Micro-Operation

- Computer system micro-operations are of four types:
  - Register transfer micro-operations
  - Arithmetic micro-operations
  - Logic micro-operations
  - Shift micro-operations

Arithmetic Micro-operations

Summary of Typical Arithmetic Micro-Operations

$R3 \leftarrow R1 + R2$	Contents of R1 plus R2 transferred to R3
$R3 \leftarrow R1 - R2$	Contents of R1 minus R2 transferred to R3
$R2 \leftarrow R2'$	Complement the contents of R2
$R2 \leftarrow R2' + 1$	2's complement the contents of R2 (negate)
$R3 \leftarrow R1 + R2' + 1$	subtraction
$R1 \leftarrow R1 + 1$	Increment
$R1 \leftarrow R1 - 1$	Decrement

Logical Micro-operations

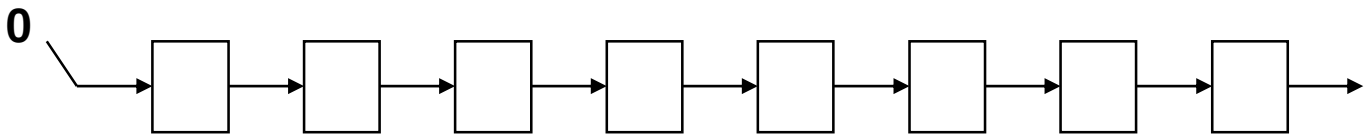
Boolean function	Microoperation	Name
$F_0 = 0$	$F \leftarrow 0$	Clear
$F_1 = xy$	$F \leftarrow A \wedge B$	AND
$F_2 = xy'$	$F \leftarrow A \wedge \overline{B}$	
$F_3 = x$	$F \leftarrow A$	Transfer A
$F_4 = x'y$	$F \leftarrow \overline{A} \wedge B$	
$F_5 = y$	$F \leftarrow B$	Transfer B
$F_6 = x \oplus y$	$F \leftarrow A \oplus B$	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

Shift Micro-operations

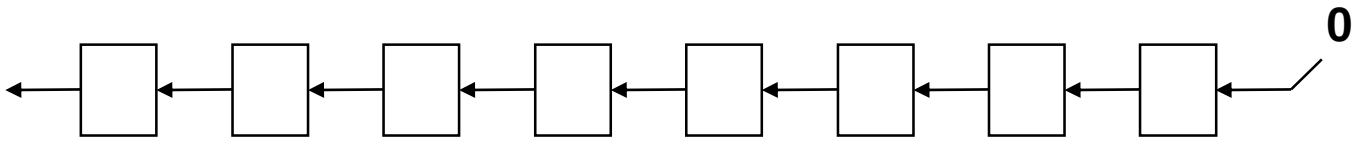
- R ← shl R — Shift-left register R
- R ←shr R — Shift-right register R
- R ←cil R — Circular shift-left register R
- R ← cir R — Circular shift-right register R
- R ← ashl R — Arithmetic shift-left R
- R ← ashr R — Arithmetic shift-right R

LOGICAL SHIFT

- In a logical shift the serial input to the shift is a 0.
- A right logical shift operation:



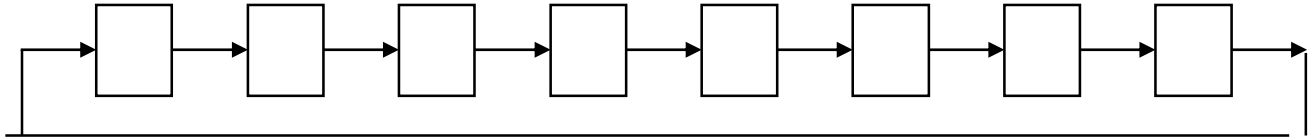
- A left logical shift operation:



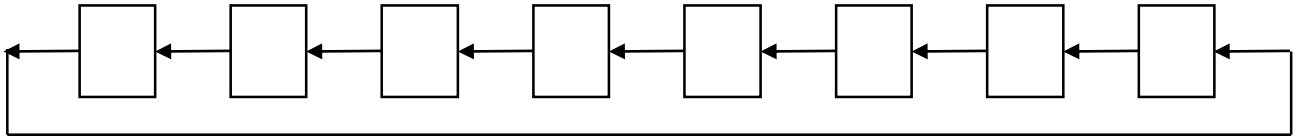
- In a Register Transfer Language, the following notation is used
  - *shl* for a logical shift left
  - *shr* for a logical shift right

CIRCULAR SHIFT

- In a circular shift the serial input is the bit that is shifted out of the other end of the register.
- A right circular shift operation:



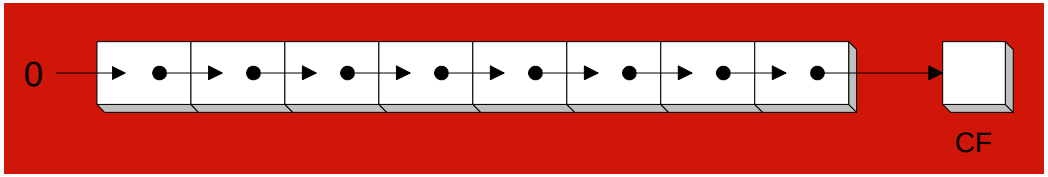
- A left circular shift operation:



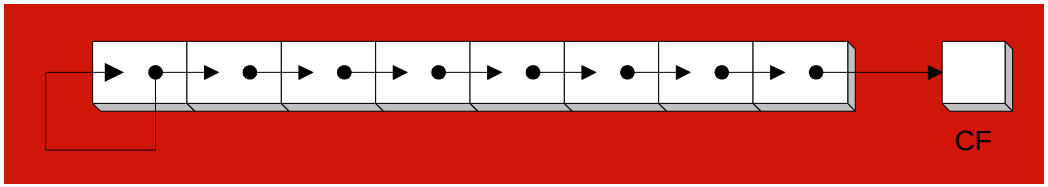
- In a RTL, the following notation is used
  - *cil* for a circular shift left
  - *cir* for a circular shift right

Logical versus Arithmetic Shift

- A logical shift fills the newly created bit position with zero:



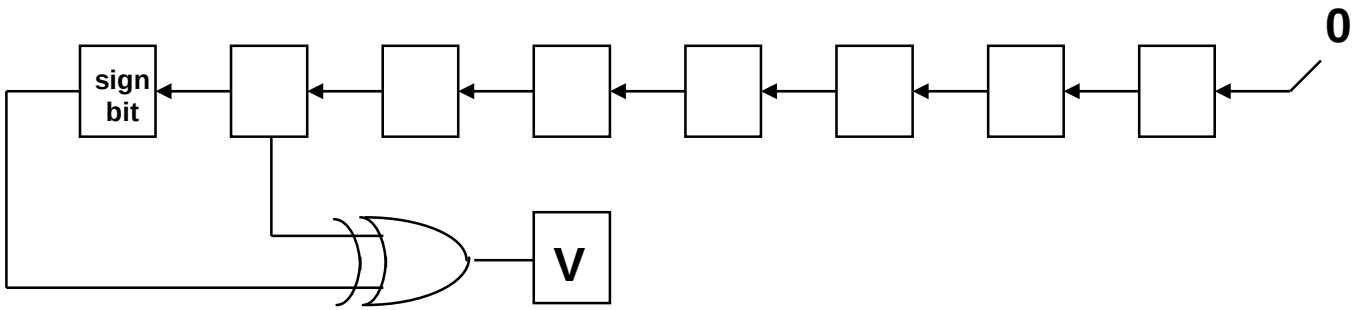
- An arithmetic shift fills the newly created bit position with a copy of the number's sign bit:



Arithmetic Right Shift Process

ARITHMETIC SHIFT

- An left arithmetic shift operation must be checked for the overflow



Before the shift, if the leftmost two bits differ, the shift will result in an overflow

- **In a RTL, the following notation is used**
  - ***ashl*** **for an arithmetic shift left**
  - ***ashr*** **for an arithmetic shift right**

APPLICATIONS OF LOGIC MICROOPERATIONS

- Logic micro-operations can be used to manipulate individual bits or a portions of a word in a register.
- Consider the data in a register A. In another register B is bit data that will be used to modify the contents of A.

SELECTIVE SET

- In a selective set operation, the bit pattern in B is used to *set* certain bits in A.

```
    1 1 0 0  A
    1 0 1 0  B        (A ← A + B)      OR Operation
    ---------
    1 1 1 0  A
```

SELECTIVE COMPLEMENT

- In a selective complement operation, the bit pattern in B is used to *complement* certain bits in A.

```
  1 1 0 0  A
  1 0 1 0  B        (A ← A ⊕ B)     XOR Operation
-----------
  0 1 1 0  A
```

SELECTIVE CLEAR

- In a selective clear operation, the bit pattern in B is used to *clear* certain bits in A.

$$\begin{array}{ll} 1\ 1\ 0\ 0 & A \\ 1\ 0\ 1\ 0 & B \\ \hline 0\ 1\ 0\ 0 & A \end{array} \qquad (A \leftarrow A \cdot B')$$

MASK OPERATION

- The mask operation is similar to selective-clear operation except that the bits of A are cleared only where there are corresponding 0's in B.

```
    1 1 0 0  A
    1 0 1 0  B        (A ← A · B)      AND Operation
  ---------------
    1 0 0 0  A
```

INSERT OPERATION

- An insert operation inserts a new value into a group of bits. This is done by first masking the bits and then ORing them with the required value.
- Example: A register contains eight bits 0110 1010. To replace the four leftmost bits by the value 1001, we first mask the four unwanted bits.

```
  0110 1010          A before
  0000 1111          B (mask)
  ---------
  0000 1010          A after masking  AND Operation
```

And then insert the new value

```
  0000 1010          A before
  1001 0000          B (insert)
  ---------
  1001 1010          A after insertion OR Operation
```

Question

- Register A holds the 8-bit binary 11011001. Determine the B operation and the logic microoperation to be performed in order to change the value in A to : 01101101, 11111101

CENTRAL PROCESSING UNIT

- Introduction
- General Register Organization
- Stack Organization
- Instruction Formats
- Addressing Modes
- Data Transfer and Manipulation
- Program Control
- Reduced Instruction Set Computer (RISC)

Components of CPU

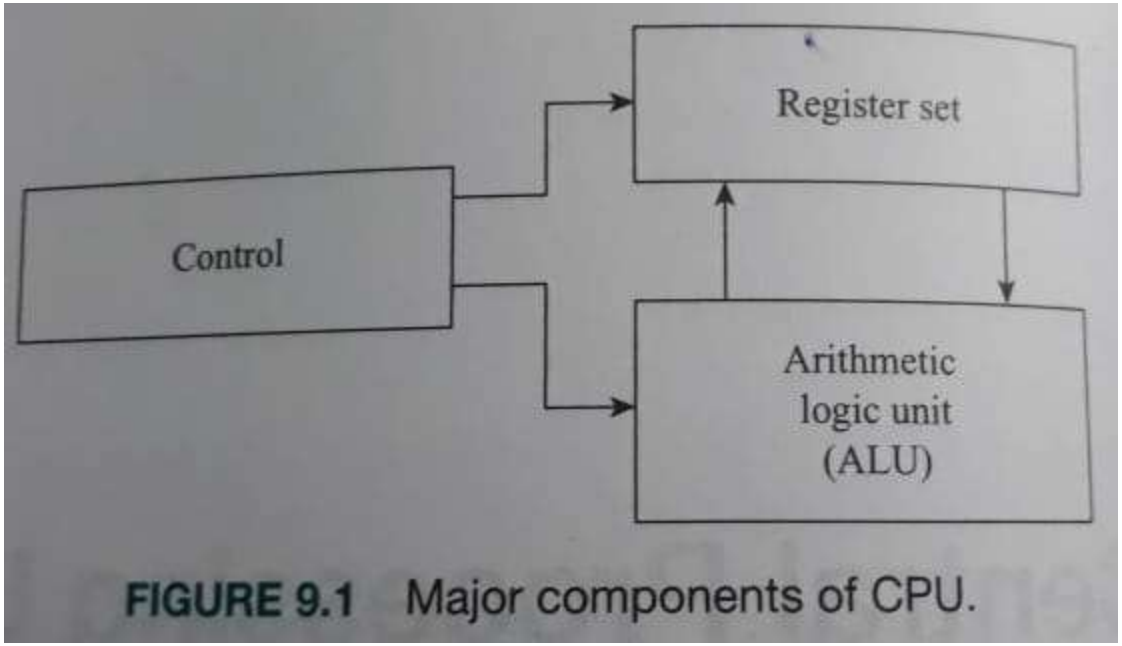
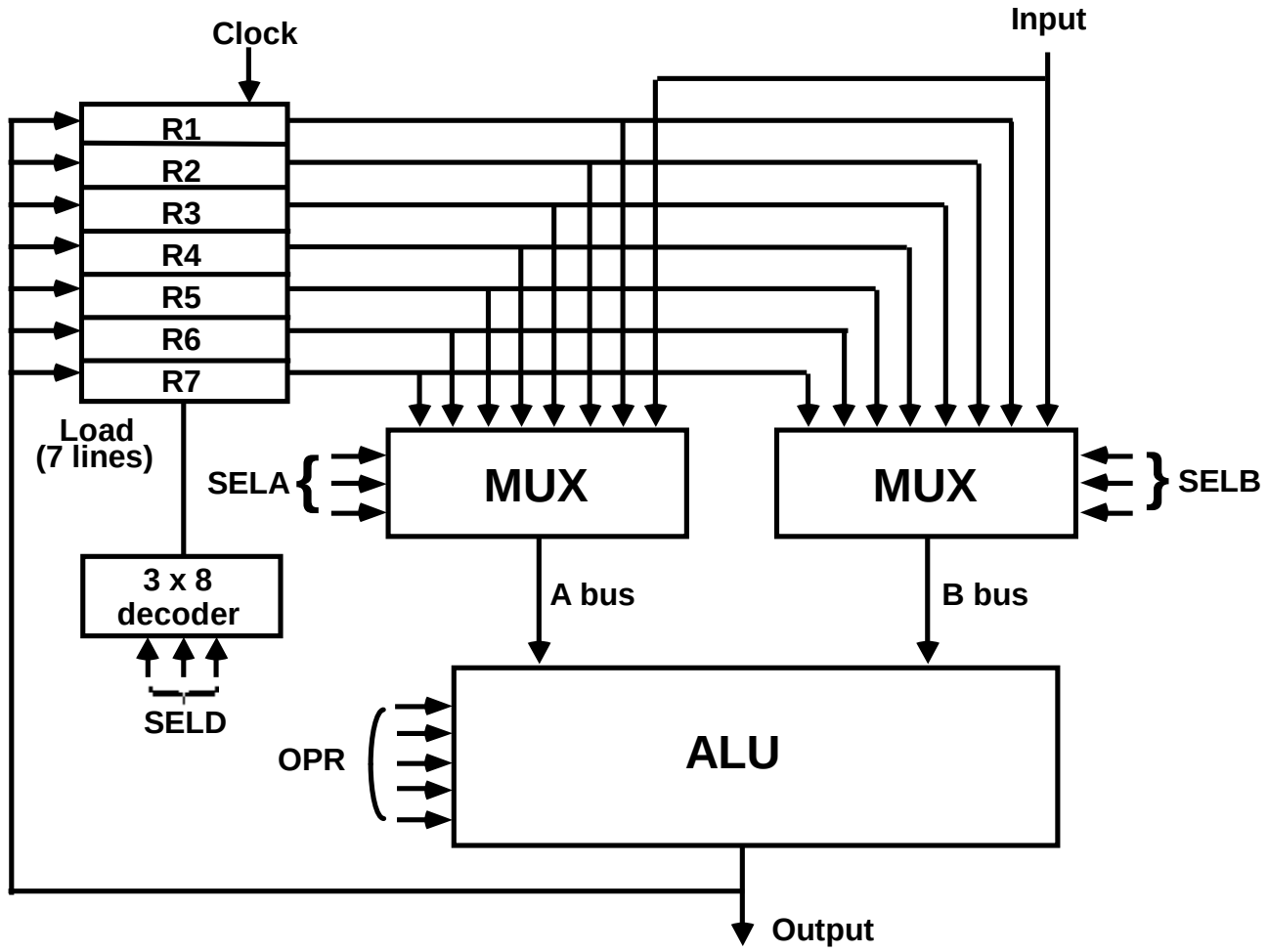


FIGURE 9.1 Major components of CPU.

