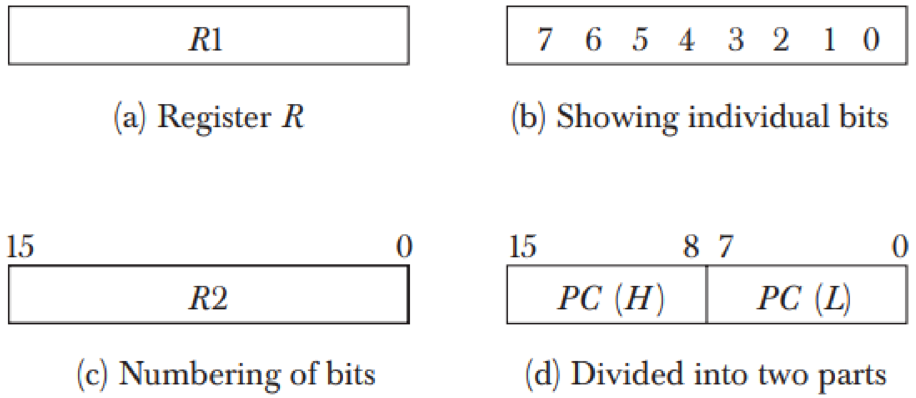
Register Transfer Language: Register transfer language, bus and memory transfer, bus architecture using multiplexer and tri-state buffer, micro-operation: arithmetic, logical, shift micro-operation with hardware implementation, arithmetic logic shift unit.

REGISTER TRANSFER LANGUAGE

The internal hardware organization of a digital computer is best defined by specifying:

1. The set of registers it contains and their function.
2. The sequence of microoperations performed on the binary information stored in the registers.
3. The control that initiates the sequence of microoperations.

Figure 4-1 Block diagram of register.



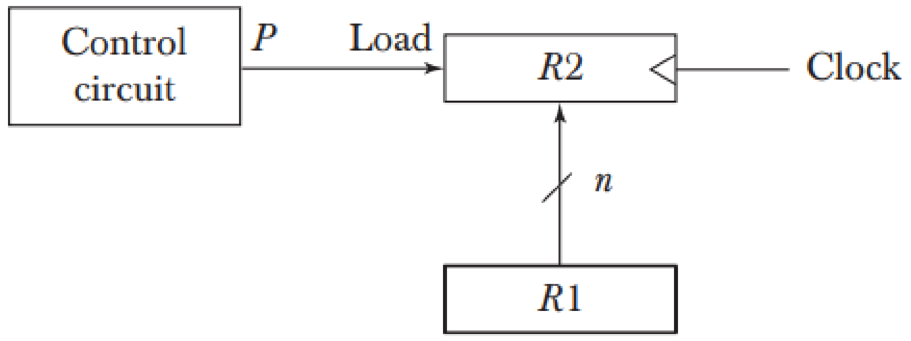
The symbolic notation used to describe the microoperation transfers among registers is called a **register transfer language.**

$$\textit{If } (P = 1) \textit{ then } (R2 \leftarrow R1) \qquad P: \quad R2 \leftarrow R1$$

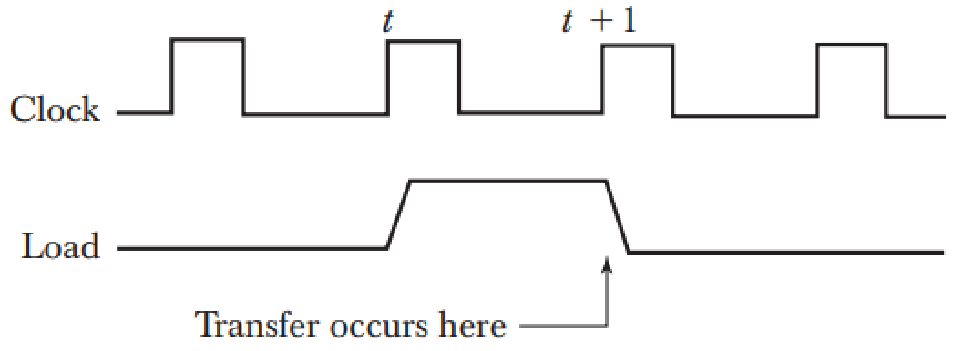
TABLE 4-1 Basic Symbols for Register Transfers

Symbol	Description	Examples
Letters (and numerals)	Denotes a register	*MAR*, *R2*
Parentheses ( )	Denotes a part of a register	$R2(0-7)$, $R2(L)$
Arrow ←	Denotes transfer of information	$R2 \leftarrow R1$
Comma ,	Separates two microoperations	$R2 \leftarrow R1$, $R1 \leftarrow R2$

Figure 4-2 Transfer from **R**1 to **R**2 when $p = 1$.



(a) Block diagram



(b) Timing diagram

BUS AND MEMORY TRANSFER, BUS ARCHITECTURE USING MULTIPLEXER

TABLE 4-2 Function Table for Bus of Fig. 4-3

S_1	S_0	Register selected
0	0	A
0	1	B
1	0	C
1	1	D

$BUS \leftarrow C, \qquad R1 \leftarrow BUS$

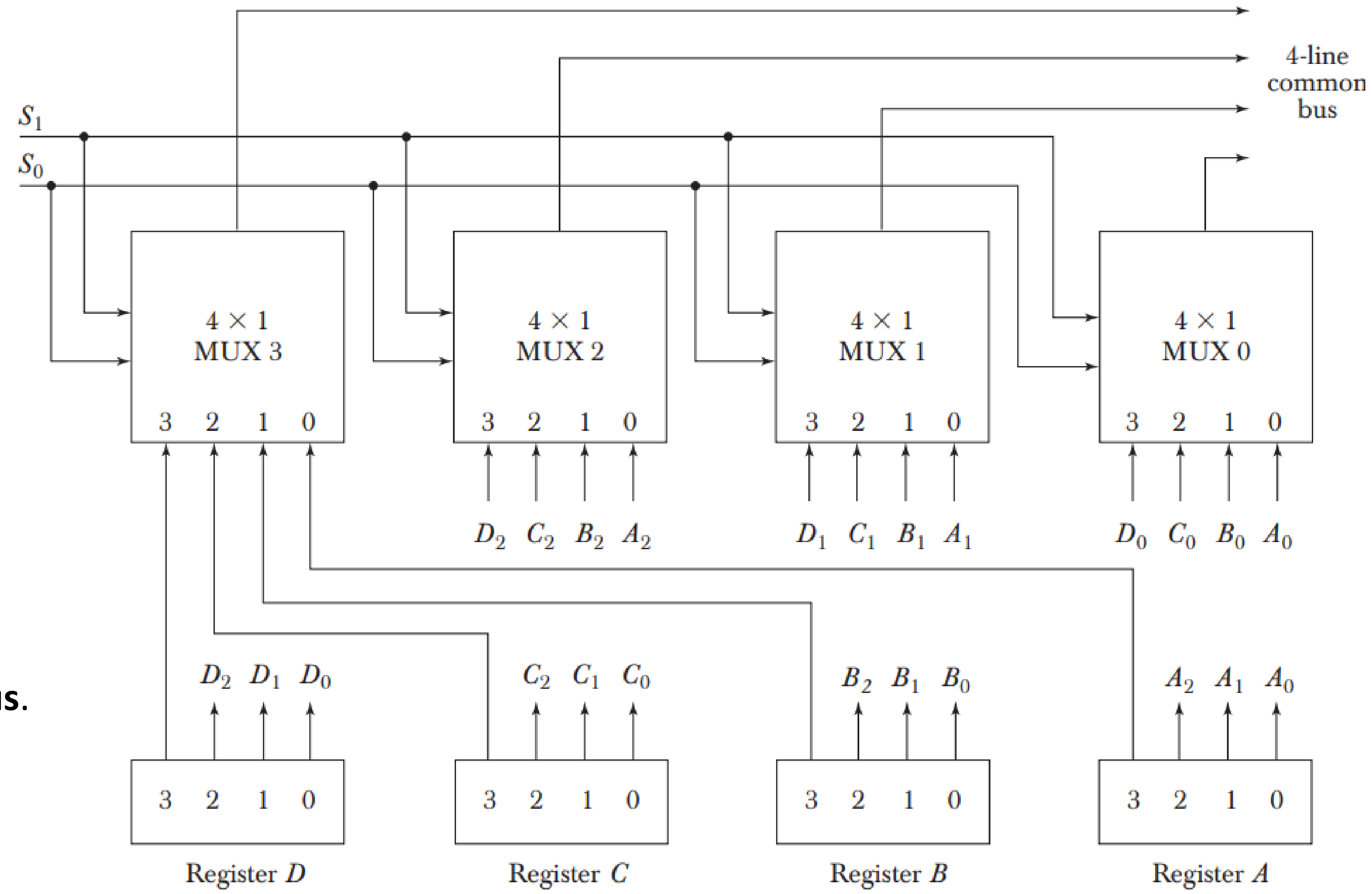
$R1 \leftarrow C$

In general,
A bus system will multiplex **k registers** of **n bits** each to produce an **n-line common bus**.

The **number of multiplexers needed** to construct the bus is **equal to n.**

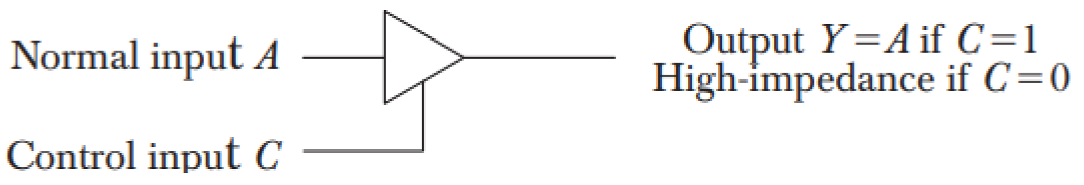
The **size of each multiplexer** must be **kX1.**

Figure 4-3 Bus system for four registers.



BUS AND MEMORY TRANSFER, BUS ARCHITECTURE USING TRI-STATE BUFFER

Figure 4-4 Graphic symbols for three-state buffer.



Read: $DR \leftarrow M[AR]$

Write: $M[AR] \leftarrow R1$

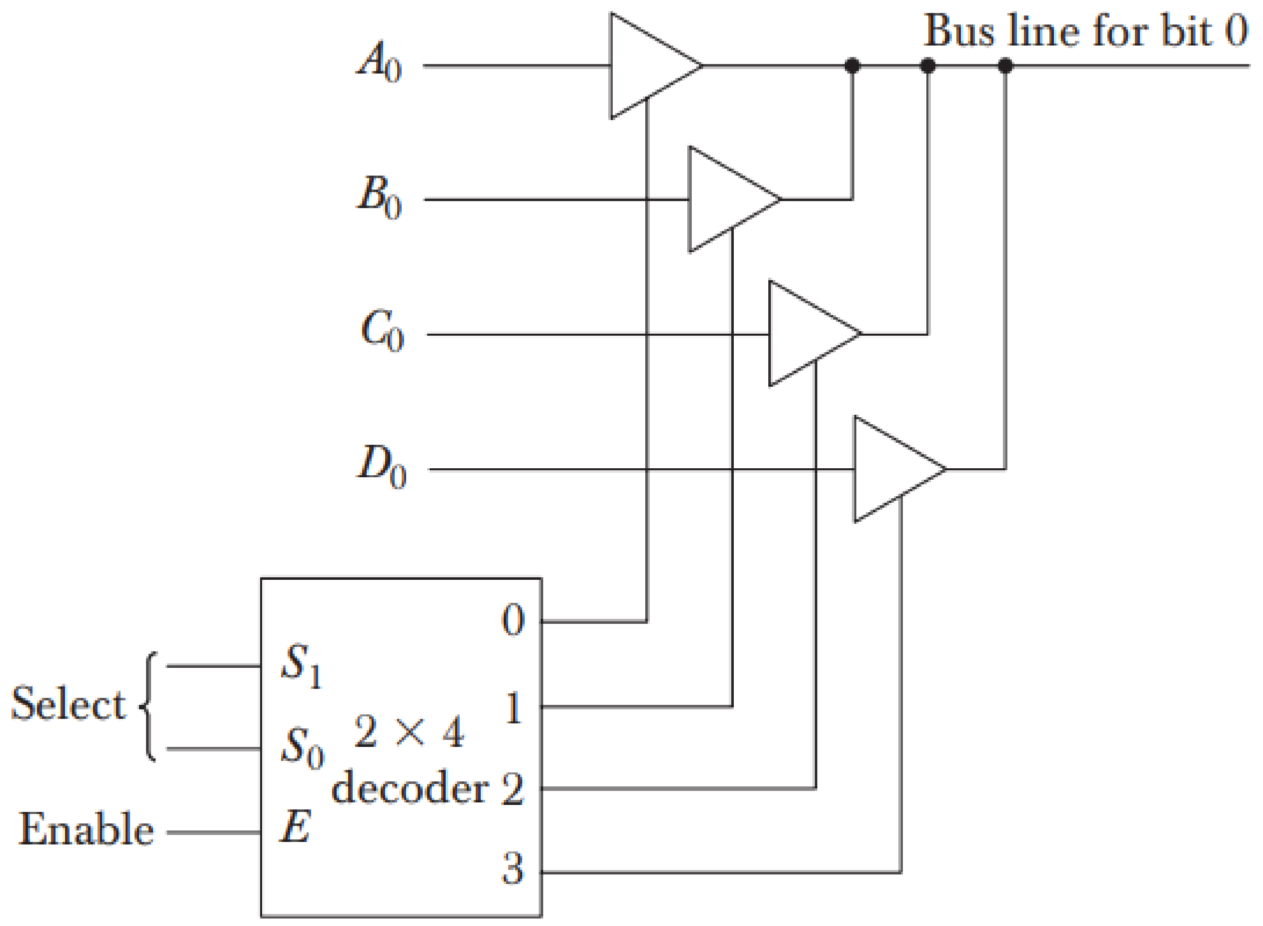


Figure 4-5 Bus line with three state-buffers.

