Computer Organization & Architecture

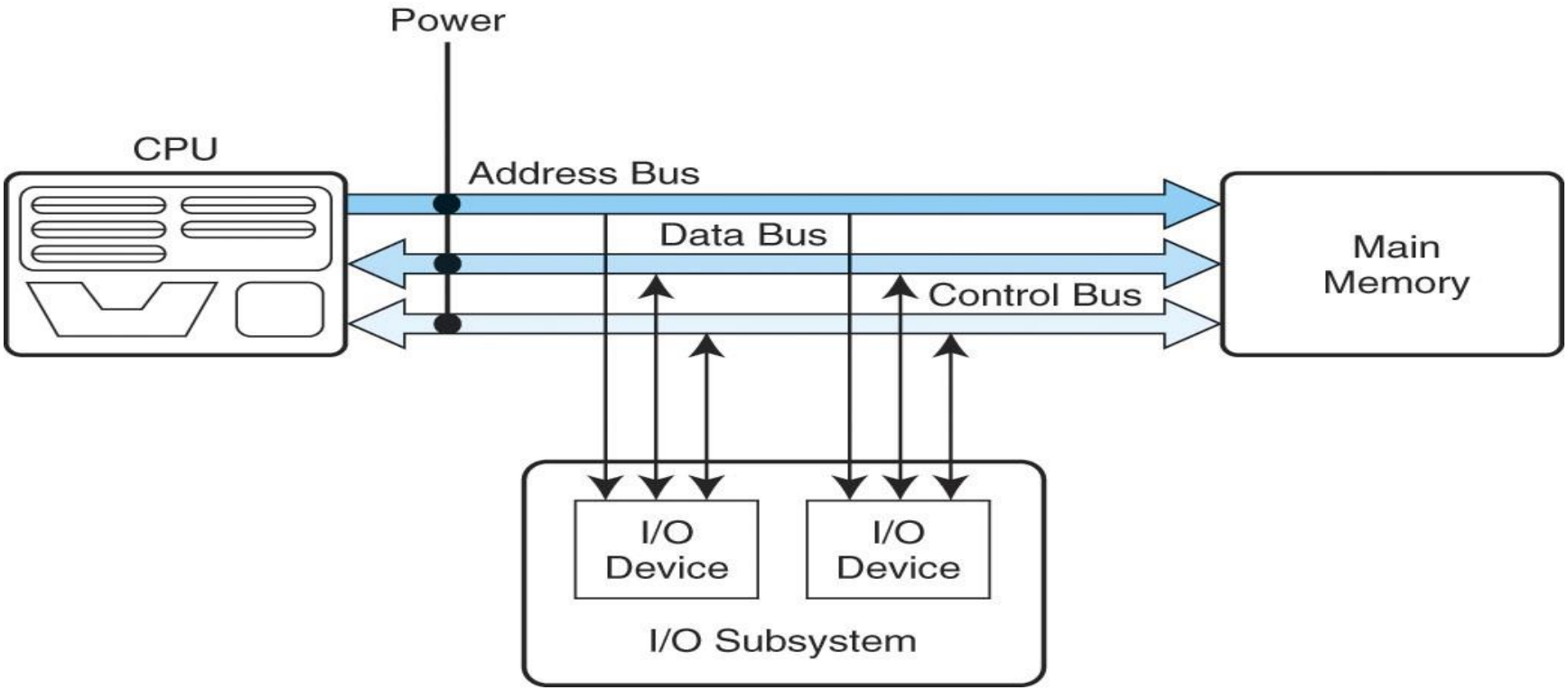
Unit-1 Notes

By

Dr. Lalit Saraswat

System bus

- *A bus* in the computing is the set of physical connection(cable, printed circuits, ...) which can be share by multiple hardware components in order to communicate with one another.