GENERAL REGISTER ORGANIZATION



OPERATION OF CONTROL UNIT

The control unit directs the information flow through ALU by:

- Selecting various *Components* in the system
- Selecting the *Function* of ALU

Example: R1 <- R2 + R3

[1] MUX A selector (SELA): BUS A ← R2
[2] MUX B selector (SELB): BUS B ← R3
[3] ALU operation selector (OPR): ALU to ADD
[4] Decoder destination selector (SELD): R1 ← Out Bus

Control Word

Encoding of register selection fields

Binary Code	SELA	SELB	SELD
000	Input	Input	None
001	R1	R1	R1
010	R2	R2	R2
011	R3	R3	R3
100	R4	R4	R4
101	R5	R5	R5
110	R6	R6	R6
111	R7	R7	R7

ALU CONTROL

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

Examples of ALU Microoperations

	Symbolic Designation				
Microoperation	SELA	SELB	SELD	OPR	Control Word
R1 ← R2 - R3	R2	R3	R1	SUB	010 011 001 00101
R4 ← R4 ∨ R5	R4	R5	R4	OR	100 101 100 01010
R6 ← R6 + 1	R6	-	R6	INCA	110 000 110 00001
R7 ← R1	R1	-	R7	TSFA	001 000 111 00000
Output ← R2	R2	-	None	TSFA	010 000 000 00000
Output ← Input	Input	-	None	TSFA	000 000 000 00000
R4 ← shl R4	R4	-	R4	SHLA	100 000 100 11000
R5 ← 0	R5	R5	R5	XOR	101 101 101 01100

ALU CONTROL

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

Examples of ALU Microoperations

	Symbolic Designation				
Microoperation	SELA	SELB	SELD	OPR	Control Word
R1 ← R2 + R3					
R4 ← R4					
R5 ← R5 - 1					
R6 ← Shl R1					
R7 ← Input					

ALU CONTROL

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

Examples of ALU Microoperations

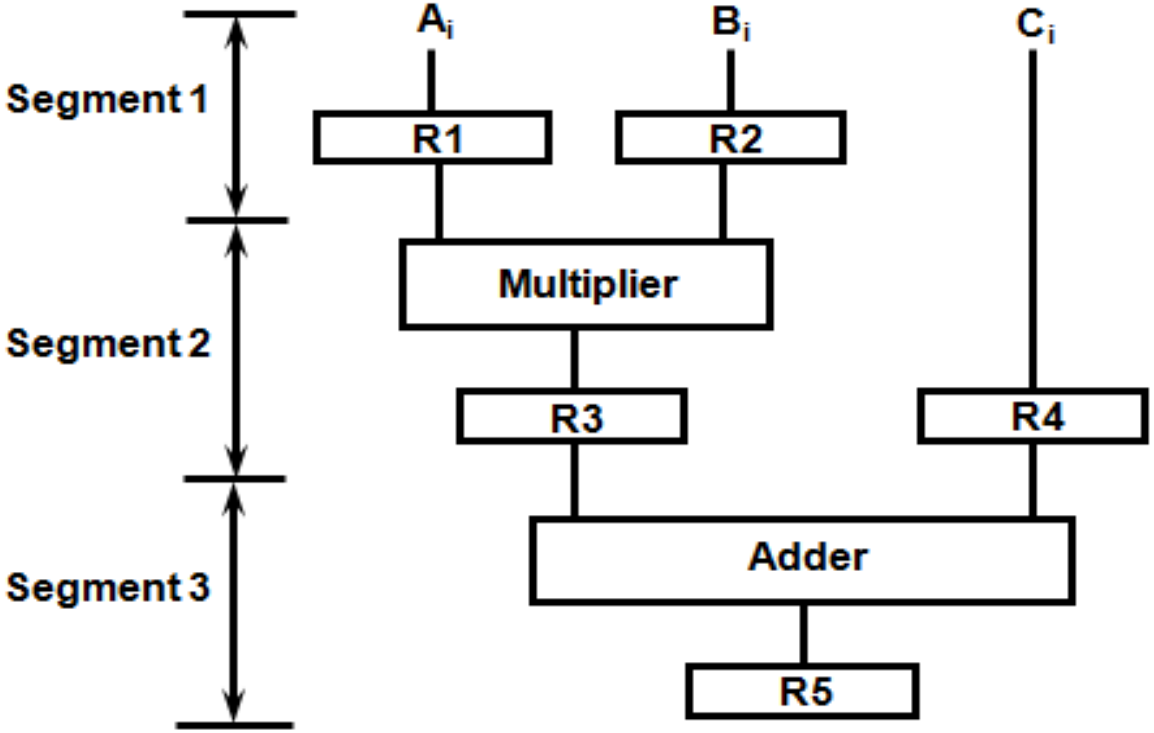
	Symbolic Designation				
Microoperation	SELA	SELB	SELD	OPR	Control Word
R1 ← R2 + R3	R2	R3	R1	ADD	
R4 ← R4	R4	-	R4	TSFA	
R5 ← R5 - 1	R5	-	R5	DECA	
R7 ← R1	R1	-	R7	TSFA	
Output ← R2	R2	-	None	TSFA	

PIPELINING

- Pipelining is a technique of decomposing a sequential process into sub-operations, with each sub-process being executed in a segment that operates concurrently with all other segment.

Example for Pipeline Processing

$A_i * B_i + C_i$ for $i = 1, 2, 3, \ldots, 7$

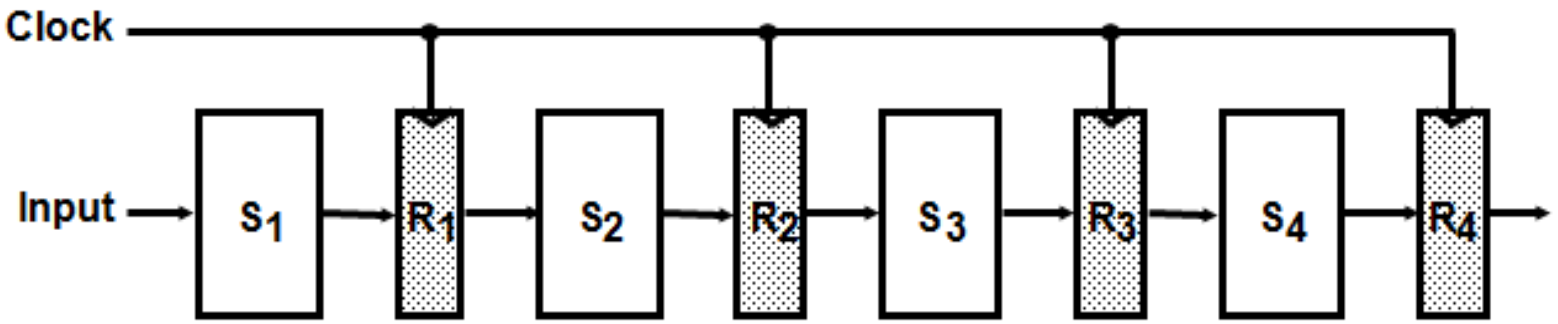


$R1 \leftarrow A_i$, $R2 \leftarrow B_i$	Input A_i and B_i
$R3 \leftarrow R1 * R2$, $R4 \leftarrow C_i$	Multiply and input C_i
$R5 \leftarrow R3 + R4$	Add C_i to product

Operations in each pipeline stage

Clock Pulse Number	Segment 1		Segment 2		Segment 3
	R1	R2	R3	R4	R5
1	A1	B1	---	---	---
2	A2	B2	A1 * B1	C1	---
3	A3	B3	A2 * B2	C2	A1 * B1 + C1
4	A4	B4	A3 * B3	C3	A2 * B2 + C2
5	A5	B5	A4 * B4	C4	A3 * B3 + C3
6	A6	B6	A5 * B5	C5	A4 * B4 + C4
7	A7	B7	A6 * B6	C6	A5 * B5 + C5
8	---	---	A7 * B7	C7	A6 * B6 + C6
9	---	---	---	---	A7 * B7 + C7

General Four-Segment Pipeline



Space-Time Diagram

	1	2	3	4	5	6	7	8	9
Segment 1	T1	T2	T3	T4	T5	T6			
2		T1	T2	T3	T4	T5	T6		
3			T1	T2	T3	T4	T5	T6	
4				T1	T2	T3	T4	T5	T6

Clock cycles

Pipeline Speedup

n: Number of tasks to be performed

Pipelined Machine (k segments)
k-segment pipeline with a clock cycle time of t_p is used to execute n tasks

The first task T_1 requires time = $k*t_p$

The remaining (n-1) tasks emerge from the pipe at the rate of one task per clock cycle = $(n-1)* t_p$

To complete n task with k-segment pipeline= k + (n-1) clock cycles

Conventional Machine (Non-Pipelined)

Time required to complete each task = t_n

Time required to complete the n tasks = $n * t_n$

Speedup

S_k: Speedup

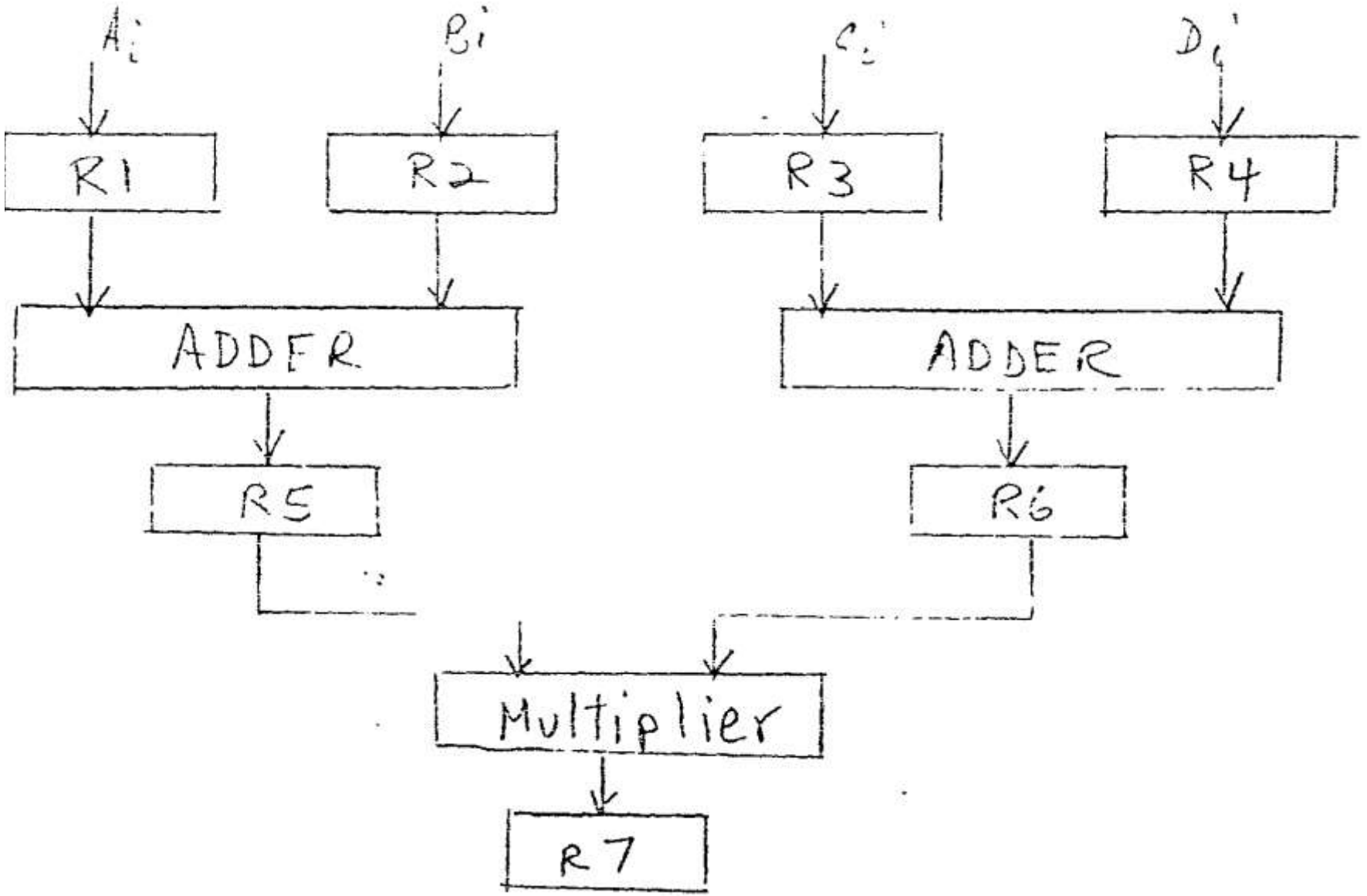
$S_k = n*t_n / (k + n - 1)*t_p$

Q1. In certain scientific computation it is necessary to perform the arithmetic operation (Ai+Bi)*(Ci*Di)

a. Specify the pipeline configuration to carry out this task.
b. List the content of all registers of all the registers in the pipeline for i=1 to 6.

Q2. Draw a space time diagram for a six-segment pipeline showing the time it takes to process eight tasks.

Q3. A non-pipelined system takes 50ns to process a task. The same task can be processed in a six-segment pipeline with a clock cycle of 10ns. Determine the speedup ratio of the pipeline for 100 tasks.


Ai
Bi
Ci
Di
R1
R2
R3
R4
ADDER
ADDER
R5
R6
Multiplier
R7

Segment	1	2	3	4	5	6	7	8	9	10	11	12	13
1	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8					
2		T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8				
3			T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8			
4				T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8		
5					T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	
6						T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8

$$(k-n-1)t_p = 6+8-1 = 13 \text{ cycles}$$

$t_n = 50\ ns$

$k = 6$

$t_p = 10\ ns$

$n = 100$

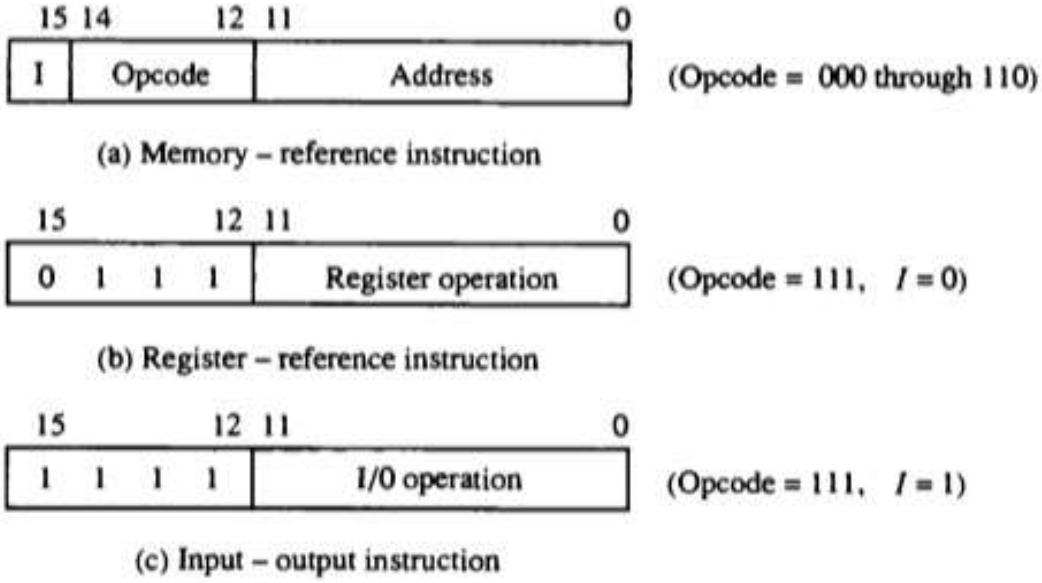
$$S = \frac{n\,t_n}{(k+n-1)\,t_r} = \frac{100 \times 50}{(6-99) \times 10} = 4.76$$

$$S_{max} = \frac{t_n}{t_p} = \frac{50}{10} = 5$$

Computer Instructions

- The basic computer has three instruction code formats.
- Each format has 16 bits.
- The operation code(op-code) part of the instruction contains three bits and the meanir encountered.

Basic computer instruction formats.