MICRO-OPERATION: ARITHMETIC

TABLE 4-3 Arithmetic Microoperations

Symbolic designation	Description
$R3 \leftarrow R1 + R2$	Contents of $R1$ plus $R2$ transferred to $R3$
$R3 \leftarrow R1 - R2$	Contents of $R1$ minus $R2$ transferred to $R3$
$R2 \leftarrow \overline{R2}$	Complement the contents of $R2$ (1's complement)
$R2 \leftarrow \overline{R2} + 1$	2's complement the contents of $R2$ (negate)
$R3 \leftarrow R1 + \overline{R2} + 1$	$R1$ plus the 2's complement of $R2$ (subtraction)
$R1 \leftarrow R1 + 1$	Increment the contents of $R1$ by one
$R1 \leftarrow R1 - 1$	Decrement the contents of $R1$ by one

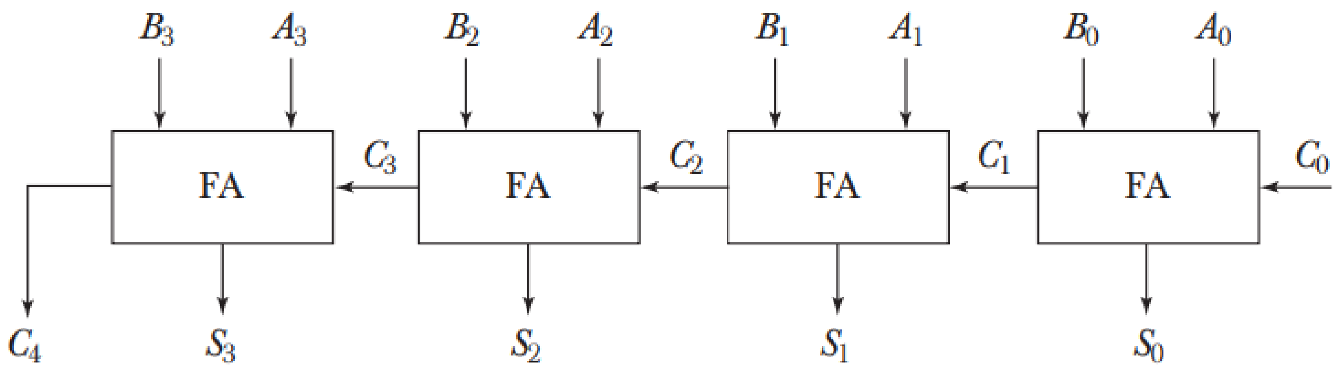


Figure 4-6 4-bit binary adder.

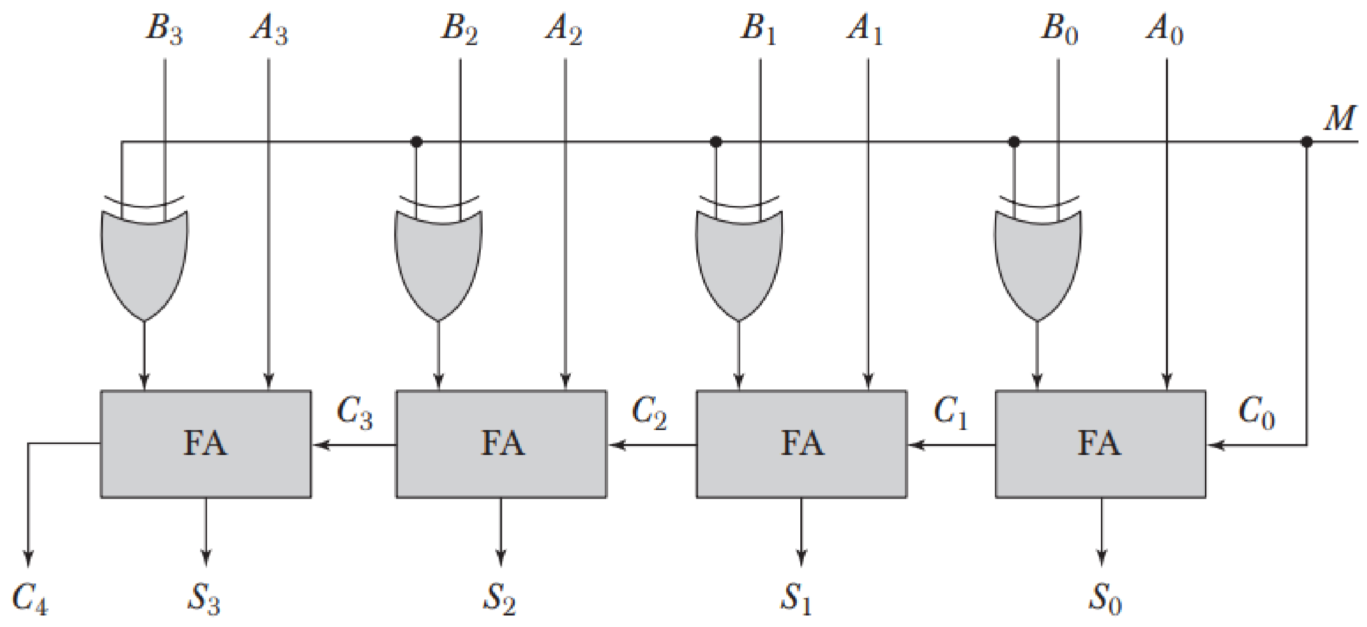


Figure 4-7 4-bit adder-subtractor.

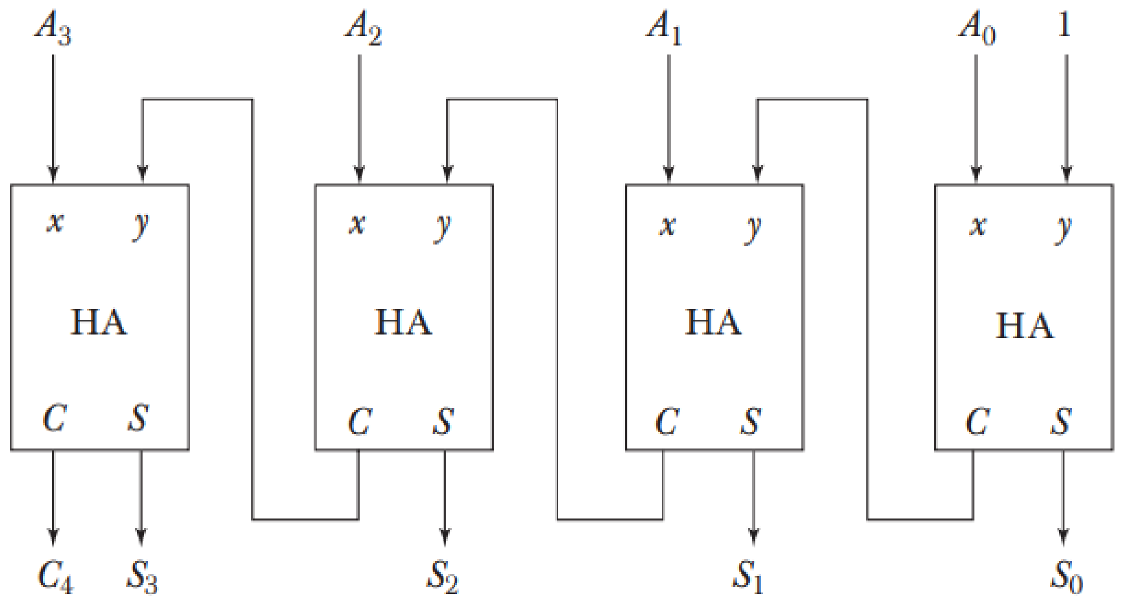


Figure 4-8 4-bit binary incrementer.

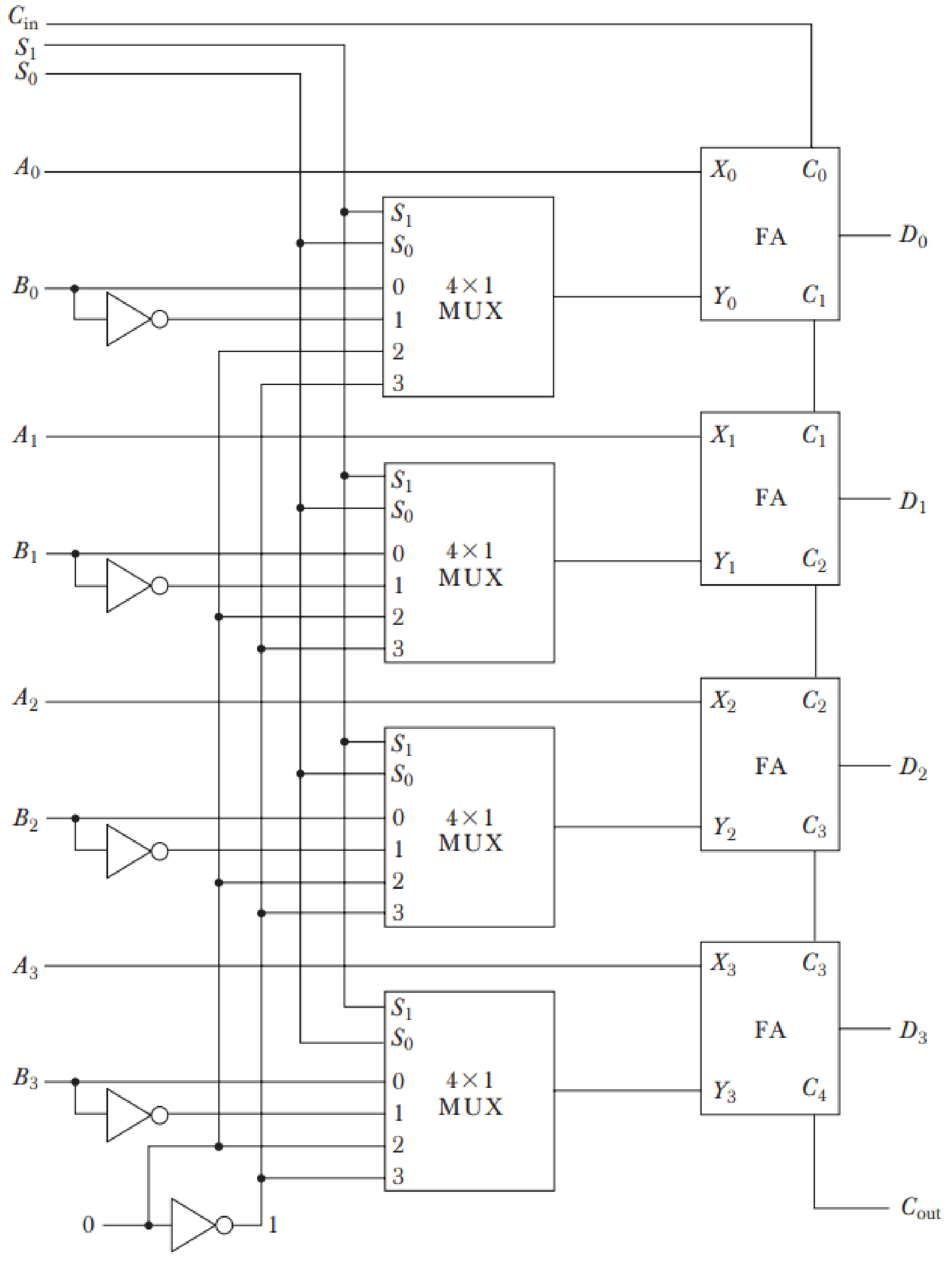


Figure 4-9 4-bit arithmetic circuit.

TABLE 4-4 Arithmetic Circuit Function Table

Select			Input	Output	
S_1	S_0	C_{in}	Y	$D = A + Y + C_{in}$	Microoperation
0	0	0	B	$D = A + B$	Add
0	0	1	B	$D = A + B + 1$	Add with carry
0	1	0	$\overline{B}$	$D = A + \overline{B}$	Subtract with borrow
0	1	1	$\overline{B}$	$D = A + \overline{B} + 1$	Subtract
1	0	0	0	$D = A$	Transfer A
1	0	1	0	$D = A + 1$	Increment A
1	1	0	1	$D = A - 1$	Decrement A
1	1	1	1	$D = A$	Transfer A

MICRO-OPERATION: LOGICAL

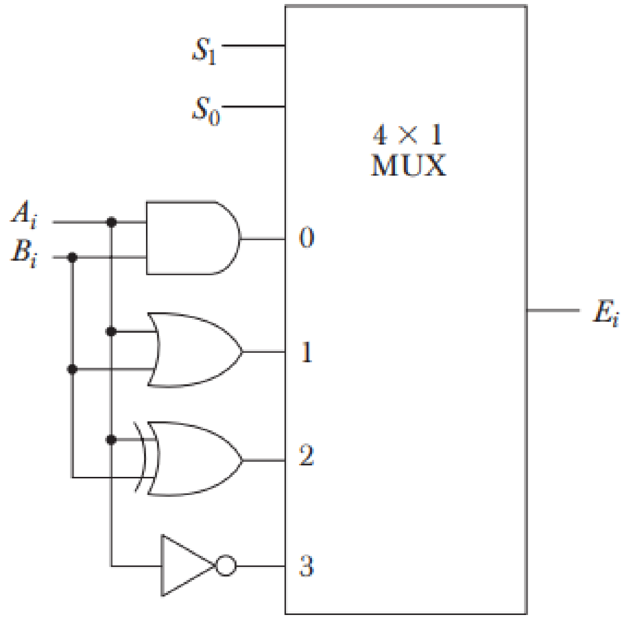
TABLE 4-6 Sixteen Logic Microoperations

Boolean function	Microoperation	Name
$F_0 = 0$	$F \leftarrow 0$	Clear
$F_1 = xy$	$F \leftarrow A \wedge B$	AND
$F_2 = xy'$	$F \leftarrow A \wedge \overline{B}$	
$F_3 = x$	$F \leftarrow A$	Transfer A
$F_4 = x'y$	$F \leftarrow \overline{A} \wedge B$	
$F_5 = y$	$F \leftarrow B$	Transfer B
$F_6 = x \oplus y$	$F \leftarrow A \oplus B$	Exclusive-OR
$F_7 = x + y$	$F \leftarrow A \vee B$	OR
$F_8 = (x + y)'$	$F \leftarrow \overline{A \vee B}$	NOR
$F_9 = (x \oplus y)'$	$F \leftarrow \overline{A \oplus B}$	Exclusive-NOR
$F_{10} = y'$	$F \leftarrow \overline{B}$	Complement B
$F_{11} = x + y'$	$F \leftarrow A \vee \overline{B}$	
$F_{12} = x'$	$F \leftarrow \overline{A}$	Complement A
$F_{13} = x' + y$	$F \leftarrow \overline{A} \vee B$	
$F_{14} = (xy)'$	$F \leftarrow \overline{A \wedge B}$	NAND
$F_{15} = 1$	$F \leftarrow$ all 1's	Set to all 1's

TABLE 4-5 Truth Tables for 16 Functions of Two Variables

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Figure 4-10 One stage of logic circuit.