ADDRESS BUS

- Computer bus (a series of lines connecting two or more devices) used to specify a physical address
- Consists of all the signals necessary to define any of the possible memory address locations within the computer, or for modular memories any of the possible memory address locations within a module
- Defined as a label, symbol, or other set of characters used to designate a location or register where information is stored

ADDRESS BUS (cont)

- An address must be transmitted to memory over the address bus before data or instructions can be written into or read from memory by the CPU or I/O sections
- The width of the address bus determines the amount of memory a system can address
- Example:
  - a system with a 32-bit address address 232 (4,294,967,296) memory locations.

DATA BUS

- Function: the bidirectional data bus, handles the transfer of all data and instructions between functional areas of the computer.
- Only transmit in one direction at a time.
- It carries data (operands) to and from the CPU and memory and input output.

Control Bus

- Control the access to and the use of the data lines and address lines
- Control signals:
  - memory read/write
  - I/O read/write
  - transfer ACK
  - bus request/grant
  - interrupt request/ACK
  - clock & reset

Bus arbitration

- Bus Arbitration is the procedure in bus communication that chooses between connected devices contending for control of the shared bus;
- the device currently in control of the bus is often termed the *bus master.*
- Devices may be allocated differing priority levels that will determine the choice of bus master in case of contention
- A device not currently bus master must request control of the bus before attempting to initiate a data transfer via the bus.

Types of Bus Arbitration

- **1. Centralized Arbitration**
- In centralized bus arbitration, a single bus arbiter performs the required arbitration. The bus arbiter may be the processor or a separate controller connected to the bus.
- There are three different arbitration schemes that use the centralized bus arbitration approach. There schemes are:
- a. Daisy chaining
- b. Polling method
- c. Independent request