Computer Instruction Set

Basic Computer Instructions

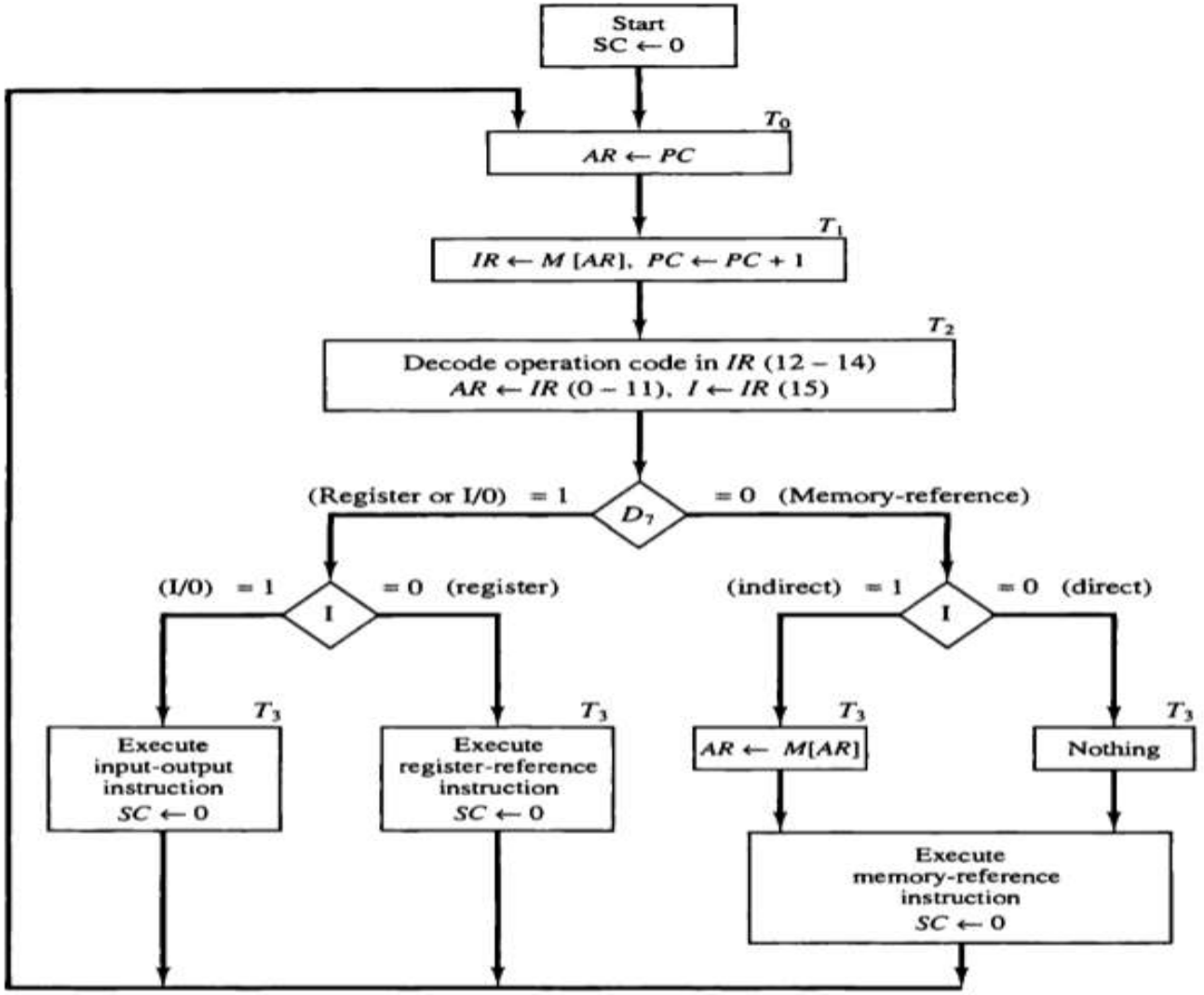
Symbol	Hexadecimal code $I = 0$	Hexadecimal code $I = 1$	Description
AND	0xxx	8xxx	AND memory word to *AC*
ADD	1xxx	9xxx	Add memory word to *AC*
LDA	2xxx	Axxx	Load memory word to *AC*
STA	3xxx	Bxxx	Store content of *AC* in memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return 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

Instruction Cycle

- The program is executed in the computer by going thru a cycle for each instruction.
- In basic computer, each instruction cycle consists of the following phases:

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

Instruction Cycle



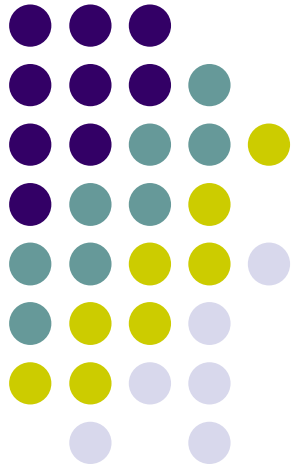
Flowchart for instruction cycle (initial configuration).

Instruction Cycle

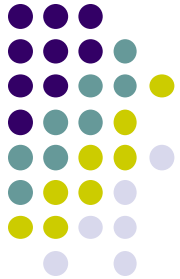
- Micro-operations for fetch & decode

T_0: $AR \leftarrow PC$
T_1: $IR \leftarrow M[AR]$, $PC \leftarrow PC + 1$
T_2: $D_0, \ldots, D_7 \leftarrow$ Decode $IR(12\text{–}14)$, $AR \leftarrow IR(0\text{–}11)$, $I \leftarrow IR(15)$

Basic Processing Unit

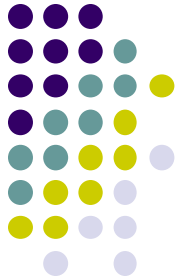


Fundamental Concepts



- Processor fetches one instruction at a time and perform the operation specified.
- Instructions are fetched from successive memory locations until a branch or a jump instruction is encountered.
- Processor keeps track of the address of the memory location containing the next instruction to be fetched using Program Counter (PC).
- Instruction Register (IR)

Executing an Instruction



- Fetch the contents of the memory location pointed to by the PC. The contents of this location are loaded into the IR (fetch phase).

$$IR \leftarrow [[PC]]$$

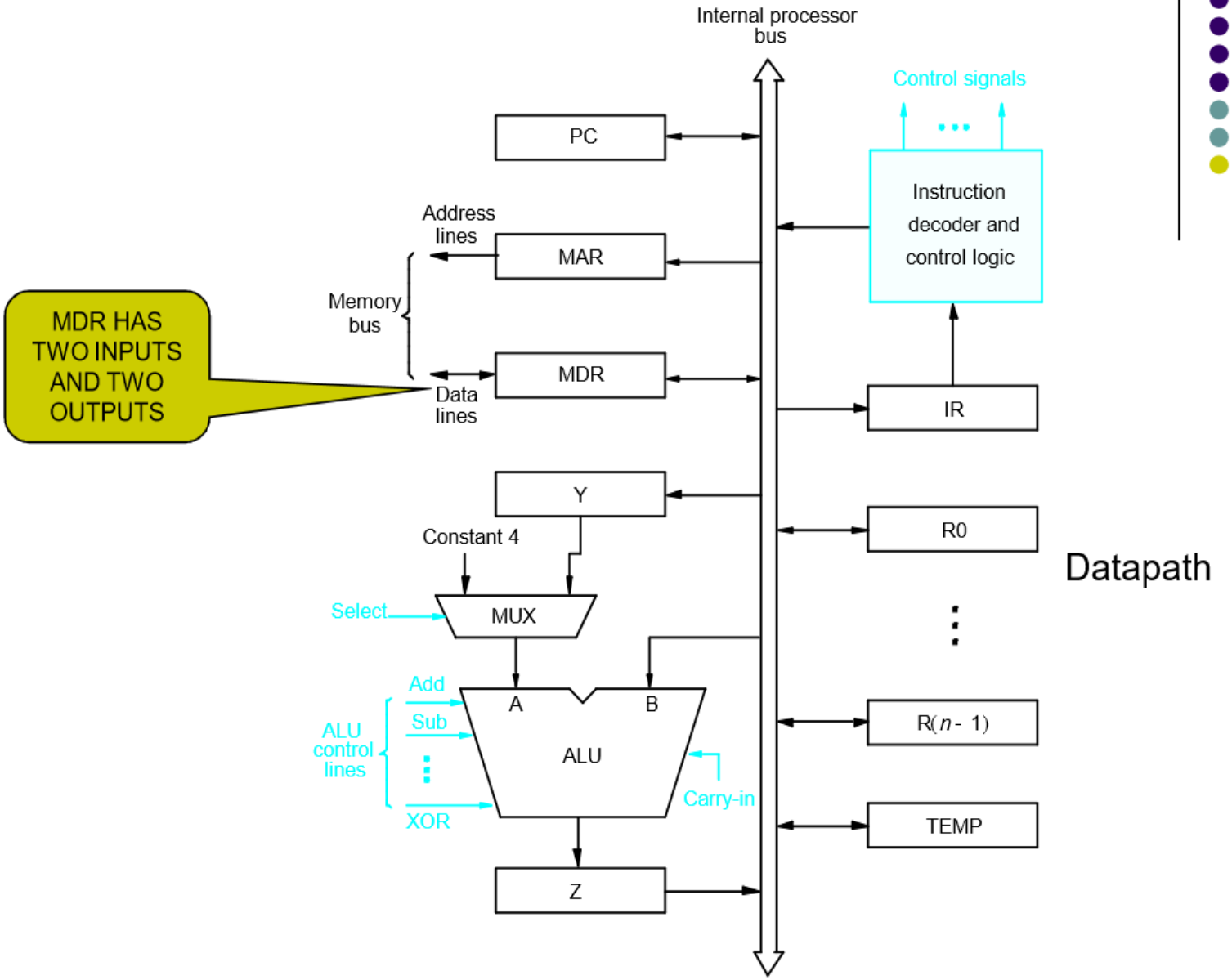
- Assuming that the memory is byte addressable, increment the contents of the PC by 4 (fetch phase).

$$PC \leftarrow [PC] + 4$$

(where each instruction comprises 4 bytes)

- Carry out the actions specified by the instruction in the IR (execution phase).

Processor Organization



Datapath

Registers:

- The processor registers R0 to Rn-1 vary considerably from one processor to another.
- Registers are provided for general purpose used by programmer.
- Registers Y, Z & TEMP are temporary registers used by processor during the execution of some instruction.

Multiplexer:

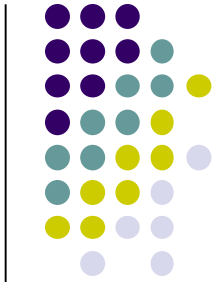
- Select either the output of the register Y or a constant value 4 to be provided as input A of the ALU.
- Constant 4 is used by the processor to increment the contents of PC.

ALU:

Used to perform arithmetic and logical operation.

Data Path:

The registers, ALU and interconnecting bus are collectively referred to as the data path.



Register Transfers

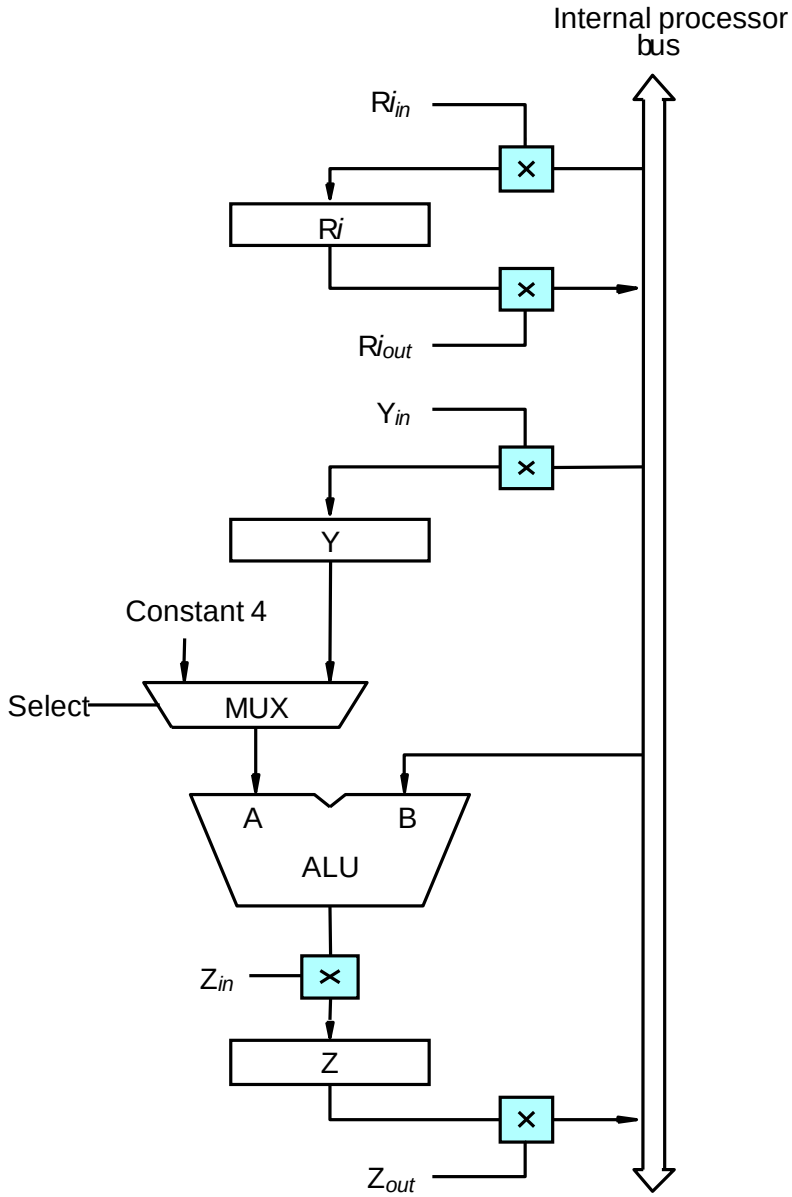
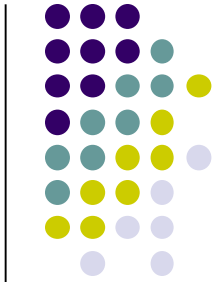


Figure 7.2. Input and output gating for the registers in Figure 7.1.

- The input and output gates for register Ri are controlled by signals is R_{in} and R_{iout} .
- If R_{in} is set to1 - data available on common bus are loaded into Ri.
- If R_{iout} is set to1 - the contents of register are placed on the bus.
- R_{iout} is set to 0 – the bus can be used for transferring data from other registers .

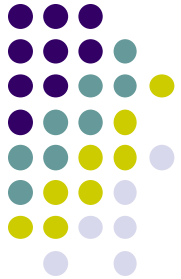
Data transfer between two registers:



Transfer the contents of R1 to R4.

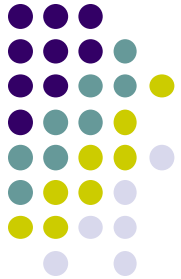
1. Enable output of register R1 by setting $R1_{out}$=1. This places the contents of R1 on the processor bus.
2. Enable input of register R4 by setting $R4_{in}$=1. This loads the data from the processor bus into register R4.

Performing an Arithmetic or Logic Operation



- The ALU is a combinational circuit that has no internal storage.
- ALU gets the two operands from MUX and bus. The result is temporarily stored in register Z.
- What is the sequence of operations to add the contents of register R1 to those of R2 and store the result in R3?
  1. $R1_{out}$, Y_{in}
  2. $R2_{out}$, Select Y, Add, Z_{in}
  3. Z_{out}, $R3_{in}$

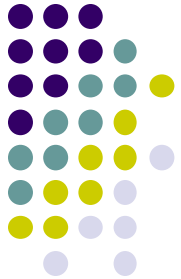
Fetching a Word from Memory



- Address into MAR; issue Read operation; data into MDR.

Figure 7.4. Connection and control signals for register MDR.

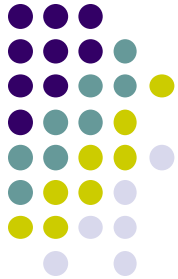
Fetching a Word from Memory



- The response time of each memory access varies (cache miss, memory-mapped I/O,…).
- To accommodate this, the processor waits until it receives an indication that the requested operation has been completed (Memory-Function-Completed, MFC).

- **Example: Move (R1), R2**
  - MAR ← [R1]
  - Start a Read operation on the memory bus
  - Wait for the MFC response from the memory
  - Load MDR from the memory bus
  - R2 ← [MDR]

Fetching a Word from Memory



Example: Move (R1), R2

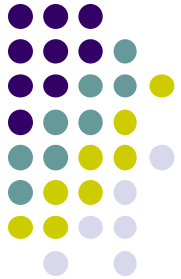
- MAR ← [R1]
- Start a Read operation on the memory bus
- Wait for the MFC response from the memory
- Load MDR from the memory bus
- R2 ← [MDR]

Assume MAR is always available on the address lines of the memory bus.

- Move (R1), R2

1. $R1_{out}$, MAR_{in}, Read
2. MDR_{inE}, WMFC
3. MDR_{out}, $R2_{in}$

Storing a word in memory



- Address is loaded into MAR
- Data to be written loaded into MDR.
- Write command is issued.

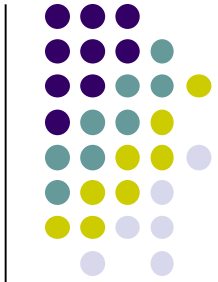
- **Example: Move R2,(R1)**

  $R1_{out}$,MAR_{in}

  $R2_{out}$,MDR_{in},Write

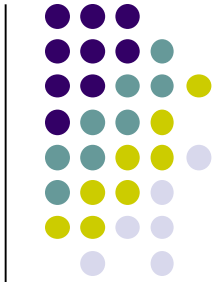
  MDR_{outE}, WMFC

Execution of a Complete Instruction



- **Example: Add (R3), R1**
- Fetch the instruction
- Fetch the first operand (the contents of the memory location pointed to by R3)
- Perform the addition
- Load the result into R1

Execution of a Complete Instruction



Add (R3), R1

Step	Action
1	PC_{out} , MAR_{in} , Read, Select4, Add, Z_{in}
2	Z_{out} , PC_{in} , Y_{in} , WMF C
3	MDR_{out} , IR_{in}
4	$R3_{out}$, MAR_{in} , Read
5	$R1_{out}$, Y_{in} , WMF C
6	MDR_{out} , SelectY, Add, Z_{in}
7	Z_{out} , $R1_{in}$, End

Question

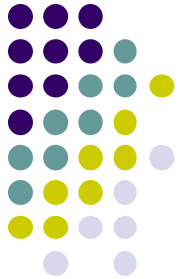
- Write the control sequence to execute

1. Add the (immediate) number NUM to register R1.

2. Add the contents of memory location NUM to register R1.

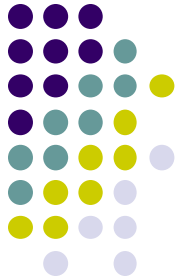
3. Add the contents of memory location whose address is at memory location NUM to register R1.