(a) Logic diagram

S_1	S_0	Output	Operation
0	0	$E = A \wedge B$	AND
0	1	$E = A \vee B$	OR
1	0	$E = A \oplus B$	XOR
1	1	$E = \overline{A}$	Complement

(b) Functional table

selective-set	1010	A before
	1100	B (logic operand)
	1110	A after
selective-complement	1010	A before
	1100	B (logic operand)
	0110	A after
selective-clear	1010	A before
	1100	B (logic operand)
	0010	A after

0110 1010	A before
0000 1111	B (mask)
0000 1010	A after masking

0000 1010	A before
1001 0000	B (insert)
1001 1010	A after insertion

MICRO-OPERATION: SHIFT

$R1 \leftarrow \text{shl } R1$
$R2 \leftarrow \text{shr } R2$

TABLE 4-7 Shift Microoperations

Symbolic designation	Description
$R \leftarrow \text{shl } R$	Shift-left register R
$R \leftarrow \text{shr} R$	Shift-right register R
$R \leftarrow \text{cil } R$	Circular shift-left register R
$R \leftarrow \text{cir } R$	Circular shift-right register R
$R \leftarrow \text{ashl } R$	Arithmetic shift-left R
$R \leftarrow \text{ashr } R$	Arithmetic shift-right R

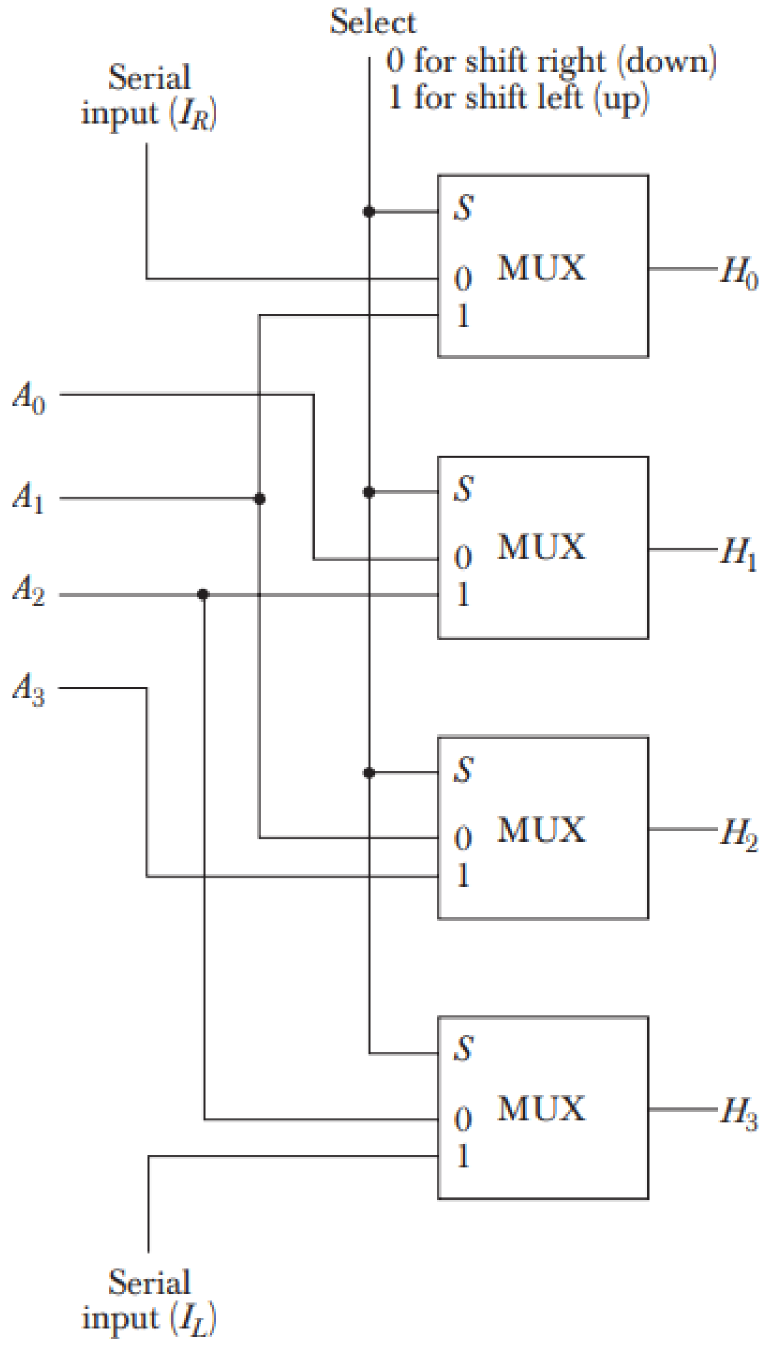


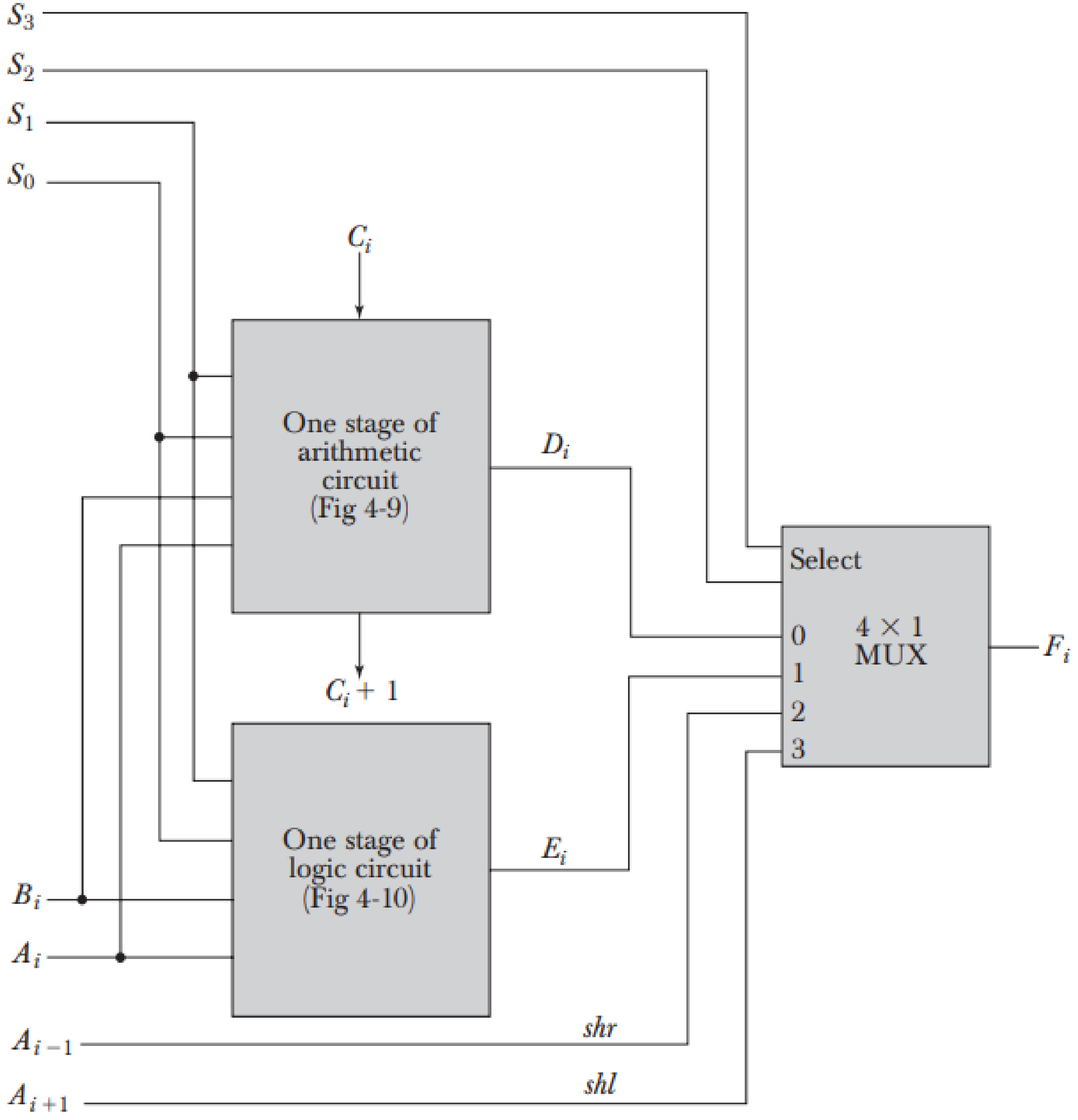
Figure 4-12 4-bit combinational circuit shifter.

Functional table

Select	Output			
S	H_0	H_1	H_2	H_2
0	I_R	A_0	A_1	A_2
1	A_1	A_2	A_3	I_L

ARITHMETIC LOGIC SHIFT UNIT

Figure 4-13 One stage of arithmetic logic shift unit.



4-1. Show the block diagram of the hardware (similar to Fig. 4-2a) that implements the following register transfer statement:

$$yT_2: \quad R2 \leftarrow R1, \quad R1 \leftarrow R2$$

4-3. Represent the following conditional control statement by two register transfer statements with control functions.

$$\text{If } (P = 1) \text{ then } (R1 \leftarrow R2) \text{ else if } (Q = 1) \text{ then } (R1 \leftarrow R3)$$

4-7. The following transfer statements specify a memory. Explain the memory operation in each case.

a. $R2 \leftarrow M[AR]$
b. $M[AR] \leftarrow R3$
c. $R5 \leftarrow M[R5]$

4-18. Register A holds the 8-bit binary 11011001. Determine the B operand and the logic microoperation to be performed in order to change the value in A to:

a. 01101101
b. 11111101

Revision
Unit – I
Chapter - 2

Computer Organization and Design: Instruction codes, general computer registers with common bus system, computer instructions: memory reference, register reference, input-output instructions, timing and control, instruction cycle, input-output configuration, and interrupt cycle. Levels of programming languages: Machine language, Assembly language, High level language.

INSTRUCTION CODES, GENERAL COMPUTER REGISTERS

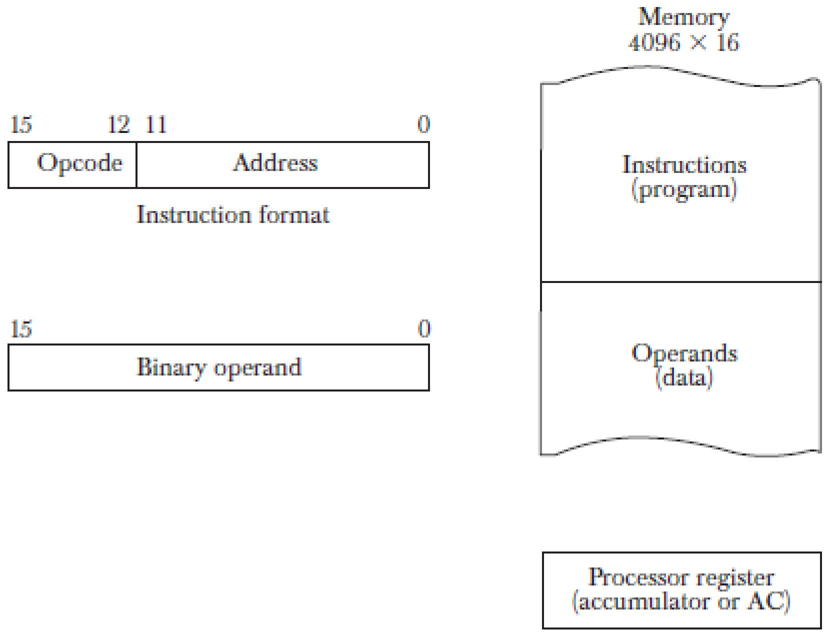


Figure 5-1 Stored program organization.

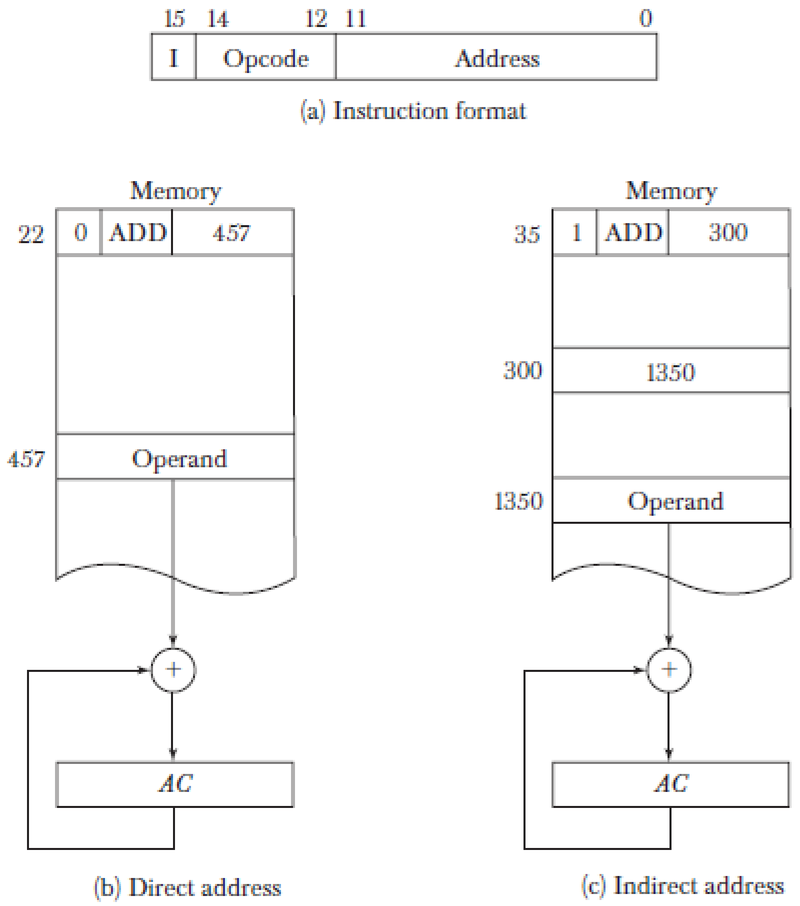


Figure 5-2 Demonstration of direct and indirect address.