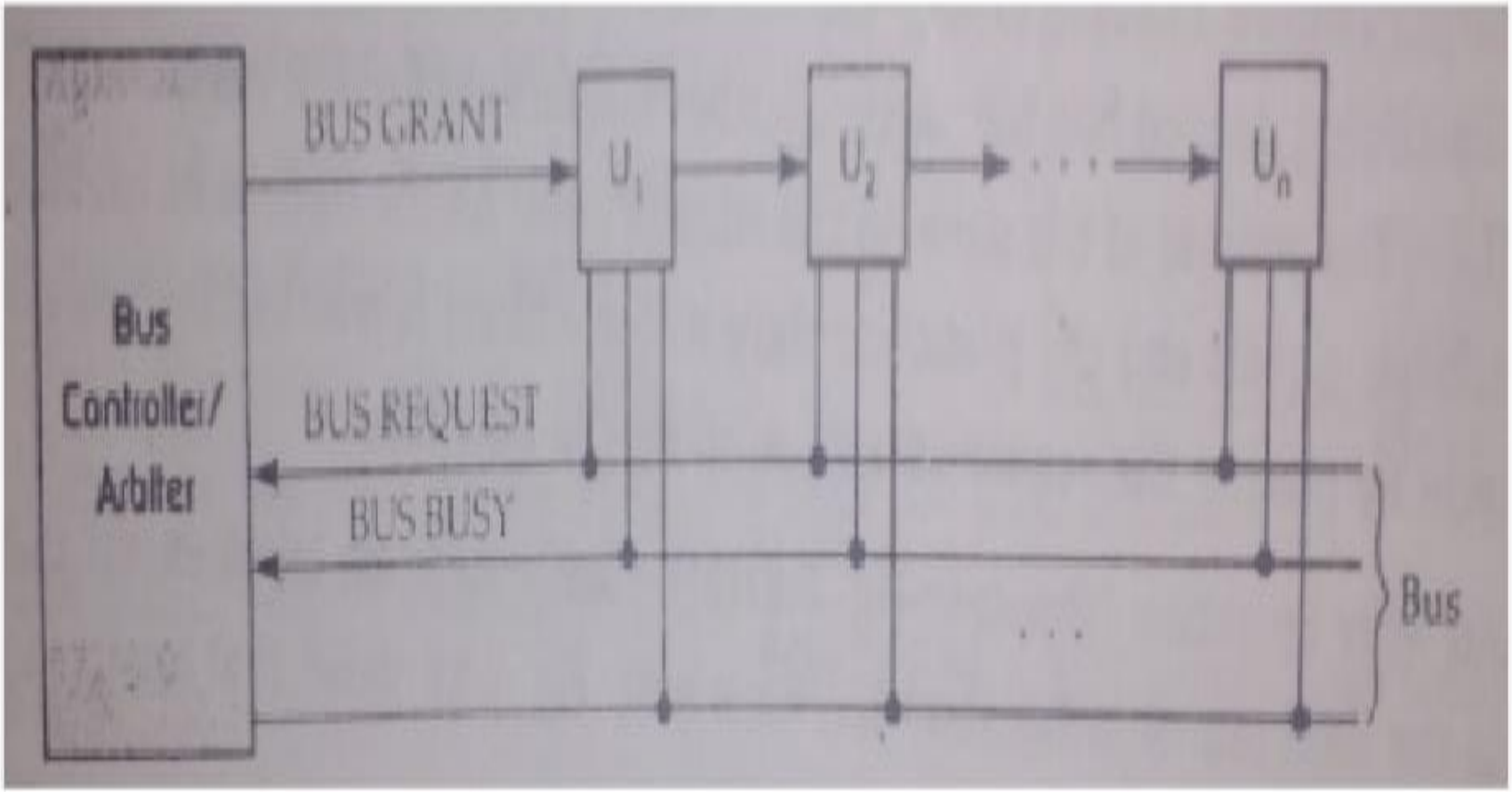
a) Daisy chaining

- It is simple and cheaper method. All masters make use of the same line for bus request.
- In response to the bus request the controller sends a bus grant if the bus is free.
- The bus grant signal serially propagates through each master until it encounters the first one that is requesting access to the bus. This master blocks the propagation of the bus grant signal, activities the busy line and gains control of the bus.
- Therefore any other requesting module will not receive the grant signal and hence cannot get the bus access

a) Daisy chaining

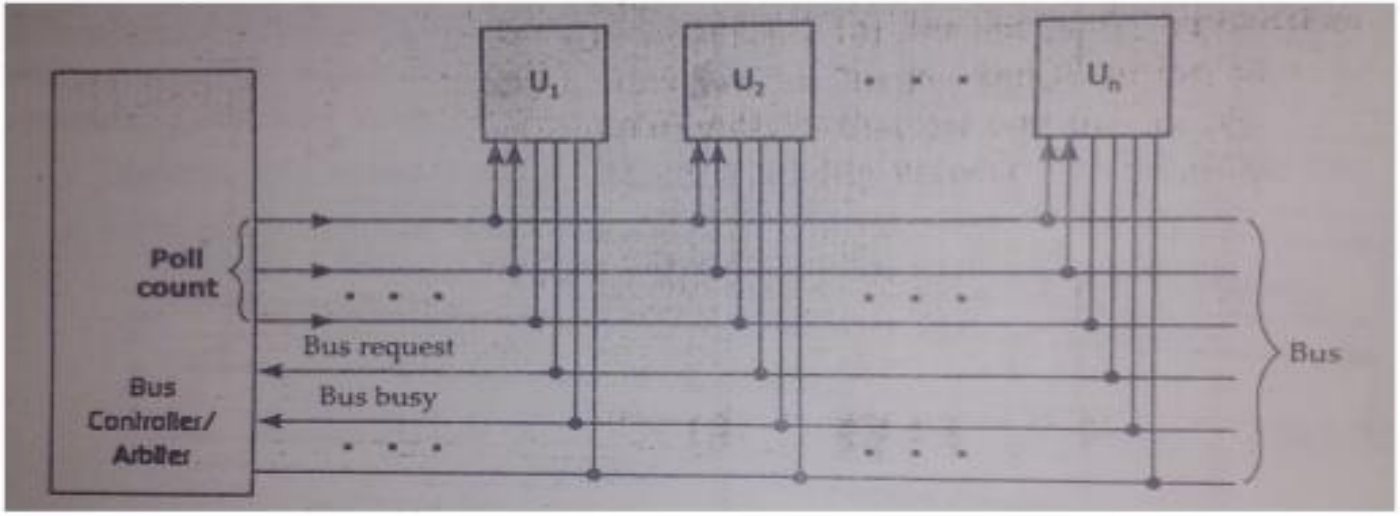
- **Advantages –**
- Simplicity and Scalability.
- The user can add more devices anywhere along the chain, up to a certain maximum value.
- **Disadvantages –**
- The value of priority assigned to a device is depends on the position of master bus.
- Propagation delay is arises in this method.
- If one device fails then entire system will stop working.