Solution 1



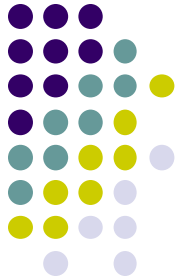
- PC out, MAR in, Read, Select4, Add , Zin
- Zout, PCin, Yin, WMFC
- MDRout, IR in,
- IR NUMout , Yin
- R1 out, Select Y, Add, Zin
- Zout, R1in, End
-

Solution 2



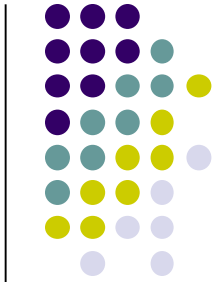
- PC out, MAR in, Read, Select4, Add , Zin
- Zout, PCin, Yin, WMFC
- MDRout, IR in,
- IRNUM out, MAR in, Read
- R1out, Yin, WMFC
- MDRout, Select Y, Add, Zin
- Zout, R1in, End

Solution 3



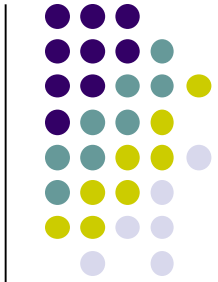
- PC out, MAR in, Read, Select4, Add , Zin
- Zout, PCin, Yin, WMFC
- MDRout, IR in
- IRNUM out, MAR in, Read
- WMFC
- MDRout, MAR in, Read
- R1out, Yin, WMFC
- MDRout , SelectY, Add, Zin
- Zout, R1in, End

Execution of Branch Instructions



- A branch instruction replaces the contents of PC with the branch target address, which is usually obtained by adding an offset X given in the branch instruction.
- The offset X is usually the difference between the branch target address and the address immediately following the branch instruction.
- UnConditional branch

Execution of Branch Instructions

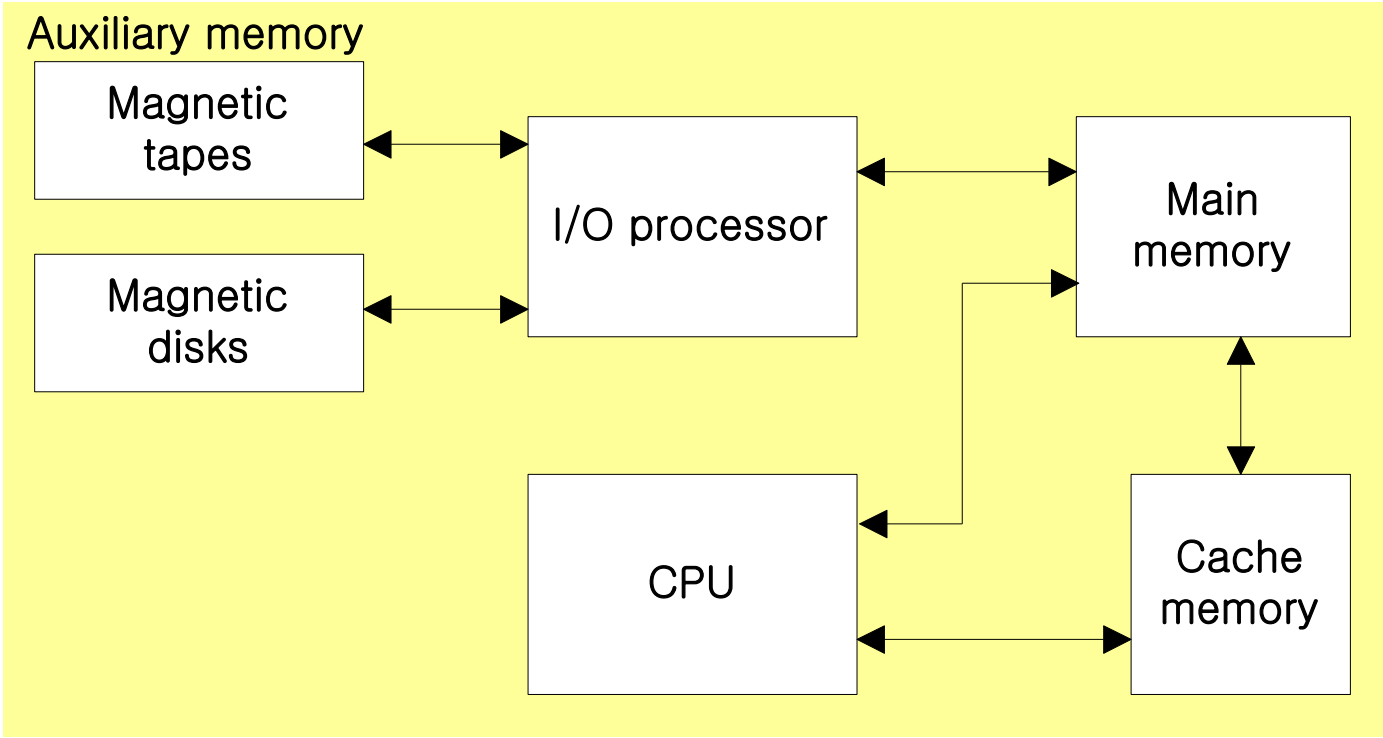


Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	Offset-field-of-IR_{out}, Select Y, Add, Z_{in}
5	Z_{out}, PC_{in}, End

Figure 7.7. Control sequence for an unconditional branch instruction.

Memory Organization

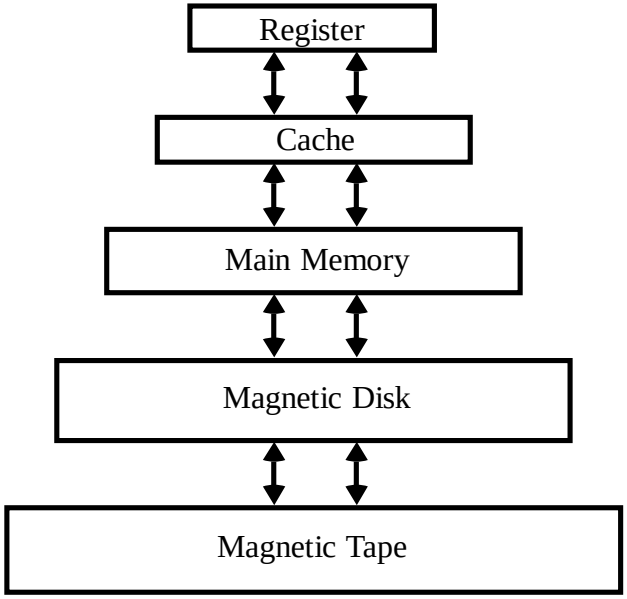
- Memory Hierarchy
  - Memory hierarchy in a computer system :
    - **Main Memory** : memory unit that communicates directly with the CPU (RAM)
    - **Auxiliary Memory** : device that provide backup storage (Disk Drives)
    - **Cache Memory** : special very-high-speed memory to increase the processing speed (Cache RAM)



Memory Organization

- Multiprogramming
  - enable the CPU to process a number of independent program concurrently
- Memory Management System :
  - supervise the flow of information between auxiliary memory and main memory

- Memory Hierarchy is to obtain the highest possible access speed while minimizing the total cost of the memory system



Random Access Memory (RAM)

RAM Chips

Static RAM

- Consists of Flip-flops to store binary information.
- Static RAM is easier to use and having short Read/Write cycles
- Used mostly in Cache memory.

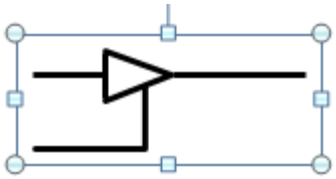
Dynamic RAM

- Stores binary information in the form of electric charge stored inside the capacitor.
- The capacitors are provided by MOS transistors.
- Refreshing circuit is required to refresh the memory.
- Dynamic RAM offers reduced power consumption and large storage capacity.
- It is used to construct main memory.

Static RAM Vs Dynamic RAM

Static RAM	Dynamic RAM
➢ SRAM uses transistor to store a single bit of data	➢ DRAM uses a separate capacitor to store each bit of data
➢ SRAM does not need periodic refreshment to maintain data	➢ DRAM needs periodic refreshment to maintain the charge in the capacitors for data
➢ SRAM's structure is complex than DRAM	➢ DRAM's structure is simplex than SRAM
➢ SRAM are expensive as compared to DRAM	➢ DRAM's are less expensive as compared to SRAM
➢ SRAM are faster than DRAM	➢ DRAM's are slower than SRAM
➢ SRAM are used in Cache memory	➢ DRAM are used in Main memory

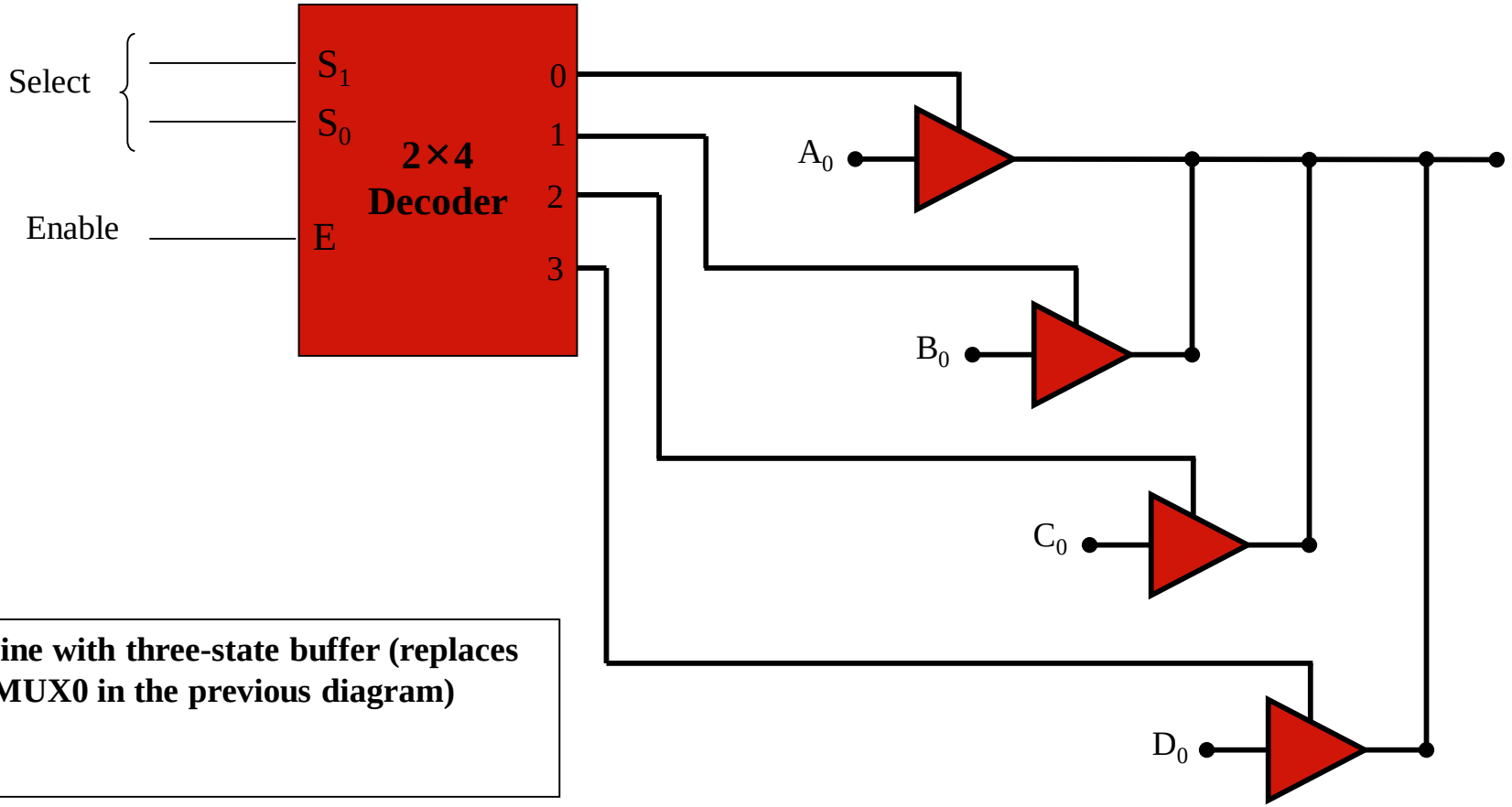
Three State Buffer



Normal input A

Control input C

Output Y=A if C=1
High-impedence if C=0

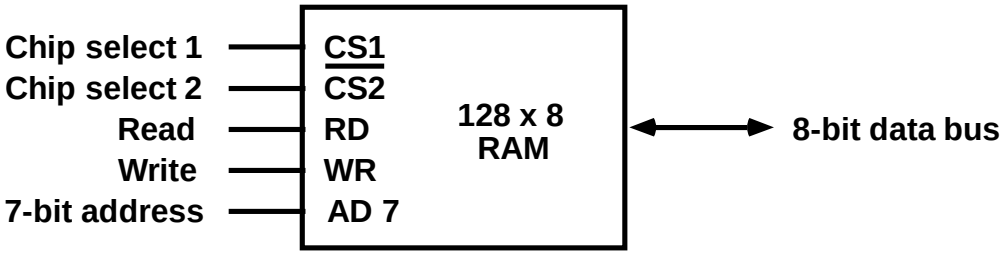


Bus line with three-state buffer (replaces MUX0 in the previous diagram)

MAIN MEMORY

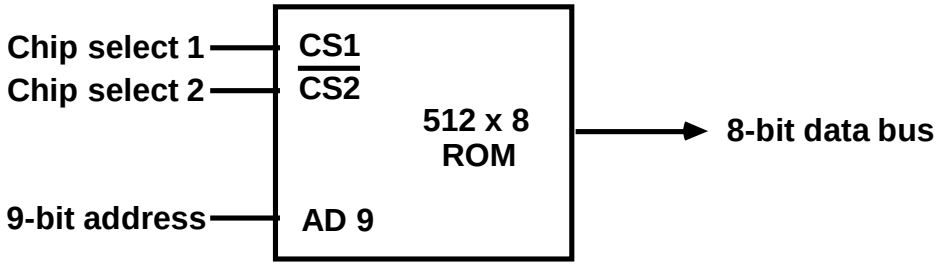
RAM and ROM Chips

Typical RAM chip



CS1	$\overline{CS2}$	RD	WR	Memory function	State of data bus
0	0	x	x	Inhibit	High-impedence
0	1	x	x	Inhibit	High-impedence
1	0	0	0	Inhibit	High-impedence
1	0	0	1	Write	Input data to RAM
1	0	1	x	Read	Output data from RAM
1	1	x	x	Inhibit	High-impedence

Typical ROM chip



MEMORY ADDRESS MAP

Address space assignment to each memory chip

Example: 512 bytes RAM and 512 bytes ROM

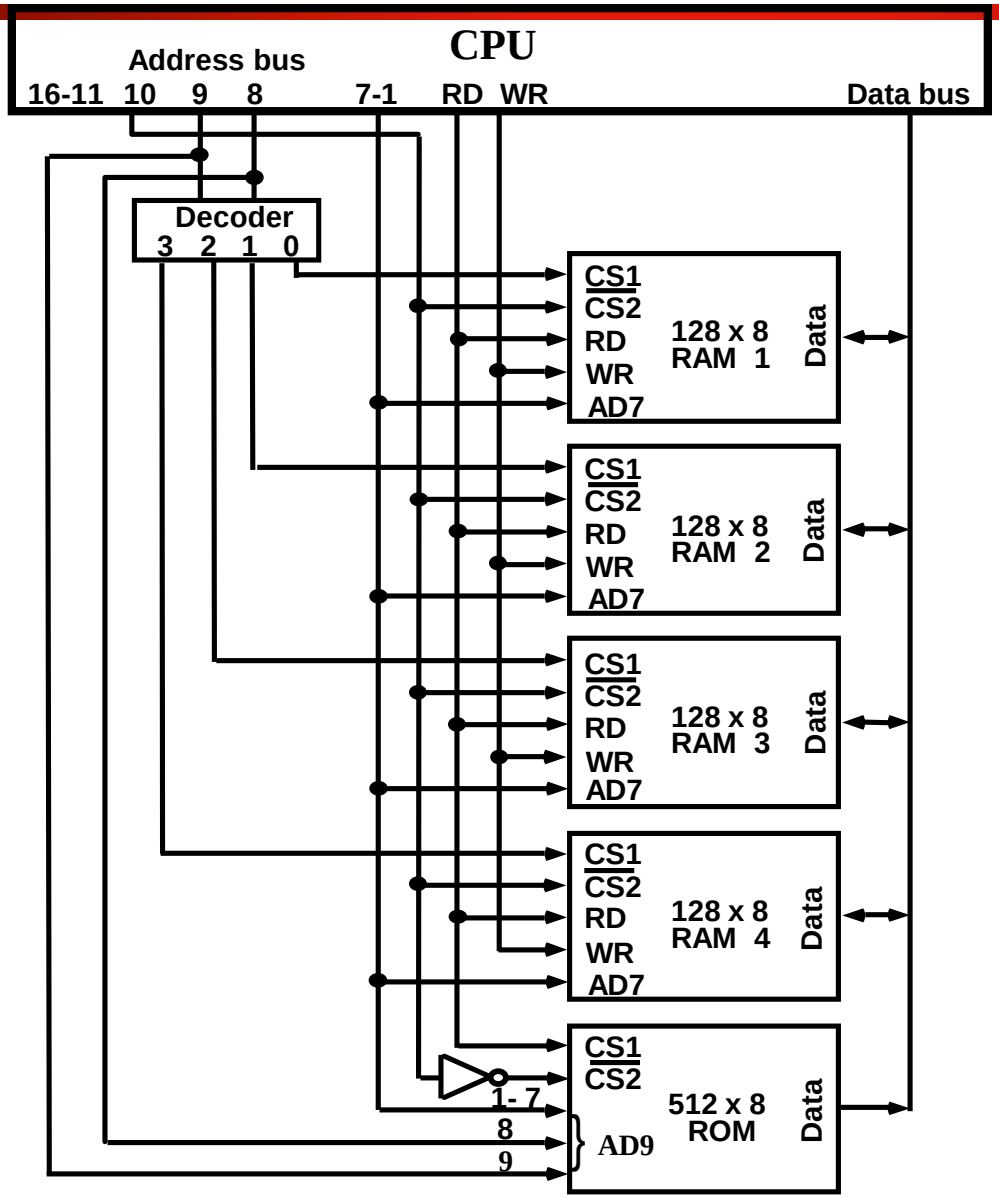
Component	Hexadecimal address	Address bus									
		10	9	8	7	6	5	4	3	2	1
RAM 1	0000 - 007F	0	0	0	x	x	x	x	x	x	x
RAM 2	0080 - 00FF	0	0	1	x	x	x	x	x	x	x
RAM 3	0100 - 017F	0	1	0	x	x	x	x	x	x	x
RAM 4	0180 - 01FF	0	1	1	x	x	x	x	x	x	x
ROM	0200 - 03FF	1	x	x	x	x	x	x	x	x	x

Memory Connection to CPU

- RAM and ROM chips are connected to a CPU through the data and address buses

- The low-order lines in the address bus select the byte within the chips and other lines in the address bus select a particular chip through its chip select inputs

CONNECTION OF MEMORY TO CPU



Numerical Problems

Q1. a) How many 128x8 RAM chips are needed to provide a memory capacity of 2048 bytes.

b) How any lines of the address bus must be used to access 2048 bytes of memory. How many of these lines will be common to all chips.

c) How many lines must be decoded for chip select? Specify the size of decoders.