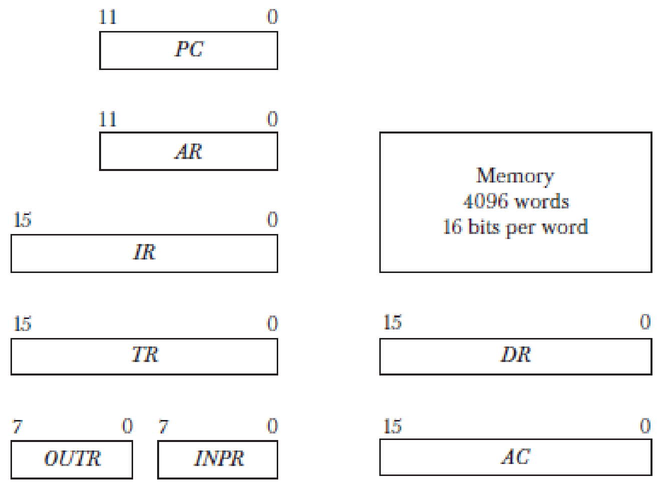


Figure 5-3 Basic computer registers and memory.

COMMON BUS SYSTEM, COMPUTER INSTRUCTIONS

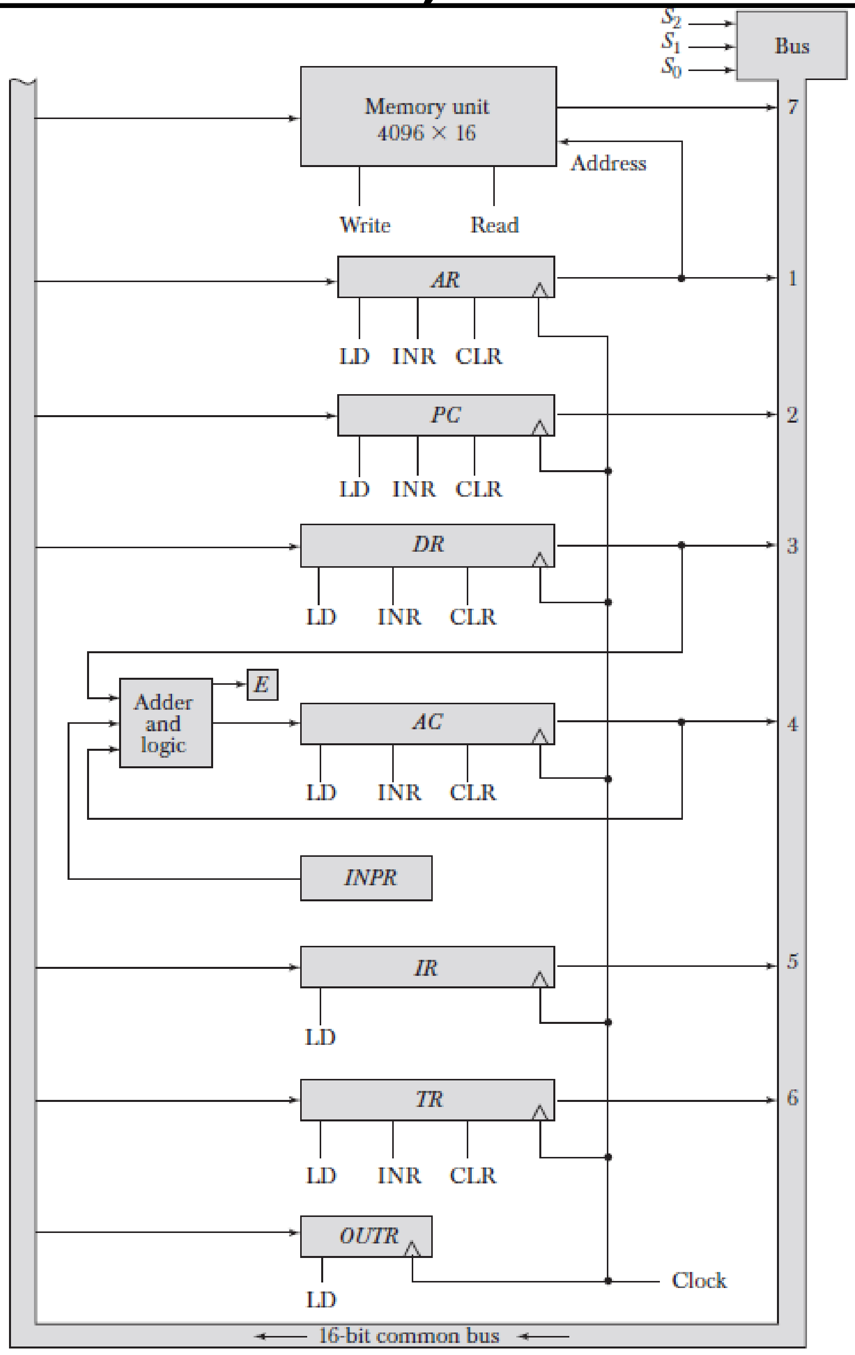
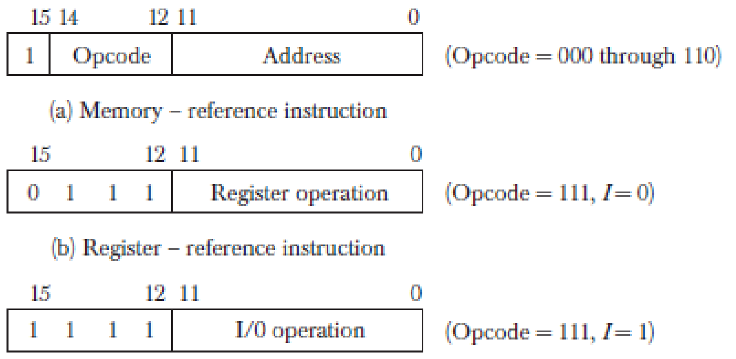


Figure 5-5 Basic computer instruction formats.



(a) Memory – reference instruction

(b) Register – reference instruction

(c) Input – output instruction

TABLE 5-2 Basic Computer Instructions

	Hexadecimal code		
Symbol	$I = 0$	$I = 1$	Description
AND	0xxx	8xxx	AND memory word to *AC*
ADD	1xxx	9xxx	Add memory word to *AC*
LDA	2xxx	Axxx	Load memory word to *AC*
STA	3xxx	Bxxx	Store content of *AC* in memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero
CLA	7800		Clear *AC*
CLE	7400		Clear *E*
CMA	7200		Complement *AC*
CME	7100		Complement *E*
CIR	7080		Circulate right *AC* and *E*
CIL	7040		Circulate left *AC* and *E*
INC	7020		Increment *AC*
SPA	7010		Skip next instruction if *AC* positive
SNA	7008		Skip next instruction if *AC* negative
SZA	7004		Skip next instruction if *AC* zero
SZE	7002		Skip next instruction if *E* is 0
HLT	7001		Halt computer
INP	F800		Input character to *AC*
OUT	F400		Output character from *AC*
SKI	F200		Skip on input flag
SKO	F100		Skip on output flag
ION	F080		Interrupt on
IOF	F040		Interrupt off

TIMING AND CONTROL

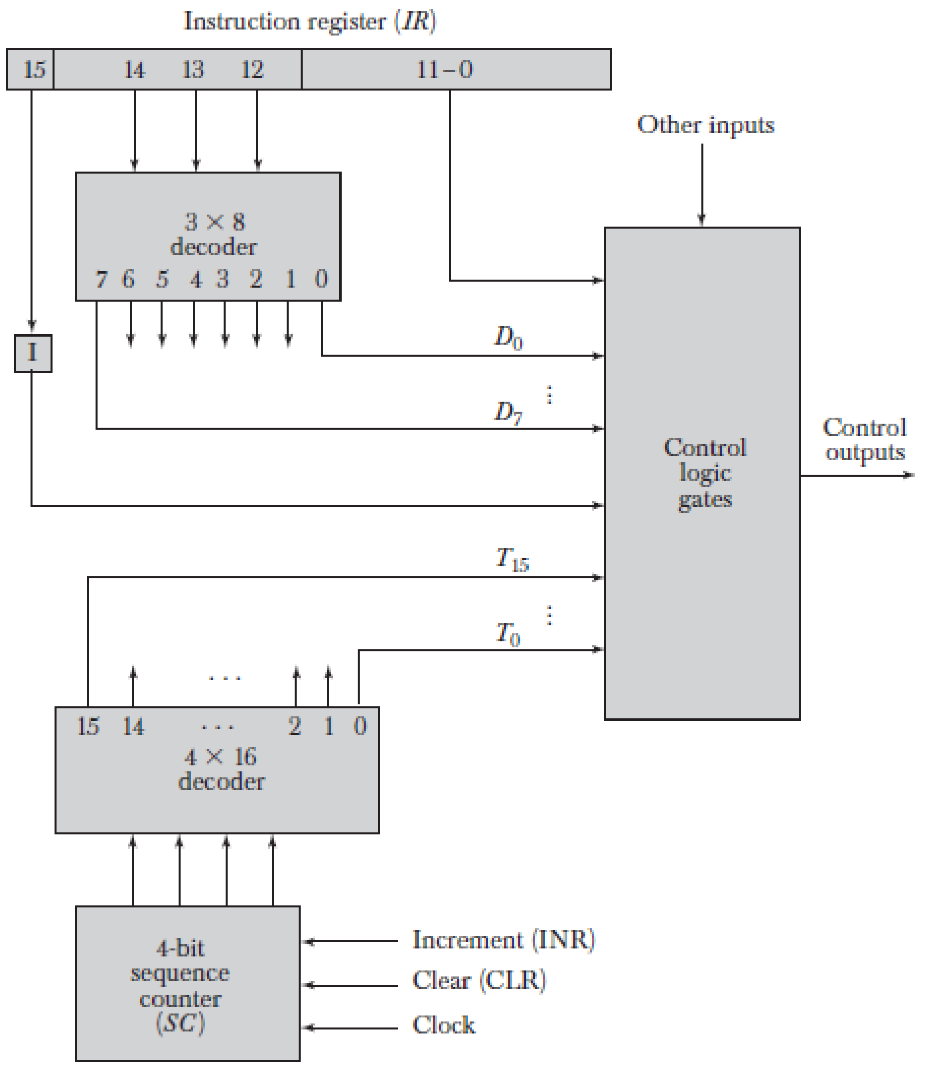


Figure 5-6 Control unit of basic computer.