a) Daisy chaining



b) Polling method

Polling



- This method replaces the BUS GRANT line of daisy chain method with a set of poll count lines that are connected directly to all devices on the bus.
- Devices request access to the bus via a common BUS REQUEST line.
- Bus controller generates a sequence of numbers on the poll count lines.
- Each device compares these numbers as their device address already assigned to them.
- When a requesting device finds that its address matches the numbers on the poll-count lines, the device activates BUS BUSY.
- The bus controller responds by terminating the polling process and the device connects to the bus.

Advantages –

•This method does not favor any particular device and processor.

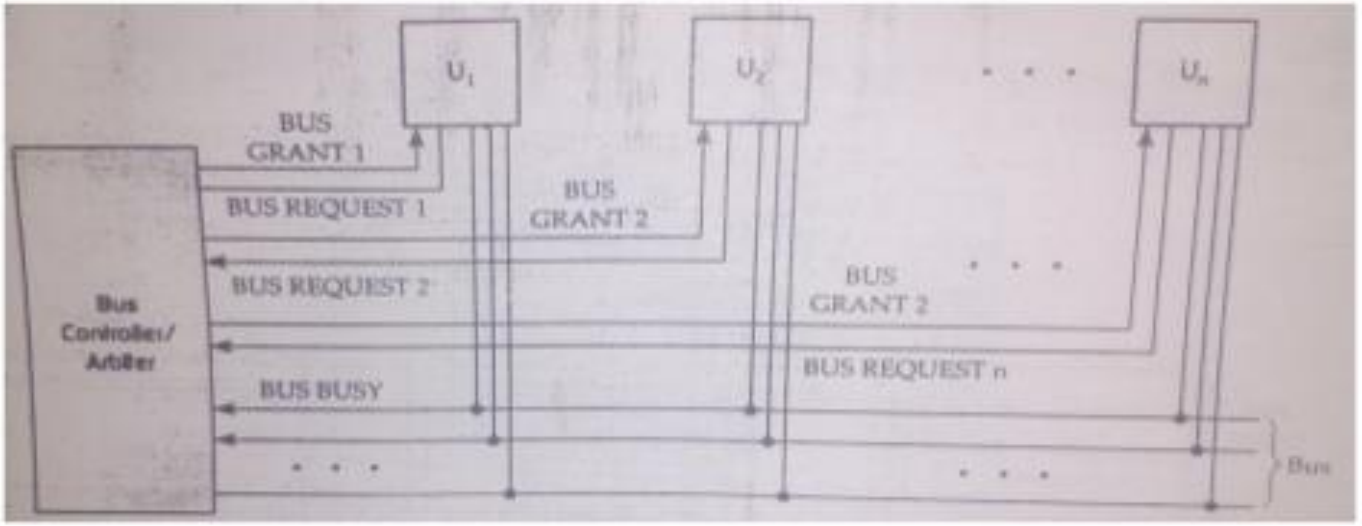
•The method is also quite simple.

•If one device fails then entire system will not stop working.

Disadvantages –

•Adding bus masters is different as increases the number of address lines of the circuit.