Q2. Extend the memory system of Fig.1 to 4096 bytes of RAM and 4096 bytes of RAM. List the memory-address map and indicate what size decoders are needed.

Q3. A computer employs RAM chips of 256x8 and ROM chips of 1024x8. The computer system needs 2K bytes of RAM, 4K bytes of ROM, and four interface units each with four registers. A memory-mapped I/O configuration used. The two highest-order bits of the address bus are assigned 00 for RAM, 01 for ROM, and 10 for interface registers.

How many RAM and ROM chips are needed.

Draw a memory-address map for the system.

Give the address range in hexadecimal for RAM, ROM and interface.

CHAPTER 12

12-1 (a) $\frac{2048}{128} = 16$ chips

(b) $2048 = 2^{11}$, $128 = 2^{7}$

11 lines to address 2048 bytes
7 lines to address each chip
4 lines to decoder for selecting 16 chips

(c) 4x16 decoder

12-4

4096/128 = 32 RAM chips ; 4096/512 = 8 ROM chips.

$4096 = 2^{12}$ — There 12 common address lines +1 line to select between RAM and ROM.

Component	Address	16 15 14 13	12 11 10 9 8 7 6 5	4 3 2 1
RAM	0000-0FFF	0 0 0 0	← 5x32 decoder → x x x	x x x x
ROM	1000-1FFF	0 0 0 1	← 3x8 → x	x x x	x x x x

to $\overline{CS2}$ ↑ decoder

Cache Memory

- 12-5 Cache Memory
  - Locality of Reference
    - the references to memory *tend to be confined within a few localized areas* in memory
  - Cache Memory : **a fast small memory**
    - keeping the most frequently accessed instructions and data in the fast cache memory
  - Hit Ratio
    - the ratio of the number of hits divided by the total CPU references (hits + misses) to memory
      - » ***hit*** : the CPU finds the word in the cache
      - » ***miss*** : the word is not found in cache (CPU must read main memory)
  - A computer with cache access time of 100ns, a main memory access time of 1000 ns, a hit ratio of 0.9 is having average access time of 200ns.

Types of Mapping of Cache Memory

- Mapping
  - The transformation of data from main memory to cache memory
    - » 1) Associative mapping
    - » 2) Direct mapping
    - » 3) Set-associative mapping

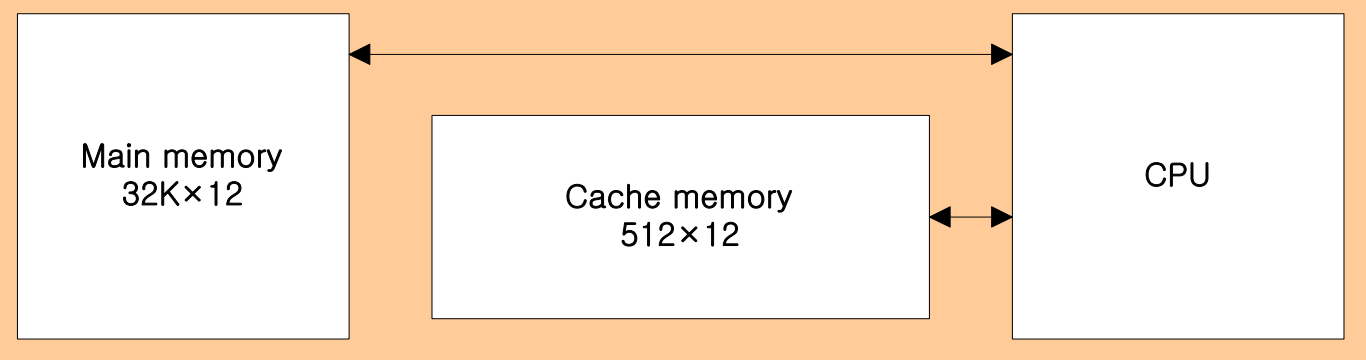
- Associative Mapping:

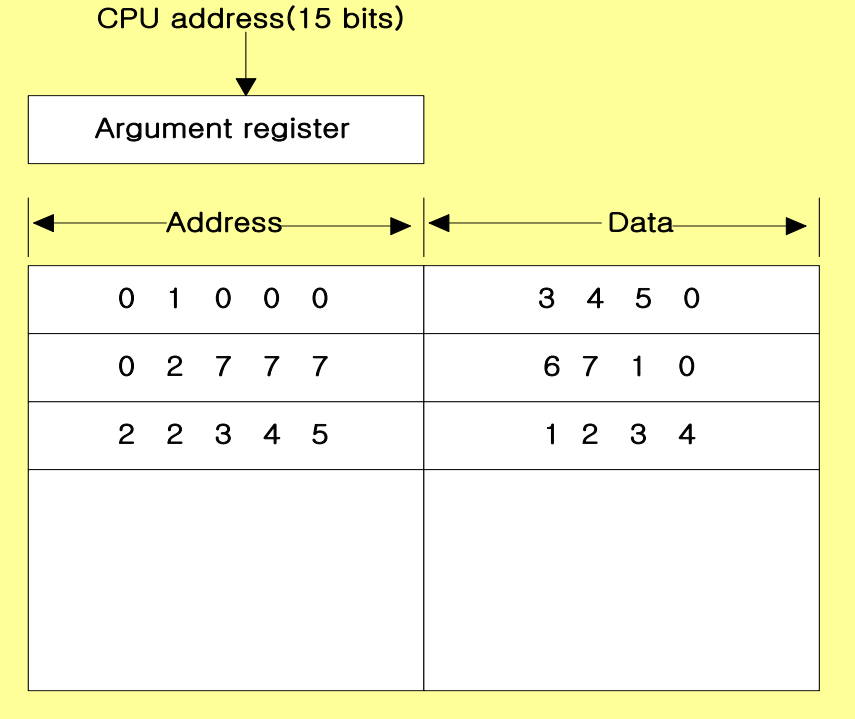
  main memory : **32 K** x 12 bit word (15 bit address lines)

  cache memory : **512** x 12 bit word

    - » CPU sends a 15-bit address to cache
      - **Hit** : CPU accepts the 12-bit data from cache
      - **Miss** : CPU reads the data from main memory (then data is written to cache)

Associative Mapping





Direct Mapping

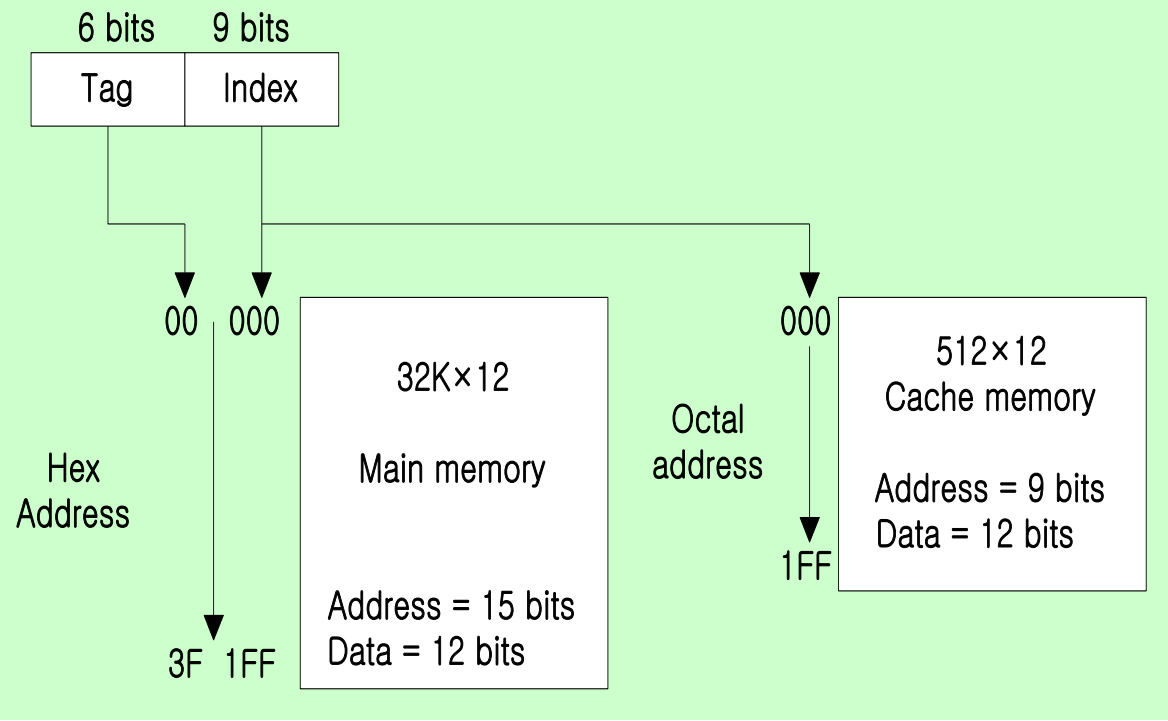
Direct mapping :

n bit memory address

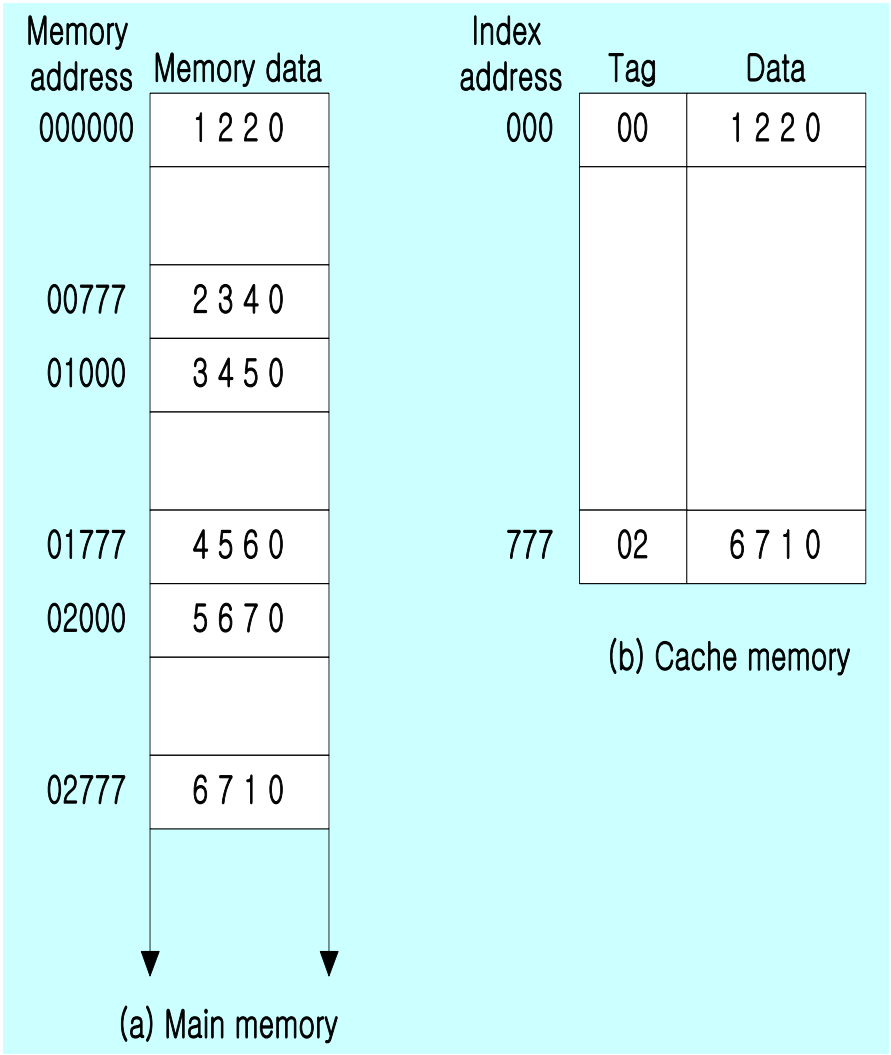
Tag field (**n - k**) : Index field (**k**)

2^k words cache memory + 2^n words main memory

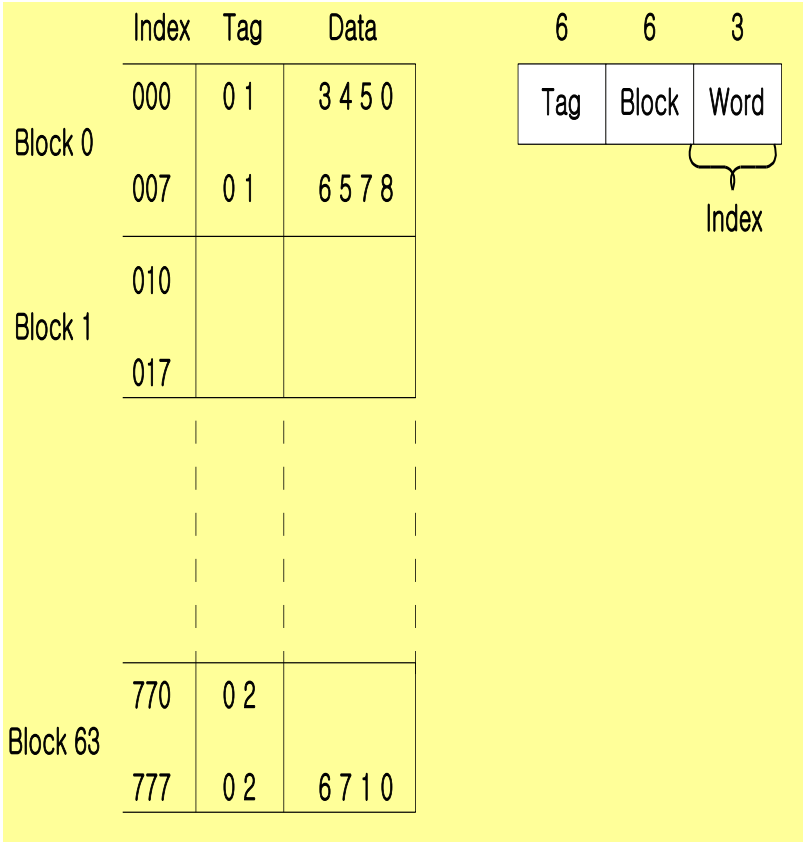
Tag = 6 bit (15 - 9), **Index** = 9 bit



Direct Mapping



- Direct mapping cache with block size of 8 words :
  - » 64 block x 8 word = 512 cache words size



Set-Associative Mapping

Set-associative mapping : (two-way)

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

Questions

- A two-way set associative cache memory uses blocks of four words. The cache can accommodate a total of 2048 words from main memory. The main memory size is 128Kx32. Formulate all pertinent information required to construct the cache memory. What is the size of cache memory.

- A computer has a memory unit of 64Kx16 and a cache memory of 1K words. The cache uses direct mapping with a block size of four words. How many bits are there in the tag, index, block, and words fields of the address format. How many bits are there in each word of cache, and how are they divided into functions? Include a valid bit. How many blocks can the cache accommodate?

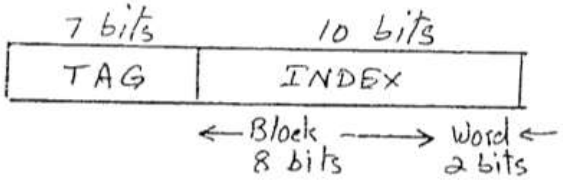
- The access time of a cache memory is 100ns and that of main memory is 1000ns. It is estimated that 80 percent of the memory requests are for read and the remaining 20 percent for write. The hit ratio for read access only is 0.9. A write-through procedure is used. What is the average access time of the system considering only memory read cycles. What is the average access time of the system for both read and write requests. What is the hit ratio considering the write requests also.

Answer

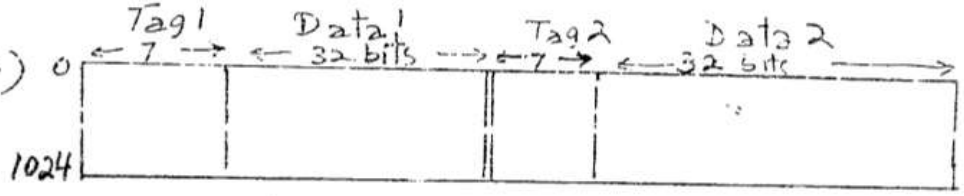
12-15

$128K = 2^{17}$; For a set size of 2, the index address has 10 bits to accomodate $\frac{2048}{2} = 1024$ words of cache.