$$D_3T_4: \quad SC \leftarrow 0$$

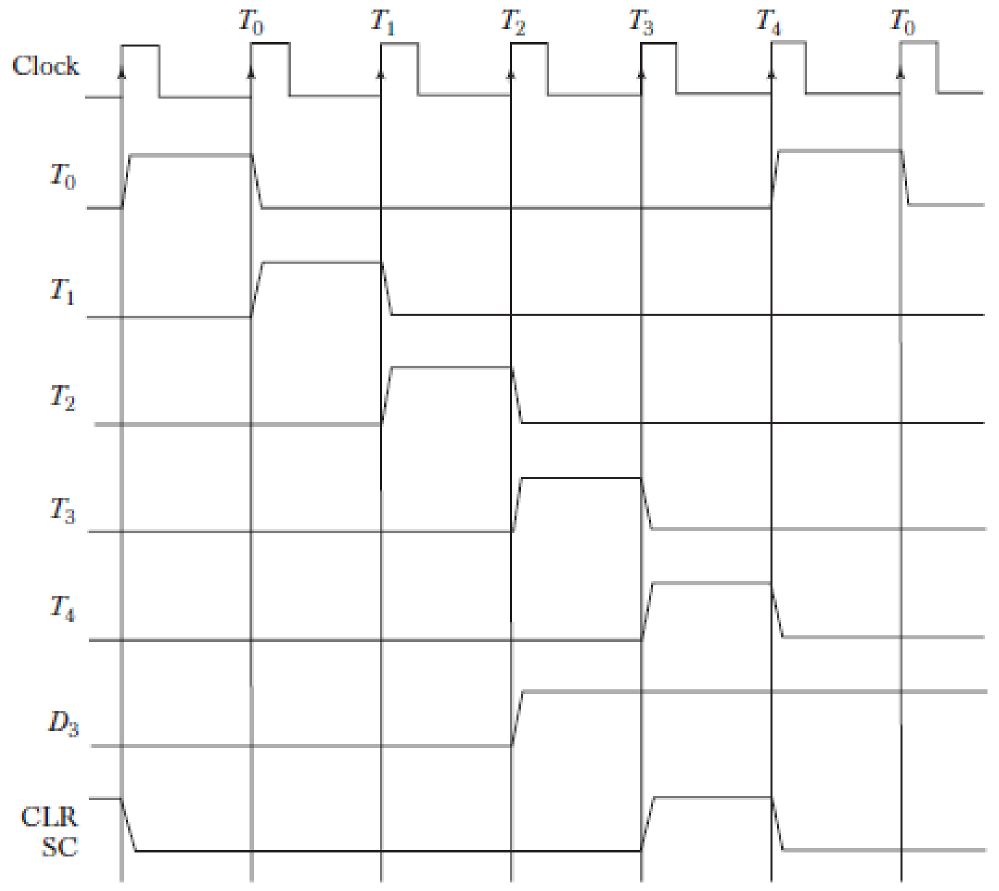
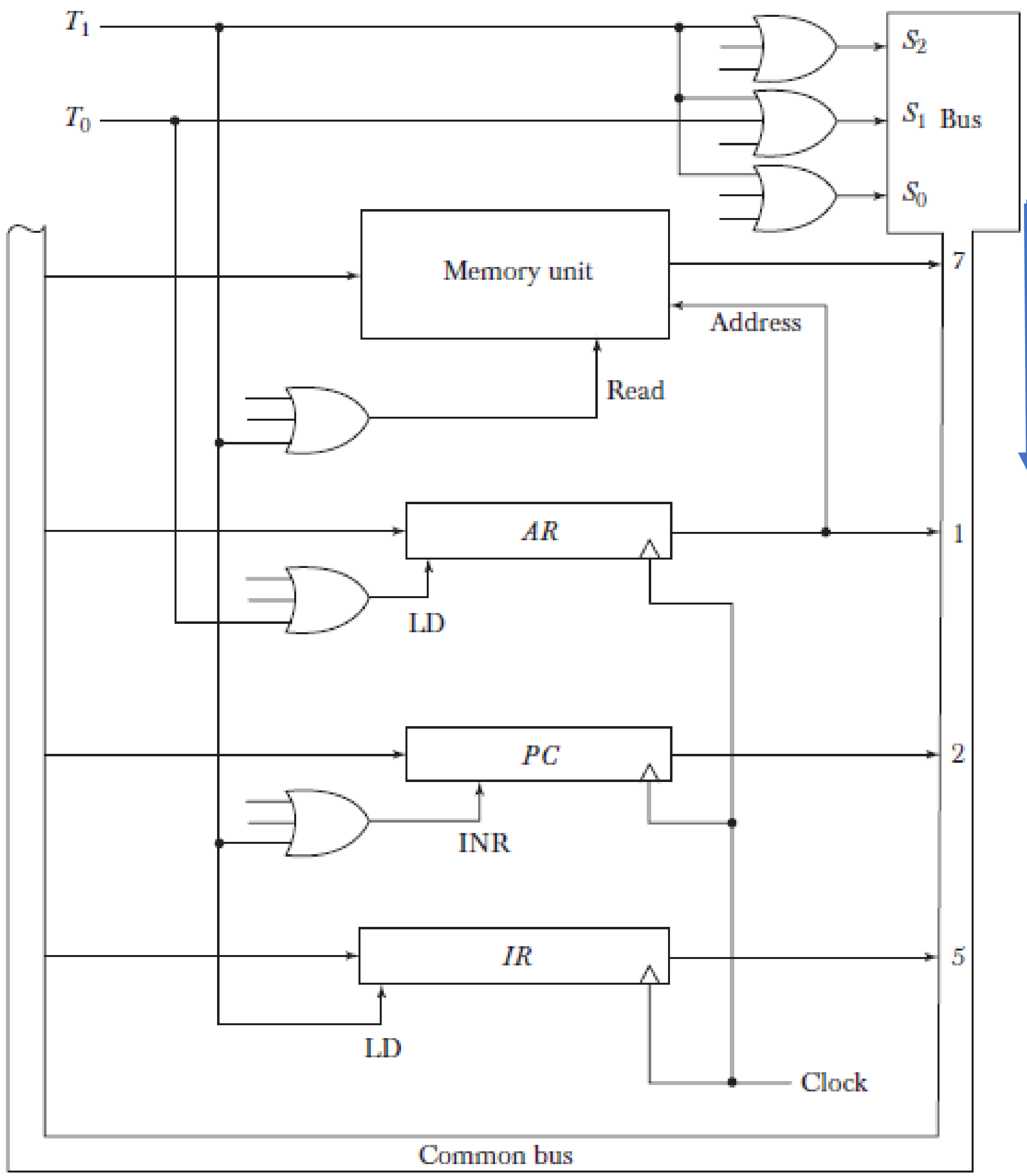


Figure 5-7 Example of control timing signals.

TIMING AND CONTROL

Figure 5-8 Register transfers for the fetch phase.



1. Fetch an instruction from memory.
2. Decode the instruction.
3. Read the effective address from memory if the instruction has an indirect address.
4. Execute the instruction.

T_0: $AR \leftarrow PC$

T_1: $IR \leftarrow M[AR],\ PC \leftarrow PC + 1$

T_2: $D_0, \ldots, D_7 \leftarrow$ Decode $IR(12\text{–}14)$, $AR \leftarrow IR(0-11)$, $I \leftarrow IR(15)$

1. Place the content of *PC* onto the bus by making the bus selection inputs $S_2S_1S_0$ equal to 010.
2. Transfer the content of the bus to *AR* by enabling the LD input of *AR*.

1. Enable the read input of memory.
2. Place the content of memory onto the bus by making $S_2S_1S_0 = 111$.
3. Transfer the content of the bus to *IR* by enabling the LD input of *IR*.
4. Increment *PC* by enabling the INR input of *PC*.

INSTRUCTION CYCLE

$D_7' I T_3$:	$AR \leftarrow M[AR]$
$D_7' I T_3$:	Nothing
$D_7 I' T_3$:	Execute a register-reference instruction
$D_7 I T_3$:	Execute an input–output instruction

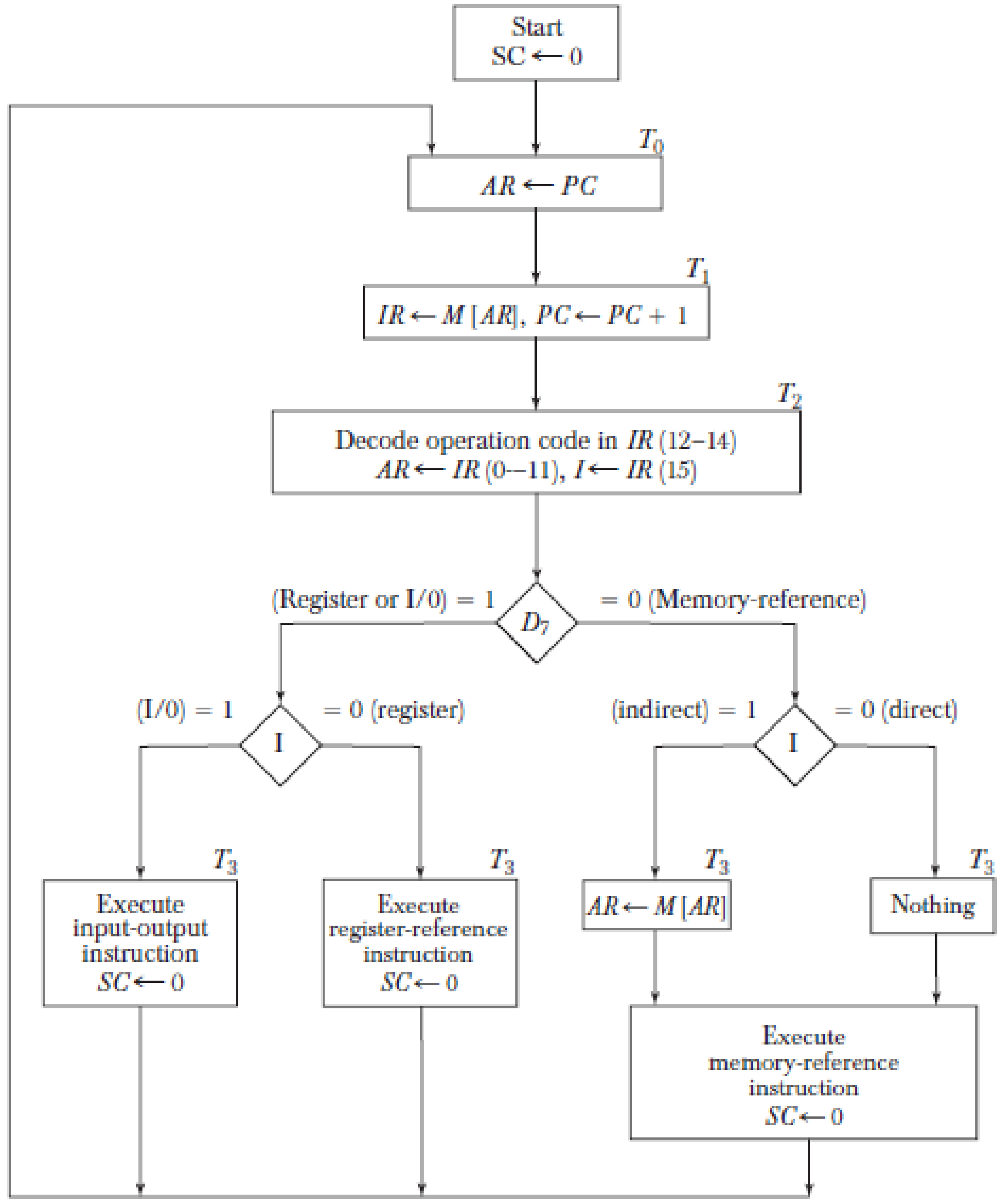
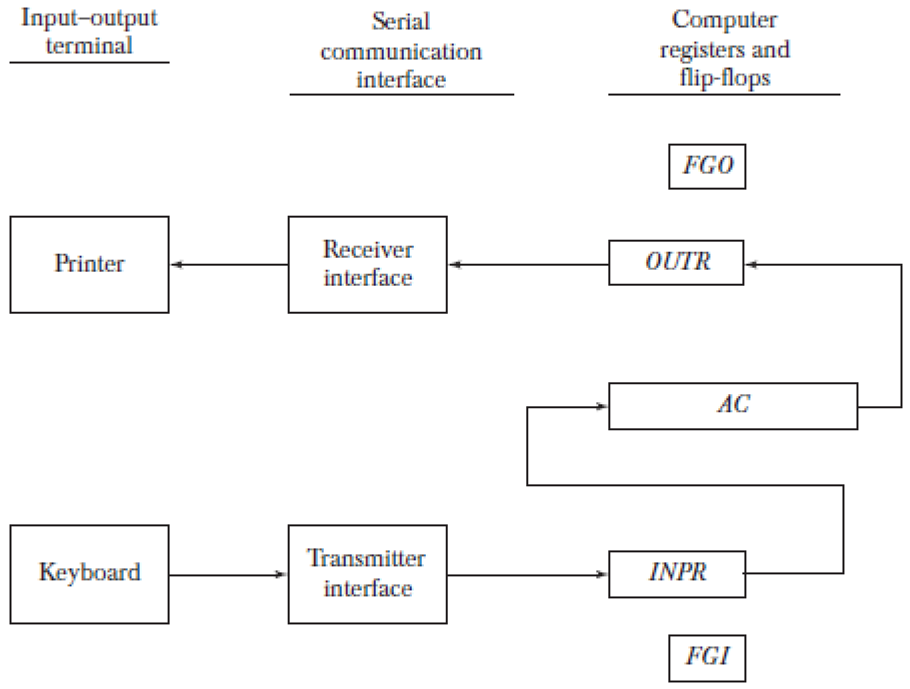


Figure 5-9 Flowchart for instruction cycle (initial configuration).

INPUT-OUTPUT CONFIGURATION, AND INTERRUPT CYCLE

Figure 5-12 Input–output configuration.



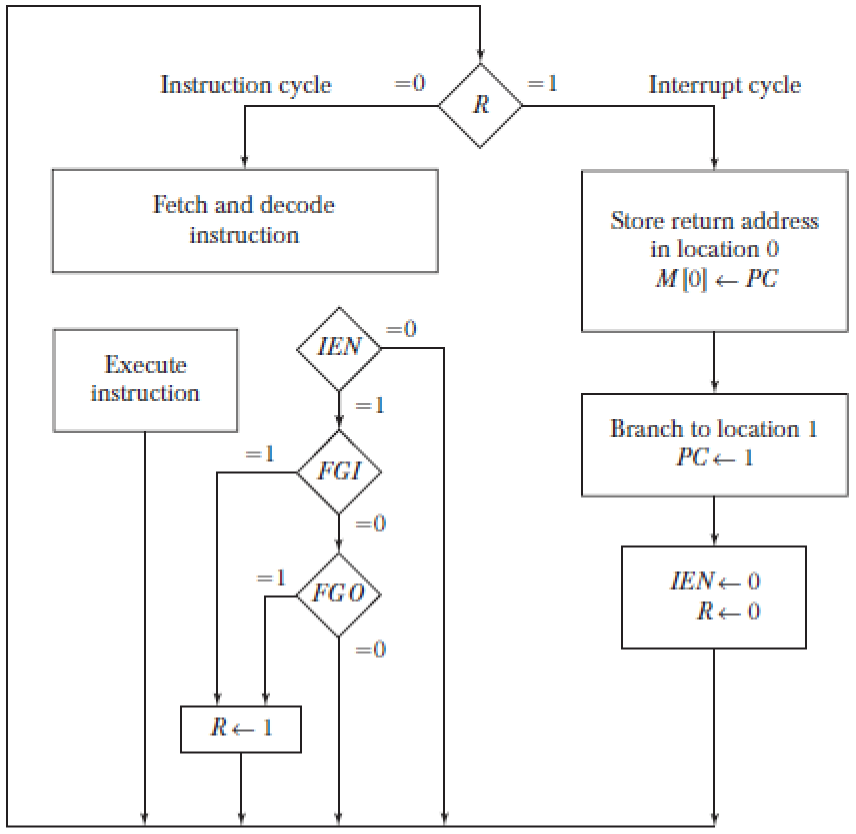
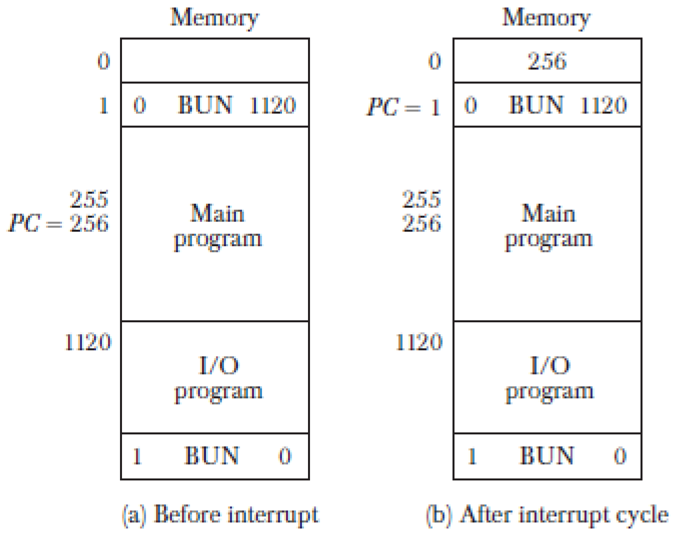


Figure 5-13 Flowchart for interrupt cycle.