Independent Requesting



- There are separate BUS REQUEST and BUS GRANT lines for every device that are sharing bus.
- In this, bus controller has the capability of immediate identifying all the requesting devices.
- Bus controller responds rapidly to the request by determining the highest priority device that has sent the bus request.
- This priority is programmable and is predetermined.

c) Independent request

- In this scheme each master has a separate pair of bus request and bus grant lines and each pair has a priority assigned to it.
- The built in priority decoder within the controller selects the highest priority request and asserts the corresponding bus grant signal.

2. Distributed Arbitration

- In distributed arbitration, all devices participate in the selection of the next bus master.
- In this scheme each device on the bus is assigned a4-bit identification number.
- The decentralized arbitration offers high reliability because operation of the bus is not dependent on any single device.

Register Transfer Language:

- A digital system is an interconnection of digital hardware modules that perform a specific task.
- The modules are constructed from digital components such as registers, decoders, arithmetic elements, and control logic.
- These modules are interconnected with common data and control paths to form a digital computer.
- The operations executed on data stored in registers are called microoperations.
- A microoperation is an elementary operation performed on the information stored in one or more registers.
- For any function of the computer, a sequence of microoperations is used to describe it
- The result of the operation may be
  - replace the previous binary information of a register or
  - transferred to another register

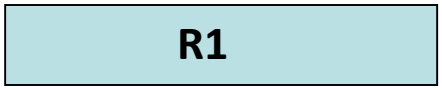
  - Examples of microoperations are shift, count, clear and load.

Register Transfer Language:

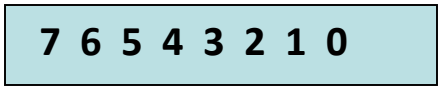
- The internal hardware organization of a digital computer is defined by specifying:
  - The set of registers it contains and their function.
  - The sequence of microoperations performed on the binary information stored in the registers.
  - The control that initiates the sequence of microoperations.
- The symbolic notation used to describe the microoperation transfers among registers is called a register transfer language.
- The term register transfer implies the availability of hardware logic circuit that can transfer the result of the operation to the same or another register.
- A register transfer language (RTL) is a system for expressing in symbolic form the microoperation sequences among the register of a digital module.
- Computer registers are designated by capital letters (sometimes followed by numerals) to denote the function of the register
  - R1: processor register (general purpose register)
  - MAR: Memory Address Register (holds an address for a memory unit)
  - PC: Program Counter
  - IR: Instruction Register
  - SR: Status Register

Register Transfer:

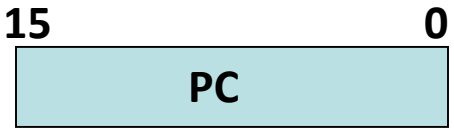
❖ The individual flip-flops in an n-bit register are numbered in sequence from 0 to n-1 (from the right position toward the left position).



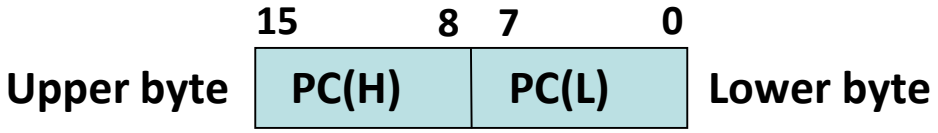
Register R1



Showing individual bits



Numbering of bits



Partitioned into two parts

❖ Information transfer from one register to another is described by a replacement operator:

R2 ← R1

❖ This statement denotes a transfer of the content of register R1 into register R2.

❖ The transfer happens in one clock cycle.