(a)



(b)



Size of cache memory is $1024 \times 2(7+32)$
$= 1024 \times 78$

12-16

(a) $0.9 \times \underbrace{100}_{\text{cache access}} + 0.1 \times \underbrace{11000}_{\text{cache+memory access}} = 90 + 110 = 200$ nsec.

(b) $0.2 \times \underbrace{1000}_{\text{write access}} + 0.8 \times \underbrace{200}_{\text{read access from (a)}} = 200 + 160 = 360$ nsec.

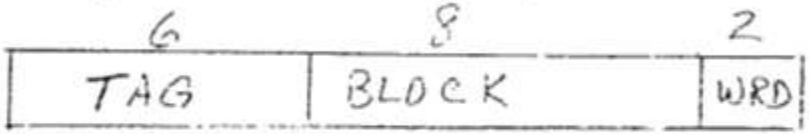
(c) Hit ratio $= 0.8 \times 0.9 = 0.72$

Answer

12-18

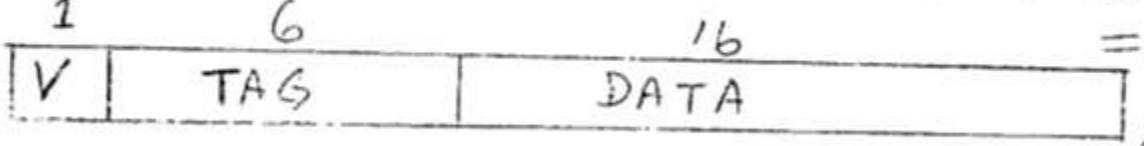
64K × 16 : 16 bit address ; 16-bit data.

(a)



= 16 bit address

(b)



= 23 bits in each word of cache

(c) 2^8 = 256 blocks of 4 words each

- Replacement Algorithm : cache miss or full
  - 1) **LRU** (Least Recently Used):
  - 2) **Random Replacement:**
  - 3) **FIFO** (First-In First-Out) :

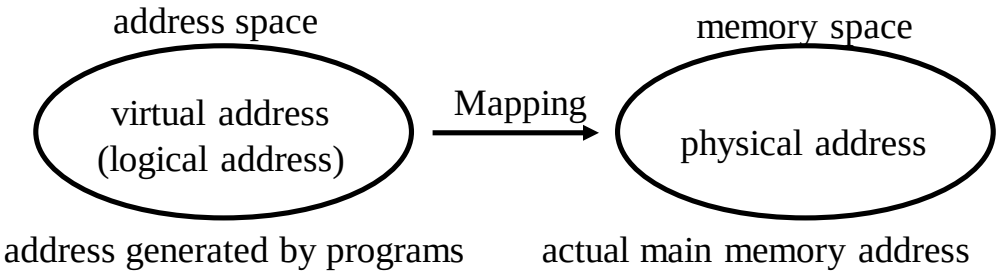
- Writing to Cache :
  - » 1) **Write-through** :
  - » 2) **Write-back** :

- Cache Initialization
  - Cache is initialized :
    - » 1) When power is applied to the computer
    - » 2) When main memory is loaded with a complete set of programs from auxiliary memory
    - » 3) Cache is initialized by clearing all the valid bits to 0.
  - Valid bit
    - » Indicate whether or not the word contains valid data

VIRTUAL MEMORY

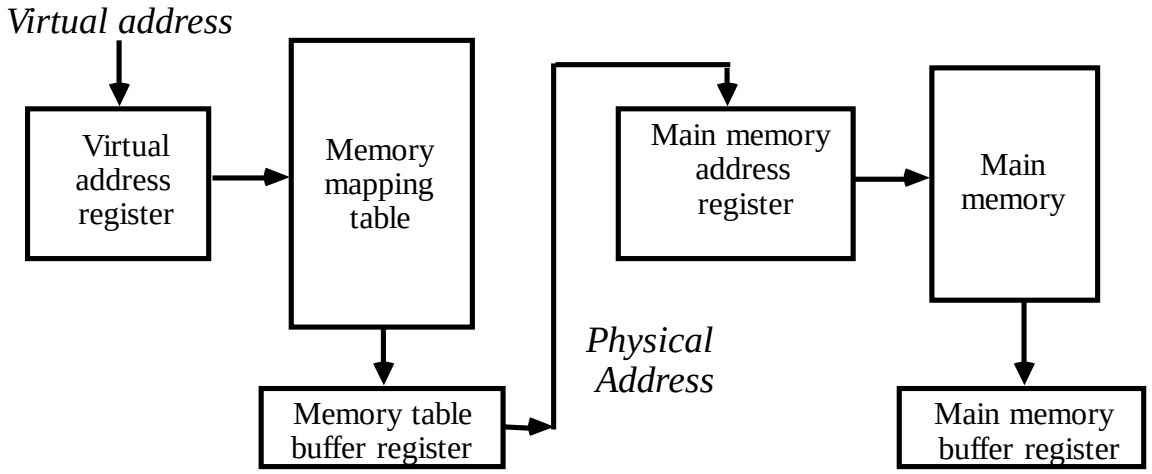
Give the programmer the illusion that the system has a very large memory, even though the computer actually has a relatively small main memory

Address Space(Logical) and Memory Space(Physical)



Address Mapping

Memory *Mapping Table* for *Virtual Address -> Physical Address*



ADDRESS MAPPING

Address Space and Memory Space are each divided into fixed size group of words called *blocks* or *pages*

1K words group

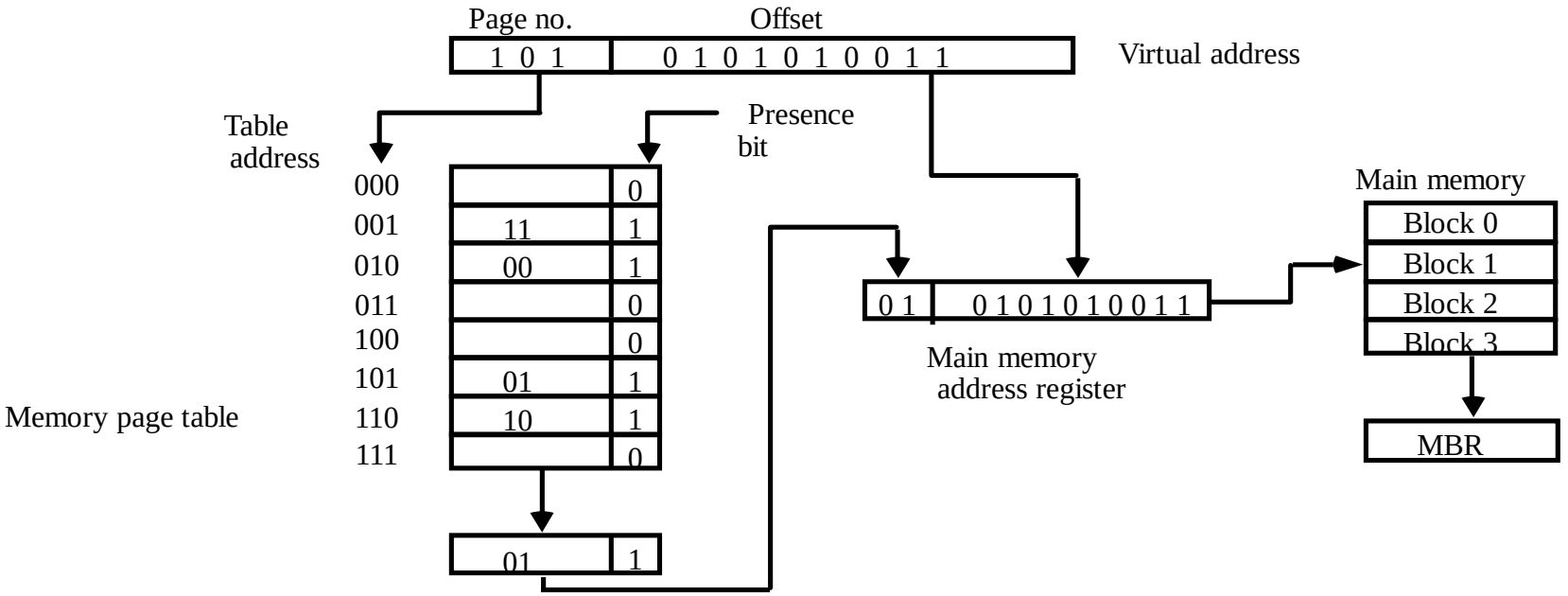
Address space
N = 8K = 2^{13}

Page 0
Page 1
Page 2
Page 3
Page 4
Page 5
Page 6
Page 7

Memory space
M = 4K = 2^{12}

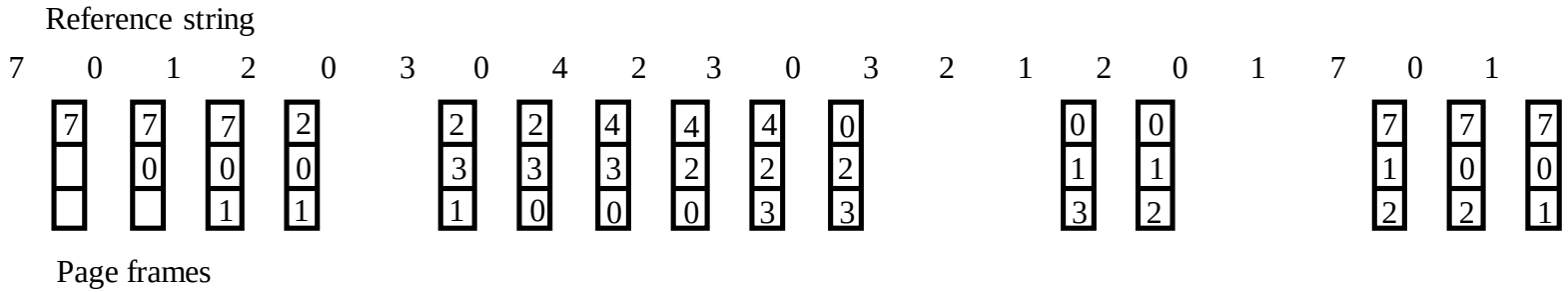
Block 0
Block 1
Block 2
Block 3

Organization of memory Mapping Table in a paged system



PAGE REPLACEMENT ALGORITHMS

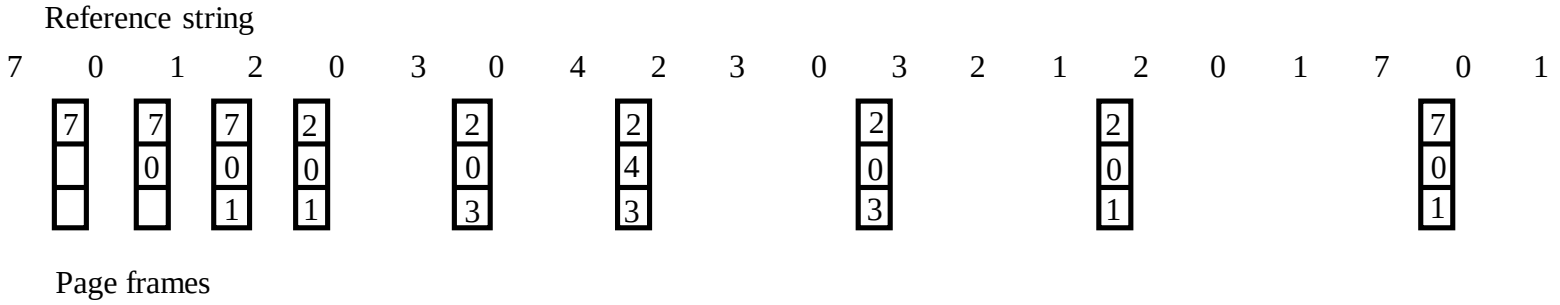
FIFO



FIFO algorithm selects the page that has been in memory the longest time

Optimal Replacement (OPT) - Lowest page fault rate of all algorithms

> Replace that page which will not be used for the longest period of time

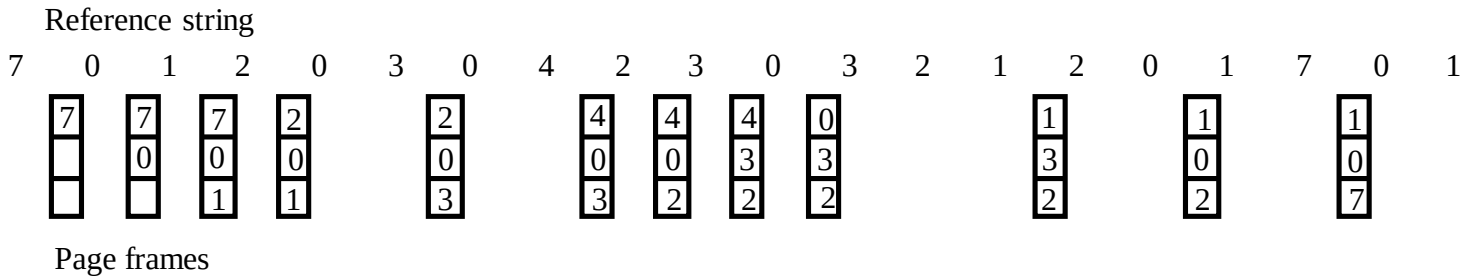


PAGE REPLACEMENT ALGORITHMS

LRU

- OPT is difficult to implement since it requires future knowledge

> Replace that page which has not been used for the longest period of time



Question

The page request are in following order. If there are three memory frames, show the execution of the following pages using

a. FIFO b. Optimal c. LRU

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

Input-Output Organization

- Peripheral Devices: Input & Output devices attached to computer are called Peripheral.
- Input and output devices communicate alphanumeric information by using ASCII 7bit code.
- To use computer efficiently, large number of programs and data must be prepared in advance for execution with computer.
- I/O interface provides a method for transferring information between internal storage and external I/O devices.

Input-Output Interface

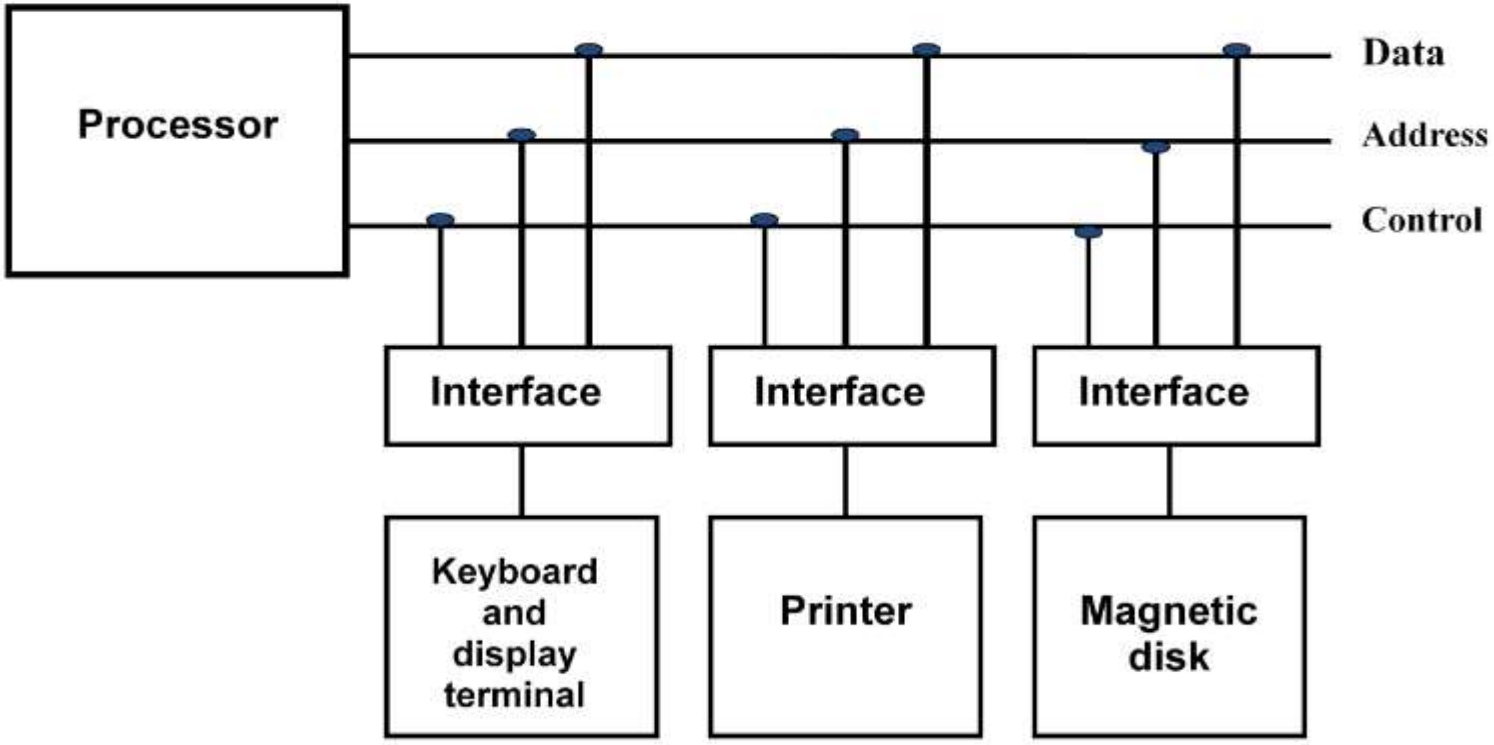
- The purpose of interfacing are as follows:
  - Peripherals are electromechanical & electromagnetic devices and are interacting with electronics devices(CPU).
  - Data transfer rate of peripherals is usually slower than transfer rate of CPU.
  - Data codes and formats in peripherals differ from word format in CPU
  - Operating modes of peripherals are different from each other and each one must be controlled without disturbing other.