Figure 5-14 Demonstration of the interrupt cycle.



LEVELS OF PROGRAMMING LANGUAGES: MACHINE LANGUAGE, ASSEMBLY LANGUAGE, HIGH LEVEL LANGUAGE.

TABLE 6-2 Binary Program to Add Two Numbers

Location	Instruction code
0	0010 0000 0000 0100
1	0001 0000 0000 0101
10	0011 0000 0000 0110
11	0111 0000 0000 0001
100	0000 0000 0101 0011
101	1111 1111 1110 1001
110	0000 0000 0000 0000

TABLE 6-3 Hexadecimal Program to Add Two Numbers

Location	Instruction
000	2004
001	1005
002	3006
003	7001
004	0053
005	FFE9
006	0000

TABLE 6-4 Program with Symbolic Operation Codes

Location	Instruction	Comments
000	LDA 004	Load first operand into *AC*
001	ADD 005	Add second operand to *AC*
002	STA 006	Store sum in location 006
003	HLT	Halt computer
004	0053	First operand
005	FFE9	Second operand (negative)
006	0000	Store sum here

TABLE 6-5 Assembly Language Program to Add Two Numbers

```
        ORG 0       /Origin of program is location 0
        LDA A       /Load operand from location A
        ADD B       /Add operand from location B
        STA C       /Store sum in location C
        HLT         /Halt computer
A,      DEC 83      /Decimal operand
B,      DEC −23     /Decimal operand
C,      DEC 0       /Sum stored in location C
        END         /End of symbolic program
```

TABLE 6-6 Fortran Program to Add Two Numbers

```
INTEGER A, B, C
DATA A, 83    B, −23
C = A + B
END
```

Revision
Unit – 2
Chapter - 1

Central processing Unit: Introduction, general register organization, stack organization, instruction format, addressing modes. Overview of GPU, CPU vs GPU computing difference.

Introduction

- The part of the computer that performs the bulk of data-processing operations is called the central processing unit and is referred to as the CPU.
- The arithmetic logic unit (ALU) performs the required microoperations for executing the instructions. The control unit supervises the transfer of information among the registers and instructs the ALU as to which operation to perform.

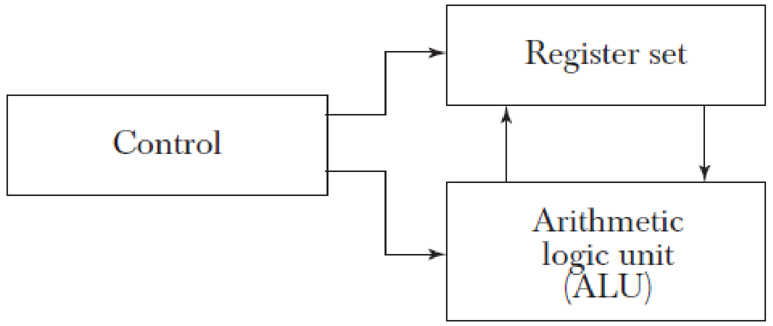
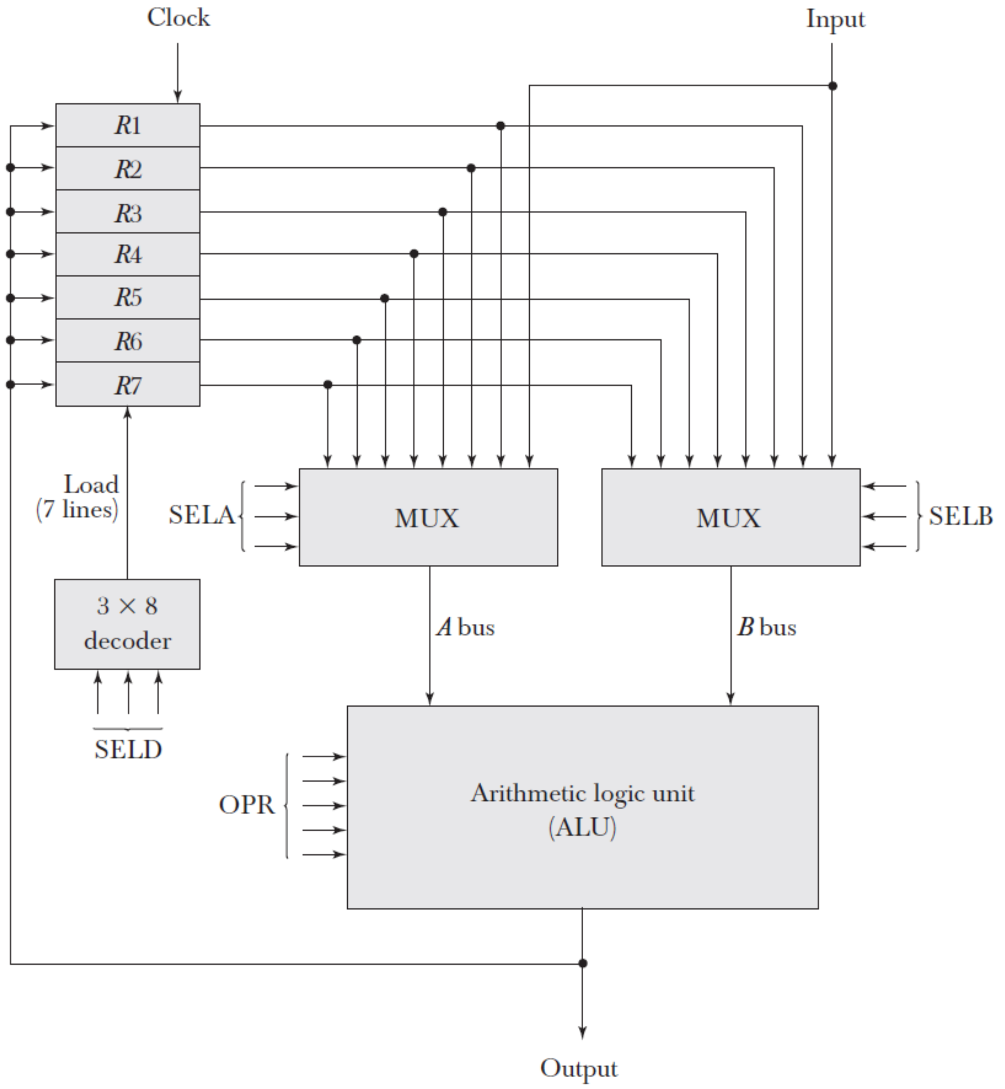


Figure 8-1 Major components of CPU.

General register organization



(a) Block diagram

3	3	3	5
SELA	SELB	SELD	OPR

(b) Control word

Figure 8-2 Register set with common ALU.

TABLE 8-2 Encoding of ALU Operations

OPR Select	Operation	Symbol
00000	Transfer *A*	TSFA
00001	Increment *A*	INCA
00010	Add *A* + *B*	ADD
00101	Subtract *A* − *B*	SUB
00110	Decrement *A*	DECA
01000	AND *A* and *B*	AND
01010	OR *A* and *B*	OR
01100	XOR *A* and *B*	XOR
01110	Complement *A*	COMA
10000	Shift right *A*	SHRA
11000	Shift left A	SHLA

Example – If the following microoperation is to be implemented, identify the corresponding control word.

$$R1 \leftarrow R2 - R3$$

Field:	SELA	SELB	SELD	OPR
Symbol:	R2	R3	R1	SUB
Control word:	010	011	001	00101

Stack organization

Register Stack

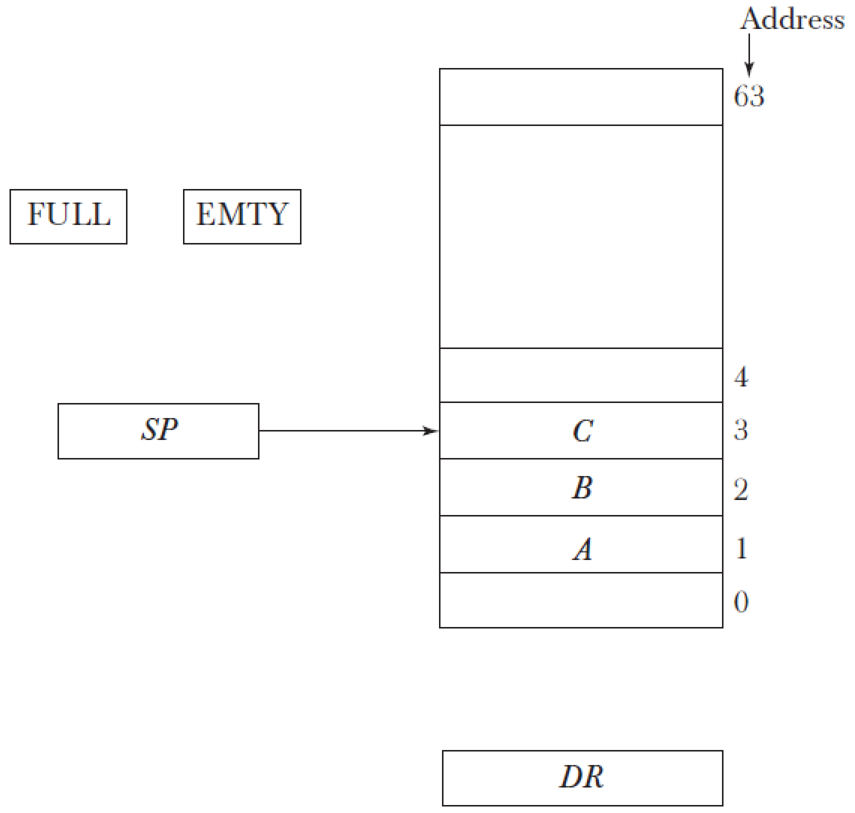


Figure 8-3 Block diagram of a 64-word stack.

$SP \leftarrow SP + 1$	Increment stack pointer
$M[SP] \leftarrow DR$	Write item on top of the stack
If $(SP = 0)$ then $(FULL \leftarrow 1)$	Check if stack is full
$EMTY \leftarrow 0$	Mark the stack not empty

$DR \leftarrow M[SP]$	Read item from the top of stack
$SP \leftarrow SP - 1$	Decrement stack pointer
If $(SP = 0)$ then $(EMTY \leftarrow 1)$	Check if stack is empty
$FULL \leftarrow 0$	Mark the stack not full

Stack organization

Memory Stack

We assume that the items in the stack communicate with a data register *DR*. A new item is inserted with the push operation as follows:

$$SP \leftarrow SP - 1$$
$$M[SP] \leftarrow DR$$

The stack pointer is decremented so that it points at the address of the next word. A memory write operation inserts the word from *DR* into the top of the stack. A new item is deleted with a pop operation as follows:

$$DR \leftarrow M[SP]$$
$$SP \leftarrow SP + 1$$

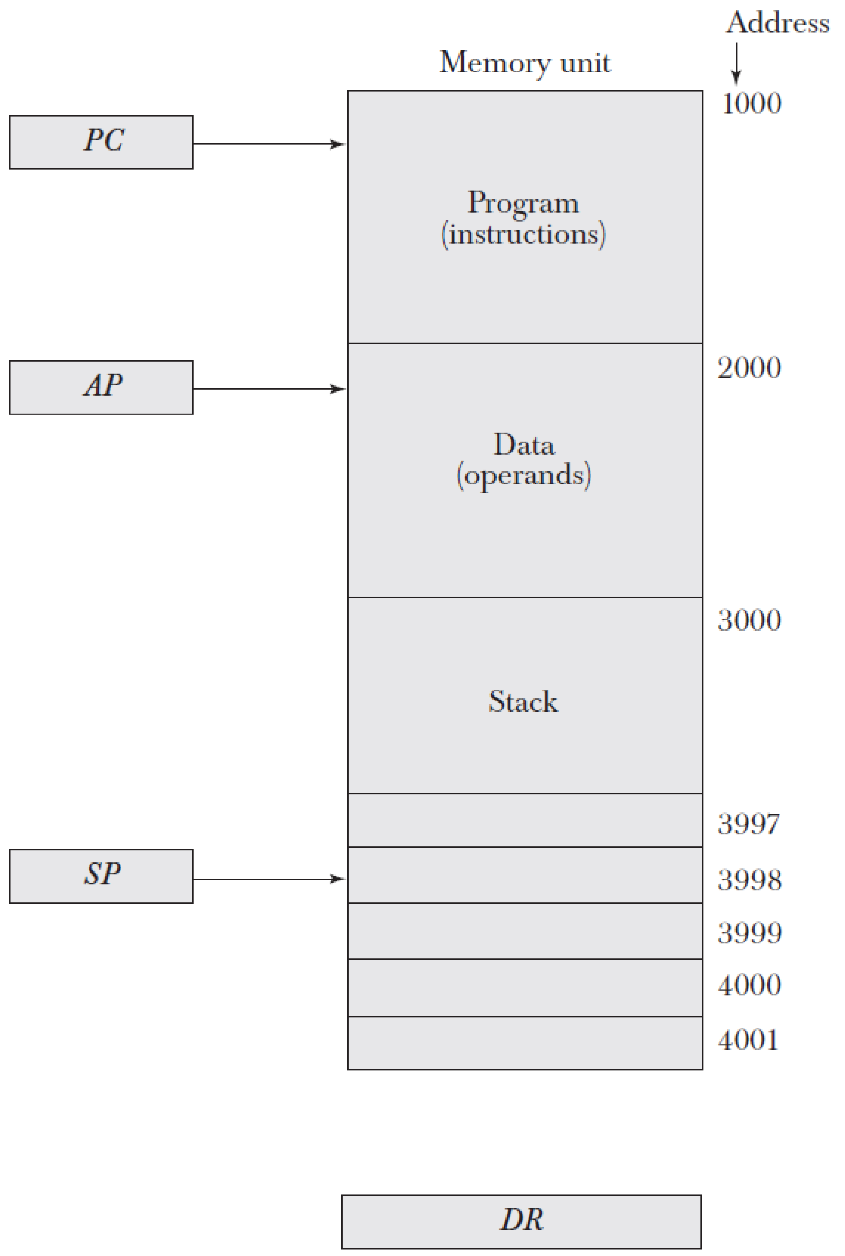


Figure 8-4 Computer memory with program, data, and stack segments.

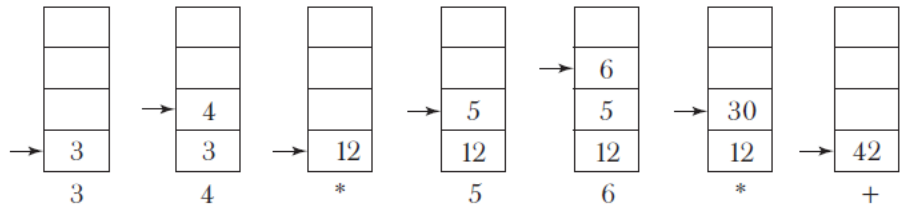
The following numerical example may clarify this procedure. Consider the arithmetic expression

$$(3 * 4) + (5 * 6)$$

In reverse Polish notation, it is expressed as

$$34 * 56 * +$$

Figure 8-5 Stack operations to evaluate 3 • 4 + 5 • 6.



Instruction format

- The most common fields found in instruction formats are:
  - 1. An operation code field that specifies the operation to be performed.
  - 2. An address field that designates a memory address or a processor register.
  - 3. A mode field that specifies the way the operand or the effective address is determined.
- The number of address fields in the instruction format of a computer depends on the internal organization of its registers. Most computers fall into one of three types of CPU organizations:
  - 1. Single accumulator organization.
  - 2. General register organization.
  - 3. Stack organization.

Code for computation of **X = (A+B)*(C+D)** using 3 Address, 2 Address, 1 Address amd 0 Address Instructions are as follows

3 Address Instructions

```
ADD     R1, A, B     R1 ← M[A]+M[B]
ADD     R2, C, D     R2 ← M[C]+M[D]
MUL     X, R1, R2    M[X] ← R1*R2
```

2 Address Instructions

```
MOV     R1, A        R1 ← M[A]
ADD     R1, B        R1 ← R1 + M[B]
MOV     R2, C        R2 ← M[C]
ADD     R2, D        R2 ← R2 + M[D]
MUL     R1, R2       R1 ← R1*R2
MOV     X, R1        M[X] ← R1
```

3 Address Instructions

```
LOAD    A        AC ← M[A]
ADD     B        AC ← AC + M[B]
STORE   T        M[T] ← AC
LOAD    C        AC ← M[C]
ADD     D        AC ← AC + M[D]
MUL     T        AC ← AC * M[T]
STORE   X        M[X] ← AC
```

0 Address Instructions

```
PUSH    A        TOS ← A
PUSH    B        TOS ← B
ADD              TOS ← (A + B)
PUSH    C        TOS ← C
PUSH    D        TOS ← D
ADD              TOS ← (C + D)
MUL              TOS ← (C + D) * (A + B)
POP     X        M[X] ← TOS
```

Addressing modes

1. Implied Mode
2. Immediate Mode
3. Register Mode
4. Register Indirect Mode
5. Autoincrement or autodecrement
6. Direct Address Mode
7. Indirect Address Mode
8. Relative Address Mode
9. Indexed Address Mode
10. Base Register Addressing Mode

Overview of GPU, CPU vs GPU computing difference

	Central Processing Unit	Graphical Processing Unit
Overview	• Brain of a computer and is designed for general-purpose computing tasks. • Has a few powerful cores optimized for sequential processing. • They are excellent for tasks that require high single-threaded performance, such as operating systems and general applications.	• It is specialized for parallel processing and is designed primarily for rendering graphics. • Consist of thousands of small, efficient cores optimized for parallelism. • They excel at tasks that can be parallelized, making them valuable for scientific simulations, deep learning, and graphics rendering.
Architecture	• A few powerful cores with large caches. • Execution units are optimized for sequential instruction execution. • High single-threaded performance with complex instruction sets.	• Have thousands of simpler cores, organized into streaming multiprocessors (SMs). • Execution units are optimized for parallel execution. • Lower clock speeds per core but high parallel throughput.
Parallelism	• Have 2-64 cores, suitable for multi-threaded tasks. • Limited parallelism compared to GPUs. • Performance improvements in multi-threaded applications are modest.	• Have thousands of cores, designed for massively parallel tasks. • Ideal for tasks involving large data sets or complex calculations. • Significant performance boost in parallel applications.

Overview of GPU, CPU vs GPU computing difference

	Central Processing Unit	Graphical Processing Unit
Memory Hierarchy	• Have complex memory hierarchies with caches (L1, L2, L3) and main memory. • Latency-optimized for fast access to a small amount of data. • Suitable for tasks with small memory footprints and irregular access patterns.	• GPUs have a simpler memory hierarchy with global memory, shared memory, and local memory. • Designed for high-throughput data access and transfer. • Ideal for tasks with large data sets and regular access patterns.
Applications	• CPUs are suitable for tasks that require strong single-threaded performance. • Examples include web browsing, office applications, and general-purpose software.	• GPUs excel in tasks that can be parallelized, such as scientific simulations, 3D rendering, video processing, and deep learning. • They are widely used in artificial intelligence and machine learning.

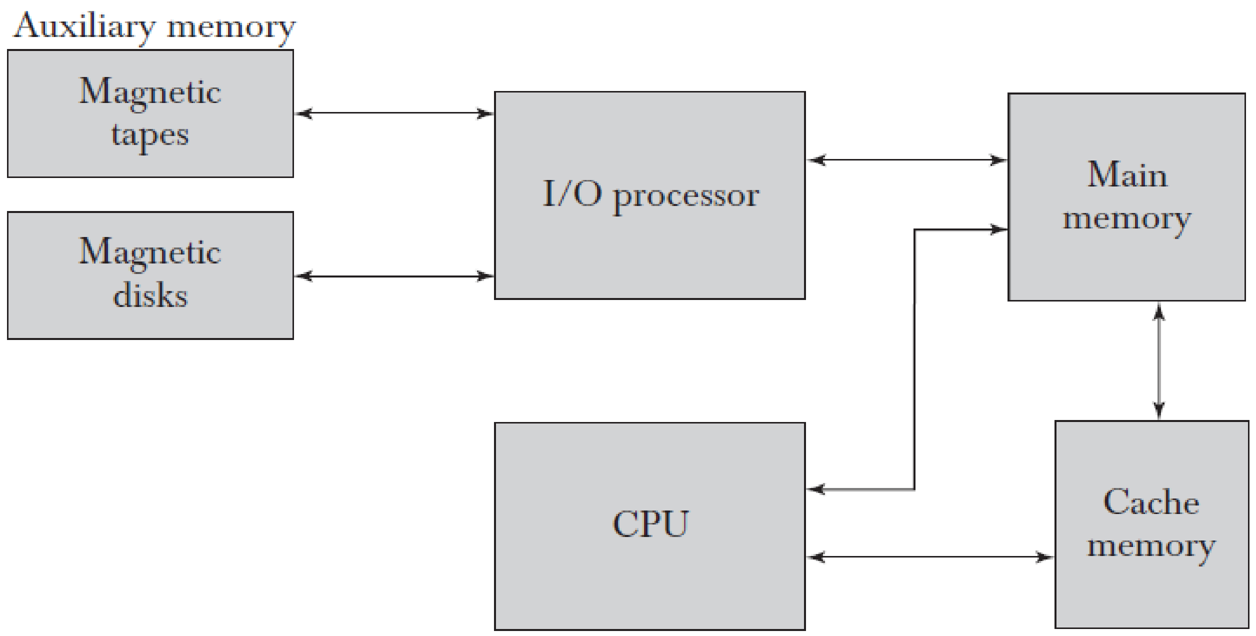
Revision
Unit – 2
Chapter - 2

Memory Hierarchy: Introduction, basics of cache, measuring and improving of cache performance, cache memory: associative mapping, direct mapping, set-associative mapping, cache writing and initialization, virtual memory, common framework for memory hierarchies. Case study of PIV and AMD opteron memory hierarchies.

Introduction

- The memory unit is an essential component in any digital computer since it is needed for storing programs and data.
- The memory unit that communicates directly with the CPU is called the *main memory.* Devices that provide backup storage are called *auxiliary memory.*

Figure 12-1 Memory hierarchy in a computer system.



Basics of cache, measuring and improving of cache performance

- A special very-high-speed memory called a *cache* is sometimes used to increase the speed of processing by making current programs and data available to the CPU at a rapid rate.
- Analysis of a large number of typical programs has shown that the references to memory at any given interval of time tend to be confined within a few localized areas in memory. This phenomenon is known as the property of *locality of reference*.
- The performance of cache memory is frequently measured in terms of a quantity called *hit ratio*.
- When the CPU refers to memory and finds the word in cache, it is said to produce a *hit*.
- If the word is not found in cache, it is in main memory and it counts as a *miss*.
- The ratio of the number of hits divided by the total CPU references to memory (hits plus misses) is the hit ratio. The

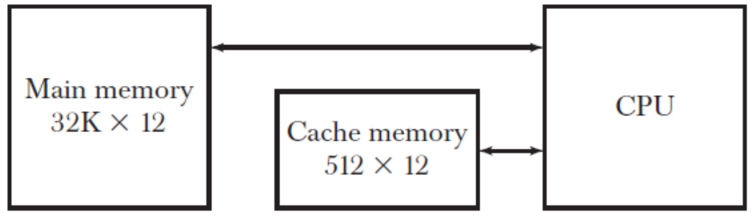
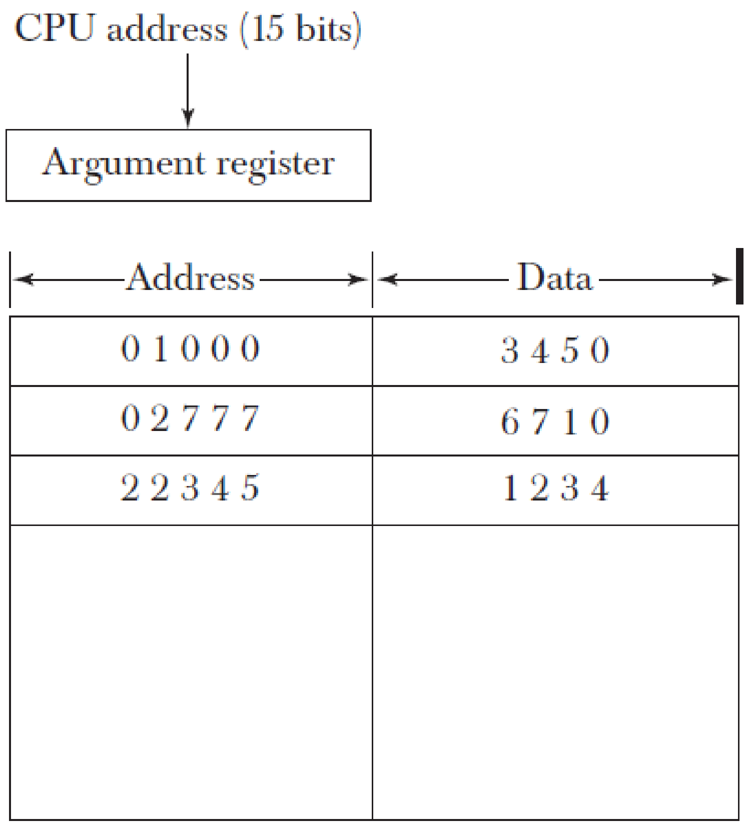


Figure 12-10 Example of cache memory.

Cache memory: Associative Mapping

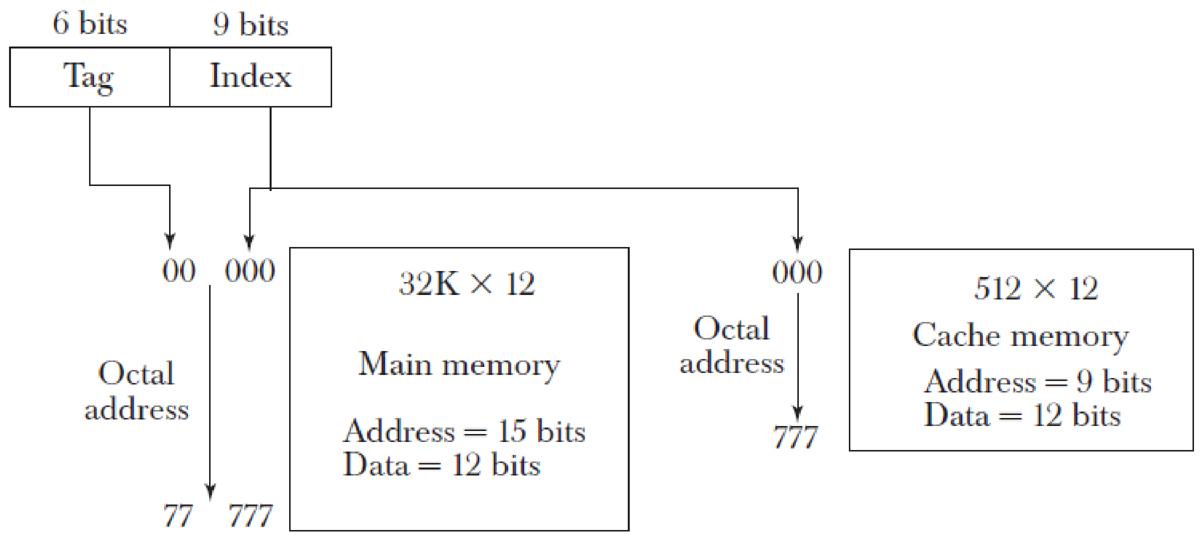
Figure 12-11 Associative mapping cache (all numbers in octal).