Input-Output Interface

- To resolve these differences, computer system includes Interface units between CPU and peripherals to supervise and synchronize all input and output transfers.

- Each peripheral has its own controller that operates a particular electromechanical device.
- To communicate with a particular device, the processor places a device address on the address lines.

Input-Output Interface



Connection of I/O bus to input-output devices

Input-Output Interface

- When the interface detects it own address, it activates path between bus lines and the device.
- At the time address is made available in address lines, the processor provides function code(I/O command) in the control lines.
- Types of I/O command:
  - Control command: To activate and inform what to do.
  - Status command: To test various status conditions.
  - Data Output data: It causes the transfer of data from bus into one of its registers.
  - Data Input Command: Interface receives data from peripheral and places them on buffer register where it is put into data lines.

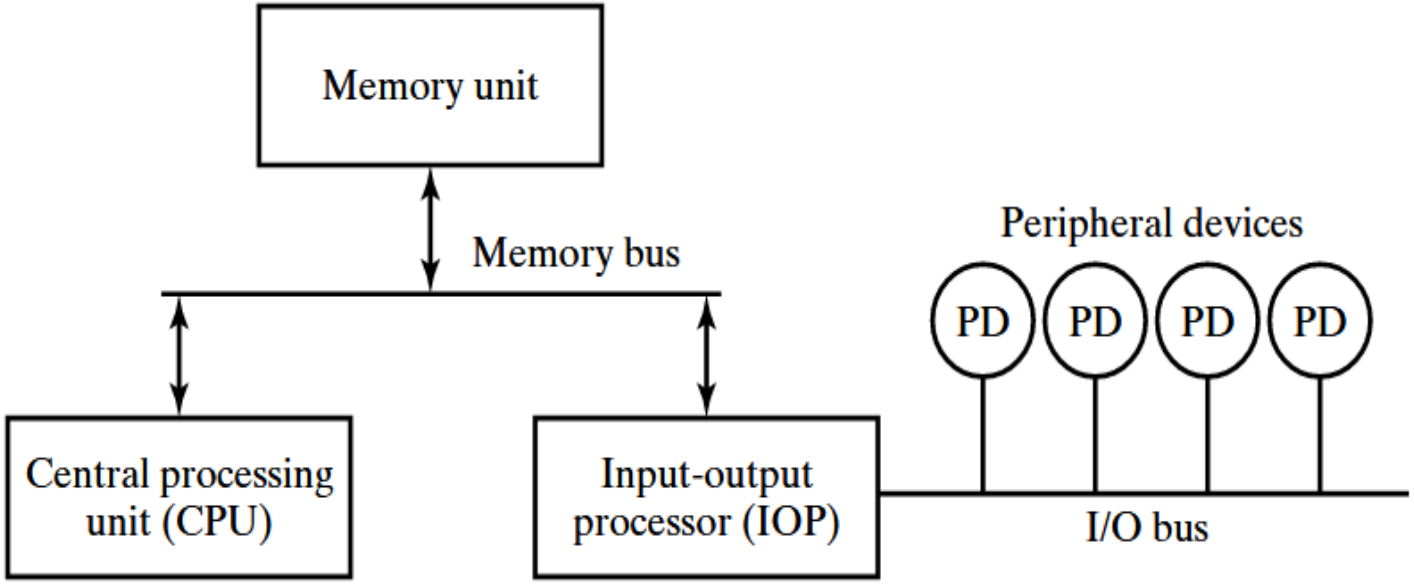
I/O versus Memory Bus

- In addition to communicating to I/O, processor must also communicate with memory unit.
- There are three ways that computer buses can be used to communicate with memory and I/O:
  - Use two separate buses, one for memory and one for I/O. (IOP)
  - Use one common bus for both memory and I/O but have separate control lines for each. (Isolated I/O)
  - Use one common bus for memory and I/O with common control lines. (Memory Mapped I/O)

Input-Output Transfer (IOP)

- An IOP takes care of input and output tasks.
- The CPU is assigned the task of initiating all operations, but I/O instructions are executed in IOP.
- When an I/O operation is required, the CPU informs the IOP where to find the I/O program and then leaves the transfer details to the IOP.
- IOP is also responsible of taking care of data synchronization, formats etc between CPU and I/O devices. In most computers CPU is master and IOP is slave.

Input-Output Transfer (IOP)



Isolated I/O

- The distinction between memory and I/O transfer is made through separate read and control lines.
- I/O read and I/O write are enabled during I/O transfer and Memory read/write are enabled during memory transfer.
- In the isolated I/O configuration, CPU have distinct input and output instructions where each of it will be associated with address of the interface register.
- When the CPU fetches and decodes the I/O instruction, it places the address associated with the instruction on the common address lines and enables I/O read or I/O write control line.

Isolated I/O

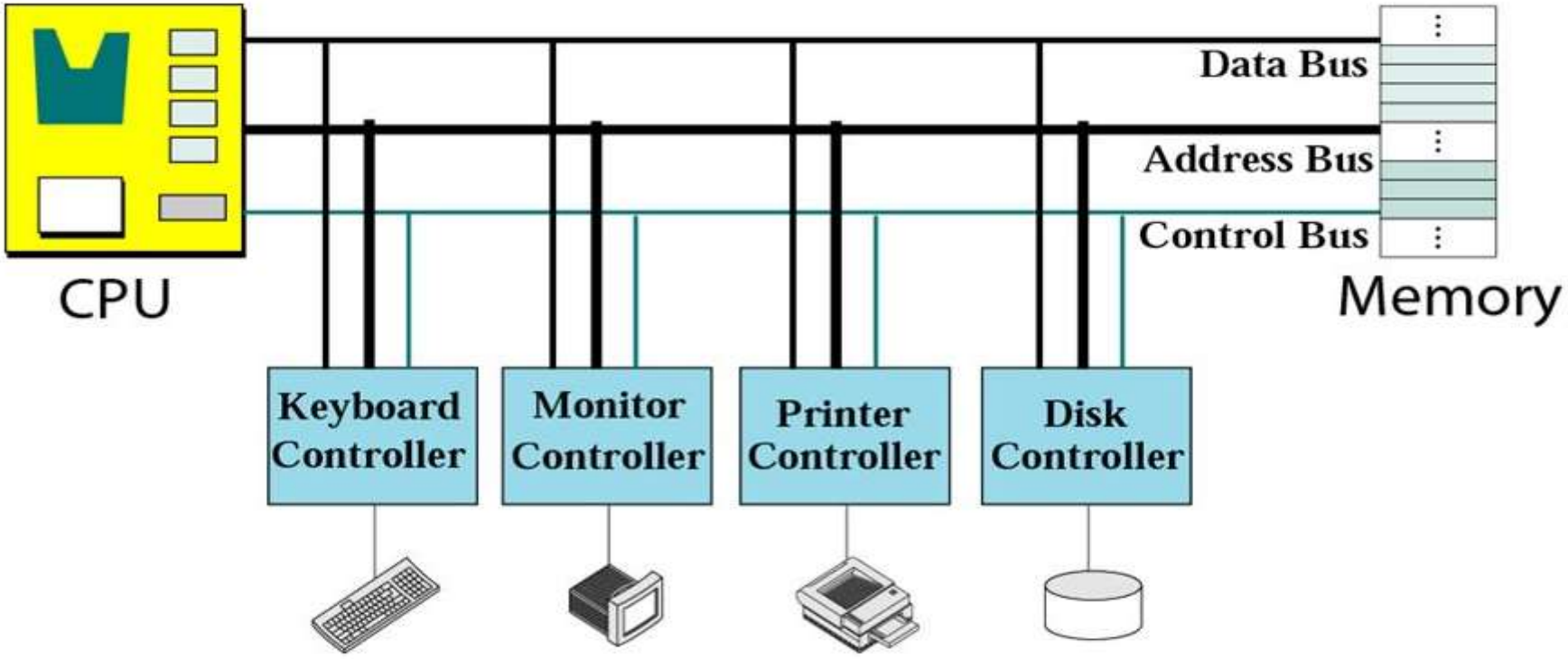
- When the CPU fetches and decodes the Memory instruction, it places the address associated with the instruction on the common address lines and enables Memory read or Memory write control line.
- The isolated I/O method isolates memory and I/O addresses.

Memory Mapped I/O

- In this configuration, same address space is used for both memory and I/O.
- The computer treats interface register (I/O) as a part of memory system.
- The assigned address cannot be used for storing memory words, which reduces memory address range available.
- In a memory-mapped I/O organization, there are no specific, input or output instruction.
- CPU manipulate I/O data residing in interface registers with the same instruction used to manipulate memory words.

Memory Mapped I/O

Connecting I/O devices to the buses



Modes of Transfer

- Data transfer between the CPU and the I/O devices may be handled in variety of modes. Some modes use the CPU as an intermediate path and others transfer the data directly to and from the memory unit.

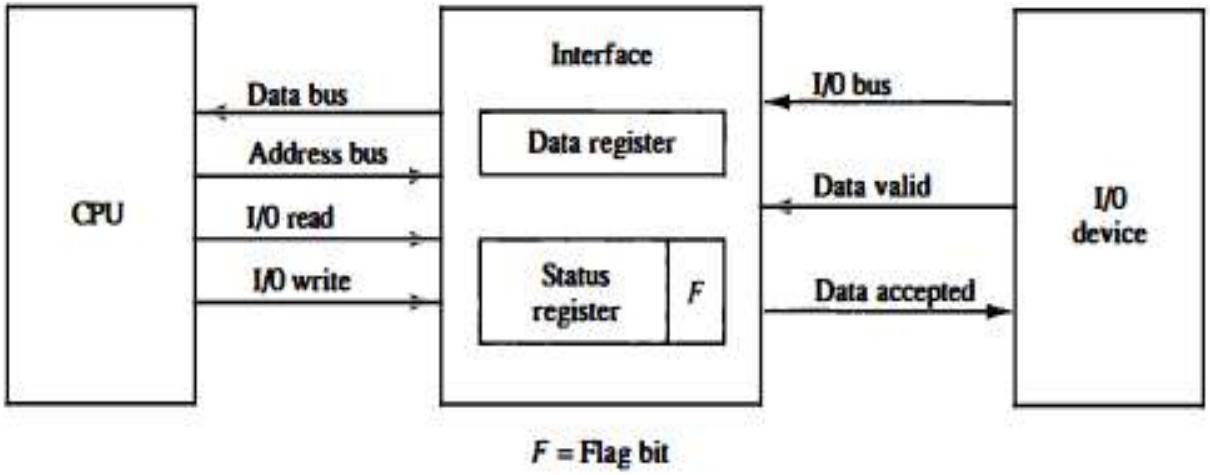
- Data transfer to and from peripherals may be handled in three ways:
  - Programmed I/O
  - Interrupt-initiated I/O
  - Direct Memory Access (DMA)

Programmed I/O

- Programmed I/O operations are the result of I/O instructions. Each data item transfer is initiate by an instruction in the program.
- Usually the transfer is to and from a CPU register and peripheral. Other instructions are needed to transfer data between memory and CPU.
- Once a data transfer is initiated, the CPU is required to monitor the interface to see when a transfer can again be made.

Programmed I/O

Figure 11-10 Data transfer from I/O device to CPU.



Interrupt Initiated I/O

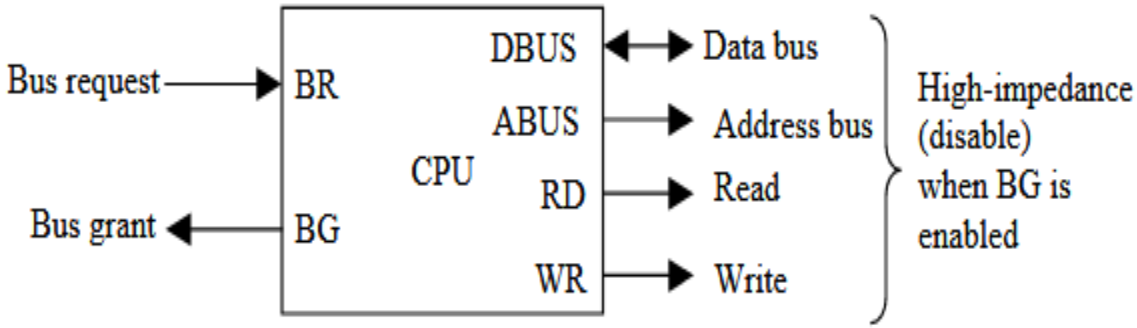
- In the programmed I/O CPU stays in program loop until the I/O indicates that it is ready for data transfer.
- This is time consuming process since it makes CPU busy needlessly.
- This can be avoided by using an interrupt facility.
- When the interface determines that device is ready for data transfer, it generates an interrupt request.
- Upon detecting external interrupt signal, the CPU momentarily stops the task it is processing.
- It then branches to fulfill the I/O request and return to the original task.

Direct Memory Access (DMA)

- The transfer of data between a fast storage device such as magnetic disk and memory often limited to the speed of CPU.
- Removing the CPU and letting the peripheral device manage the memory bus directly improve speed of transfer.
- Such transfer technique is called Direct Memory Access (DMA).
- A DMA controller takes over the buses to manage the transfer directly between I/O device and memory.
- During DMA transfer, the CPU is idle and has no control over memory buses.
- By using **Bus Request(BR)** and **Bus Grant(BG)** the buses are released to DMA controller.

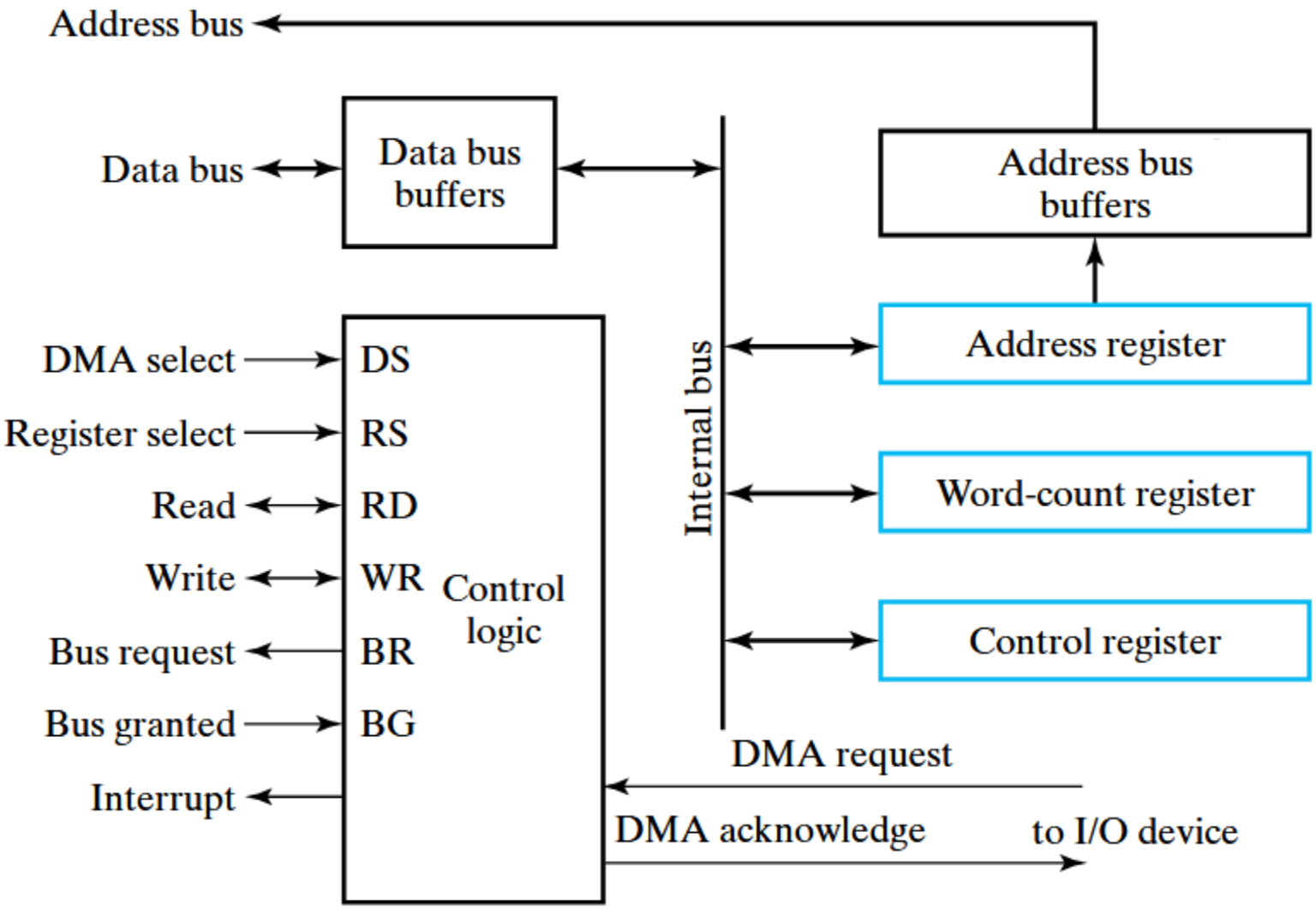
DMA

Figure 6.13 CPU bus signals for DMA transfer.