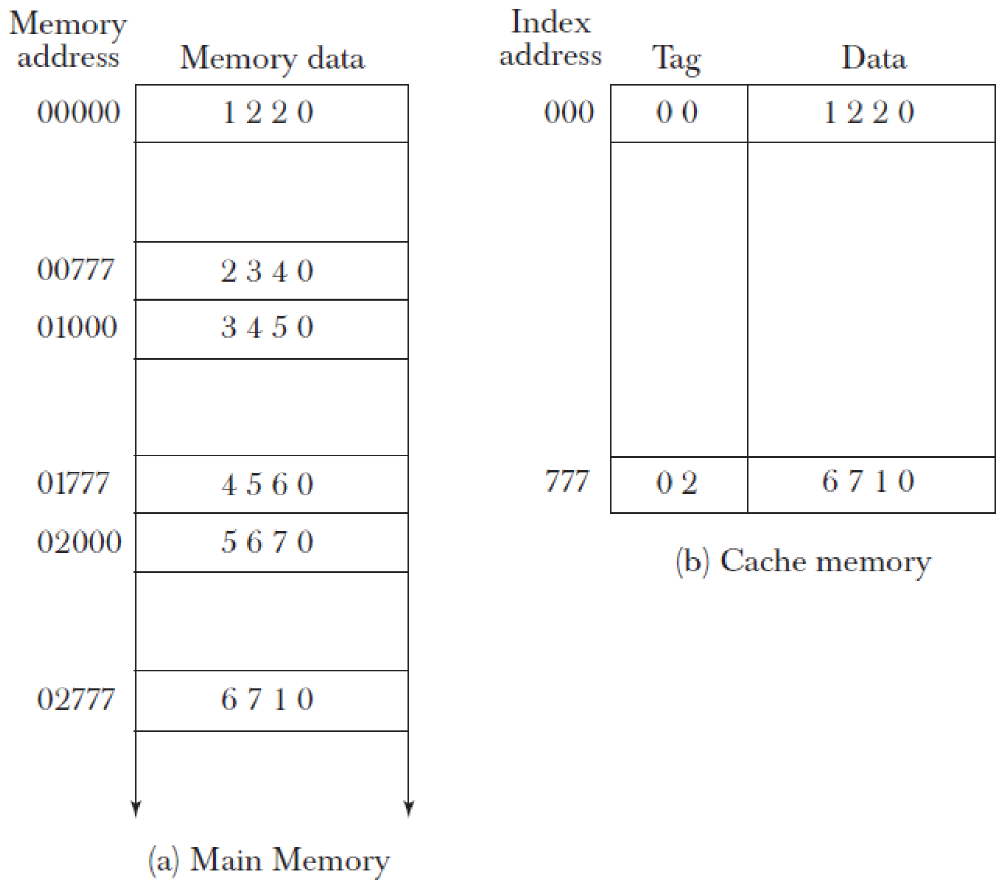


Address	Data
0 1 0 0 0	3 4 5 0
0 2 7 7 7	6 7 1 0
2 2 3 4 5	1 2 3 4

Cache memory: Direct mapping

Figure 12-12 Addressing relationships between main and cache memories.





Memory address	Memory data
00000	1 2 2 0
00777	2 3 4 0
01000	3 4 5 0
01777	4 5 6 0
02000	5 6 7 0
02777	6 7 1 0

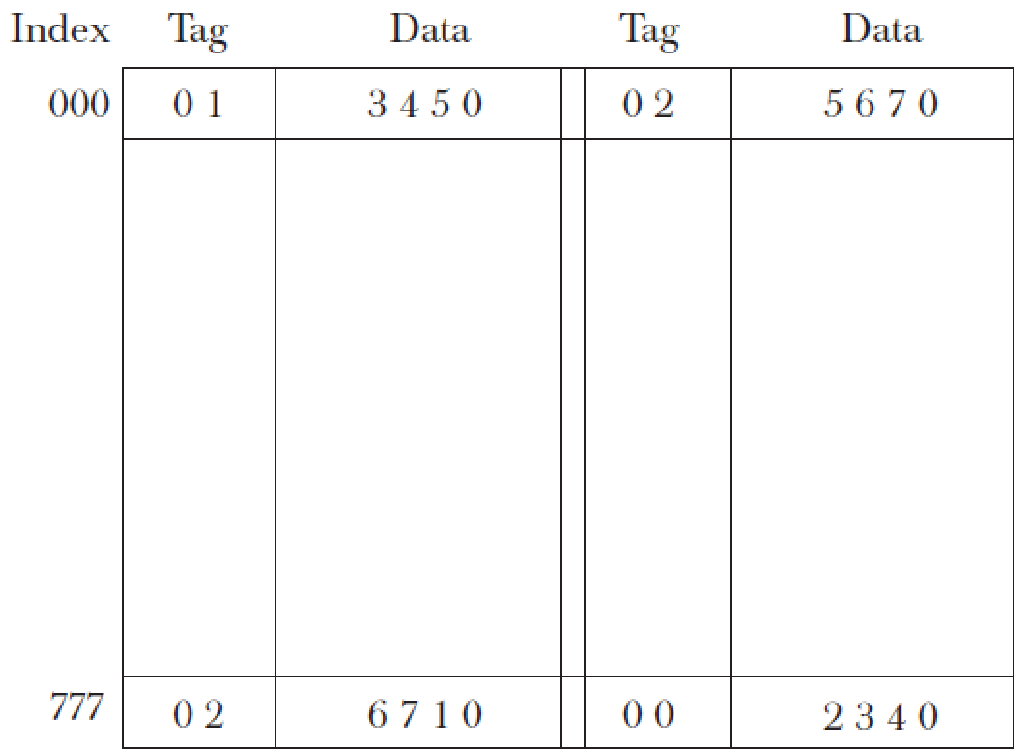
(a) Main Memory

Index address	Tag	Data
000	0 0	1 2 2 0
777	0 2	6 7 1 0

(b) Cache memory

Figure 12-13 Direct mapping cache organization.

Cache memory: Set-associative mapping



Index	Tag	Data	Tag	Data
000	0 1	3 4 5 0	0 2	5 6 7 0
777	0 2	6 7 1 0	0 0	2 3 4 0

Figure 12-15 Two-way set-associative mapping cache.

Cache writing and initialization

- Writing into the cache can be done with the help of two mechanisms
    - Write through
    - Write back

- The cache is initialized when power is applied to the computer or when the main memory is loaded with a complete set of programs from auxiliary memory. After initialization the cache is considered to be empty, but in effect it contains some nonvalid data.

Virtual memory, common framework for memory hierarchies.

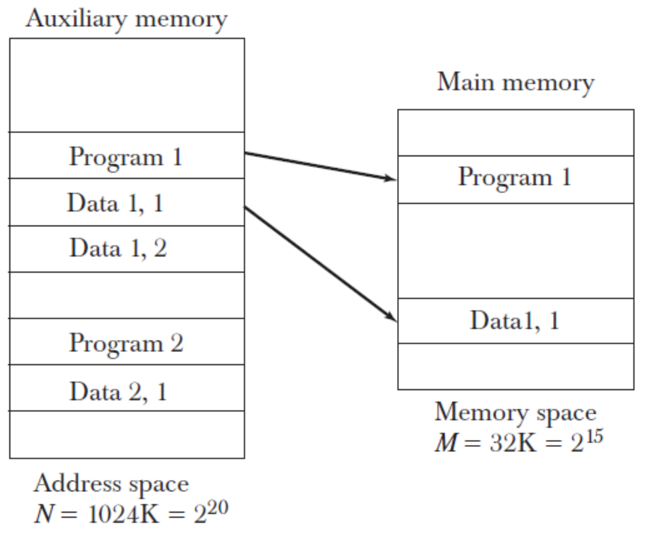
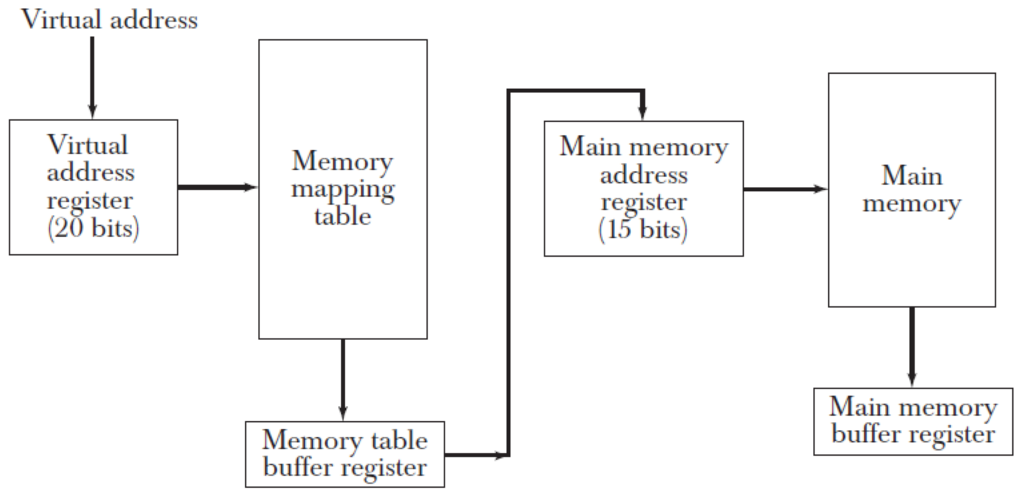


Figure 12-16 Relation between address and memory space in a virtual memory system.

Figure 12-17 Memory table for mapping a virtual address.



Virtual address
Virtual address register (20 bits)
Memory mapping table
Memory table buffer register
Main memory address register (15 bits)
Main memory
Main memory buffer register

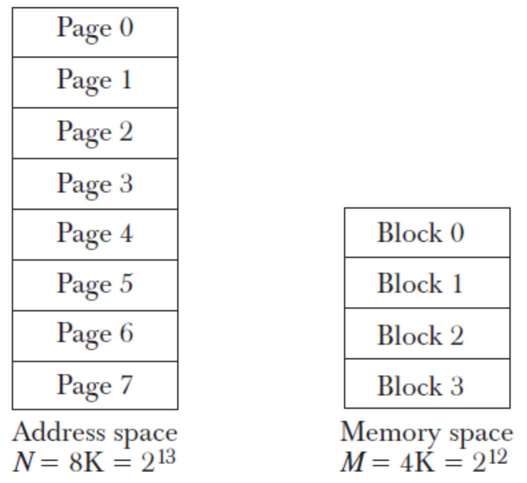
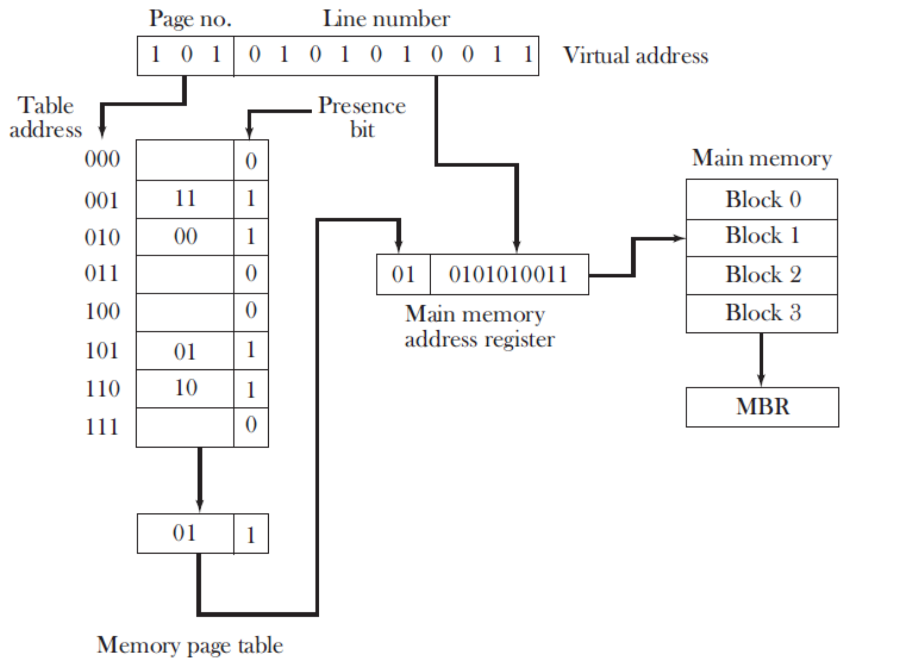


Figure 12-18 Address space and memory space split into groups of 1K words.

Figure 12-19 Memory table in a paged system.



Case study of PIV and AMD opteron memory hierarchies

Intel Pentium IV (PIV):

1. Registers: PIV had a set of general-purpose and specialized registers for rapid data access.

2. Level 1 Cache (L1): The PIV had separate instruction and data caches, each with 8 KB.

3. Level 2 Cache (L2): The PIV included a larger unified L2 cache, which varied in size between 256 KB and 2 MB, depending on the specific model.

4. Main Memory (RAM): Data not found in the caches was retrieved from RAM, which was typically DDR SDRAM at the time.

AMD Opteron:

1. Registers: Like the PIV, the AMD Opteron had a set of registers for fast data access.

2. Level 1 Cache (L1): The L1 cache was divided into two parts: a 64 KB instruction cache and a 64 KB data cache.

3. Level 2 Cache (L2): The Opteron featured a unified L2 cache ranging from 512 KB to 2 MB, depending on the specific model.

4. Main Memory (RAM): Main memory was typically DDR SDRAM or later variants, similar to the PIV.