- Data transfer ways:
  - Burst Transfer: Here number of words are transferred in a block. Example: Magnetic disk.
  - Cycle stealing: Allows the DMA controller to transfer one data word at a time after it must return the control of buses to CPU.

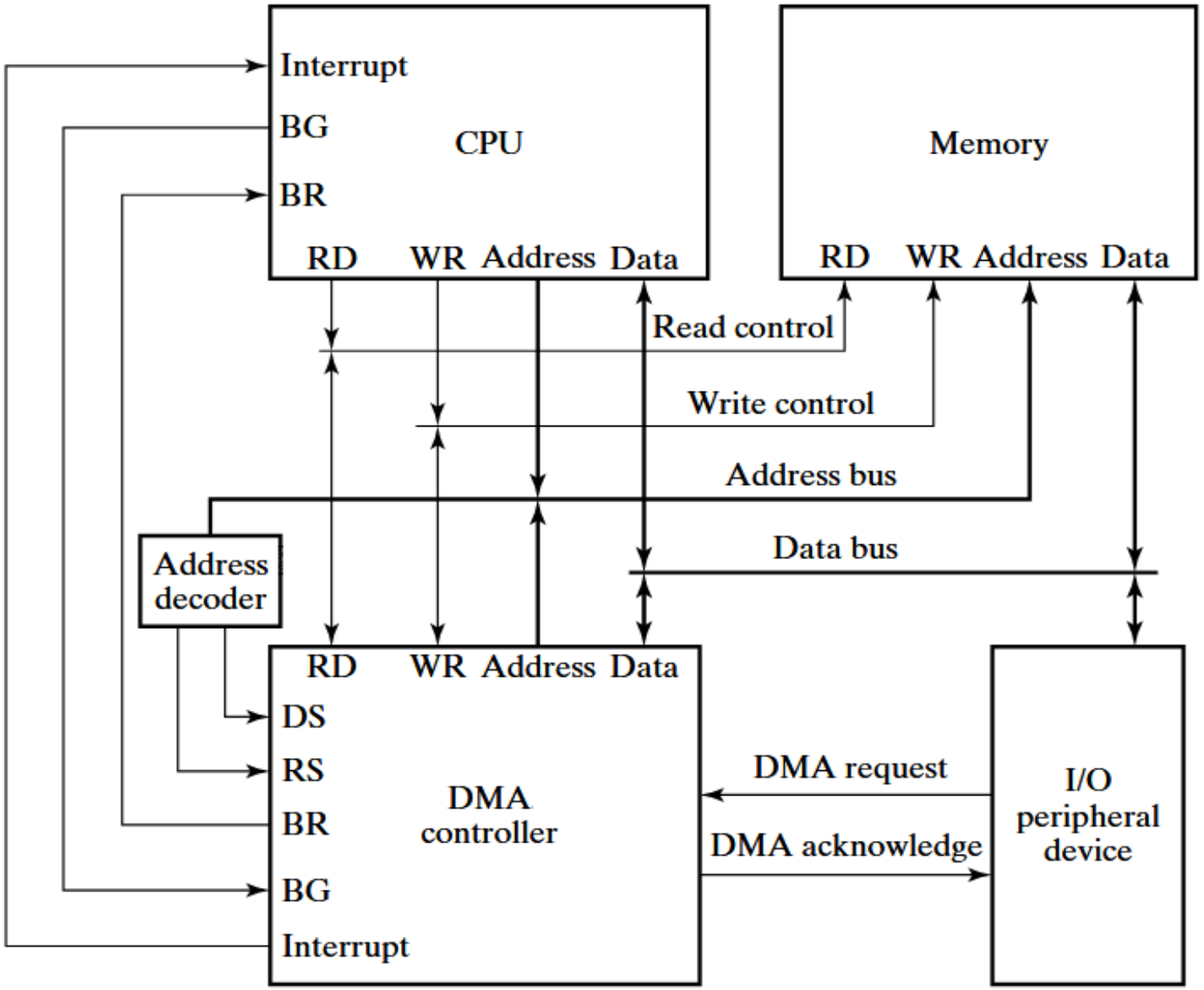
DMA Controller



Initialization of DMA

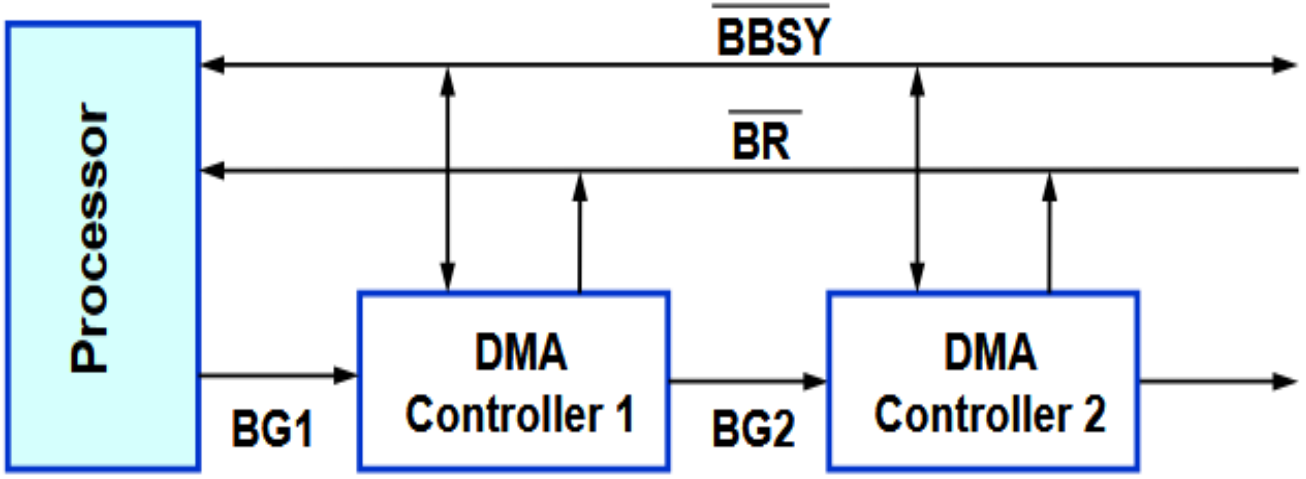
- The CPU initializes the DMA by sending the following information through the data bus.
  - The starting address of the memory block where data are available(for read) or where data are to be stored(for write).
  - The word count, which is the number of words in the memory block.
  - Control to specify the mode of transfer such as read or write.
  - A control to start the DMA transfer.

DMA Transfer in a computer system



Bus Arbitration

- The device that is allowed to initiate data transfers on the bus at any given time is called Bus master.
- Bus arbitration is the process by which the next device becomes Bus master and will do the data transfer.
- Two approaches: Centralized Arbitration and Distributed Arbitration.
  - **Centralized Arbitration**



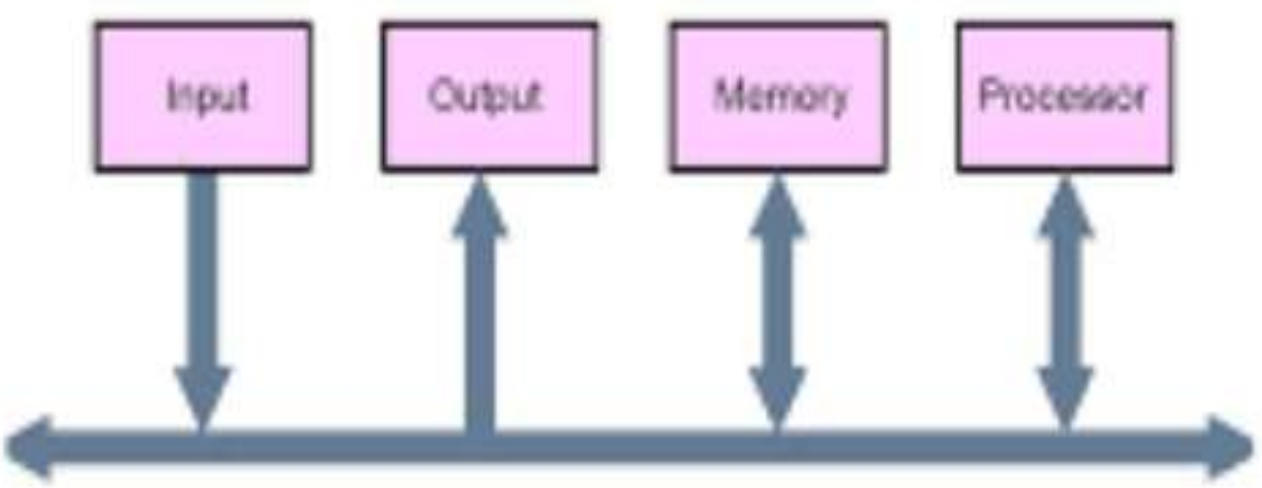
Types of Interrupt

- Major types of Interrupt are:
  - External Interrupt: It comes from I/O devices, from timing device, or from any other external source.
  - Internal Interrupt: It includes register overflow, invalid operation code, stack overflow etc.
  - Software Interrupt or Hardware Interrupt: External and Internal interrupts are initiated from signals that occur in the hardware of the CPU. *A software interrupt is initiated by executing an instruction.*
  - Priority Interrupt: In case several sources will request service simultaneously, in this case system must decide which device to service first. For ex: Polling, Daisy Chain.

Buses

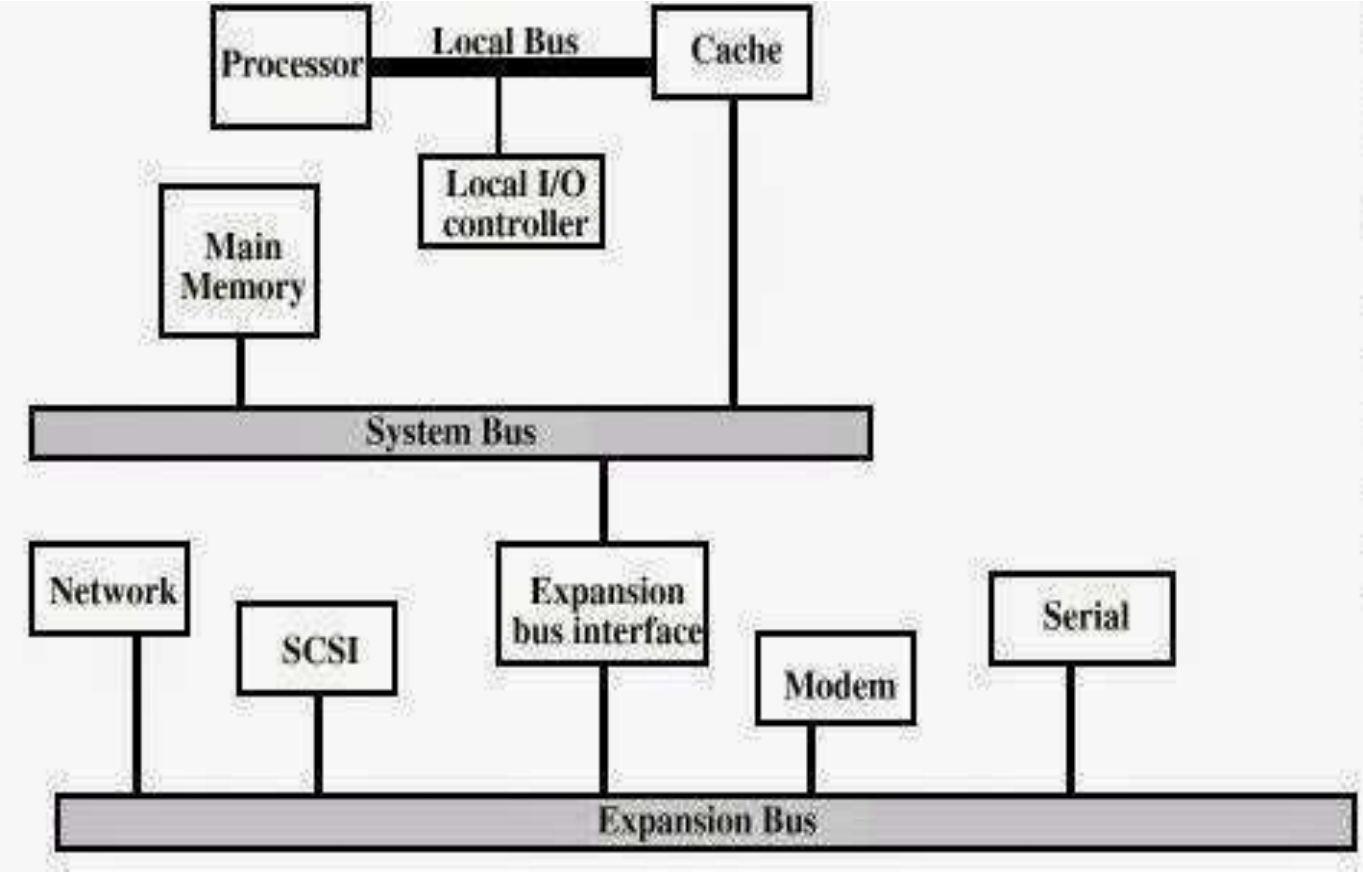
- Data Bus: Bi-directional and transfers data.
- Address Bus: Uni-directional and sends the address.
- Control Bus: R/W, BR,BG etc.

- Bus Structure: Single bus



Buses

- Bus Structure: Multi bus



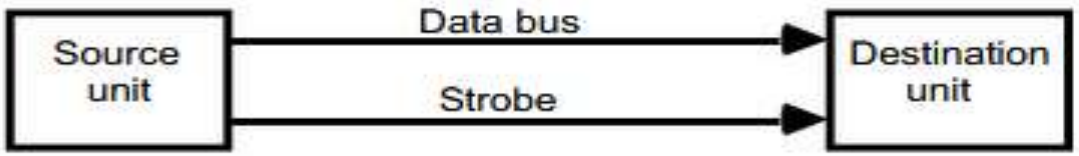
Synchronous or Asynchronous Data Transfer

- Internal operations in a digital system are synchronized by means of clock pulses supplied by common pulse generator.
- If the registers in the interface share a common clock with the CPU registers, the transfer is synchronous.
- Asynchronous data transfer requires that control signals be transmitted between communicating units to indicate the time at which data is being transmitted.
- Asynchronous data transfer can be accomplished by Strobe & Handshaking.

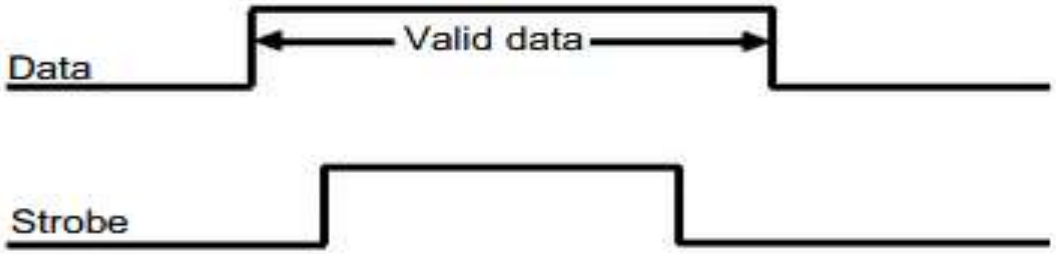
Strobe Control

- Strobe control:
  - It employs single control line.
  - It can be activated either by source or destination unit.

Block Diagram

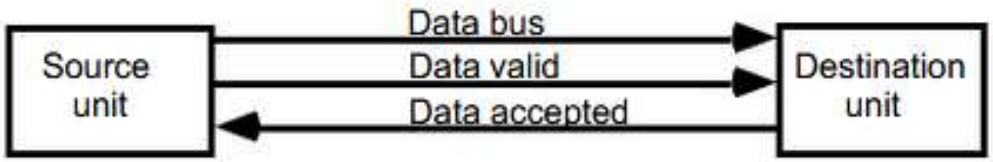


Timing Diagram

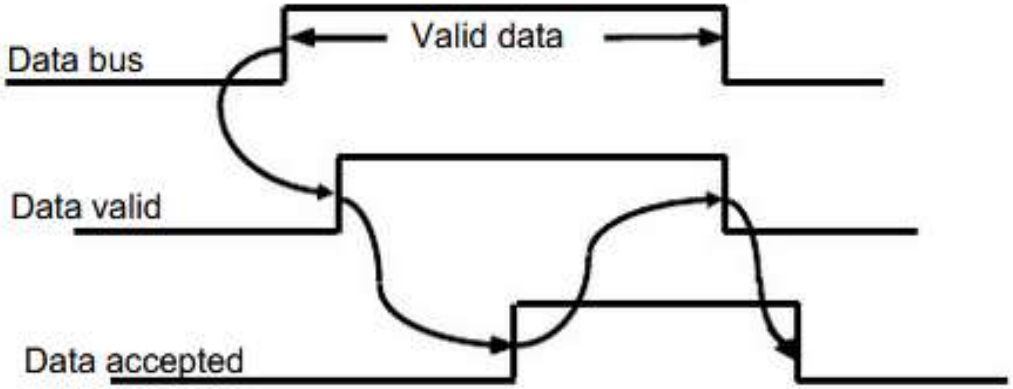


Handshaking

Block Diagram



Timing Diagram



Sequence of Events