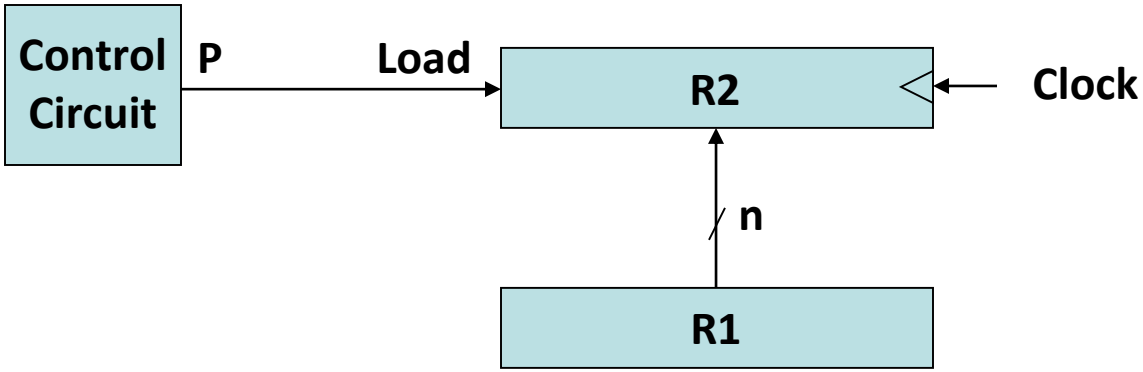
❖ The content of the R1 (source) does not change.

❖ The content of the R2 (destination) will be lost and replaced by the new data transferred from R1.

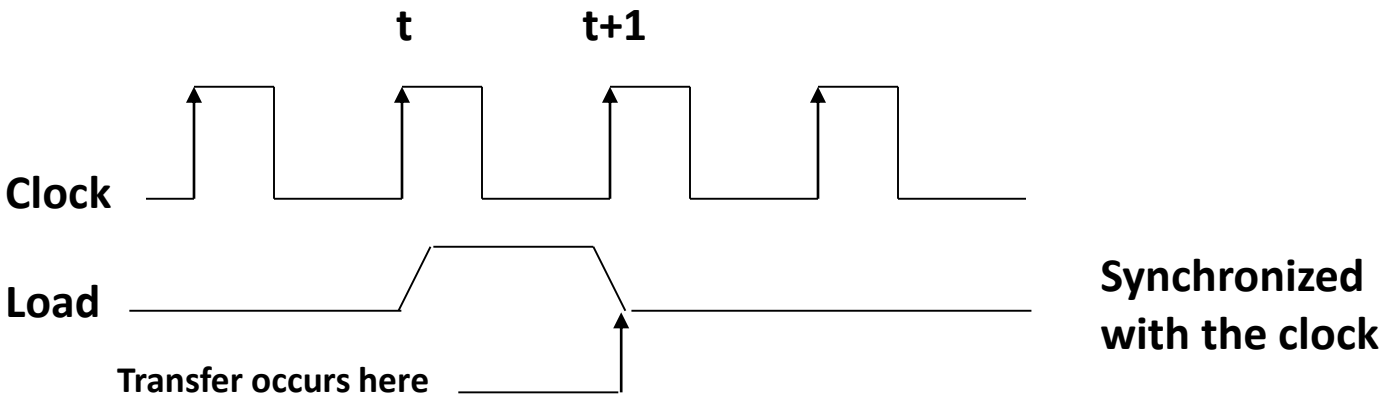
Register Transfer:

- Register transfer statement implies that the circuits are available from the outputs of the source register to the inputs of the destination register, and that the destination register has a parallel load capability.
- Conditional transfer occurs only under a control condition
- This can be shown by means of an if-then statement:
  - If (P = 1) then (R2 ← R1)
  - where P is control signal generated in the control section.
- The control variables are separated from the register transfer operation by specifying a control function.
- A control function is a Boolean variable that is equal to 1 or 0.
- The control function is included in the statement as follows:
  - P: R2 ← R1
- The control condition is terminated with a colon.
- It symbolizes that the transfer operation takes place by the hardware only if P = 1.

Register Transfer:



Block Diagram:



Timing Diagram:

Register Transfer:

- The statement
  - T: R2 ← R1, R1 ← R2

denotes an operation that exchanges the contents of two registers during one common clock pulse provided that T = 1.

This simultaneous operation is possible with registers that have edge-triggered flip-flops.

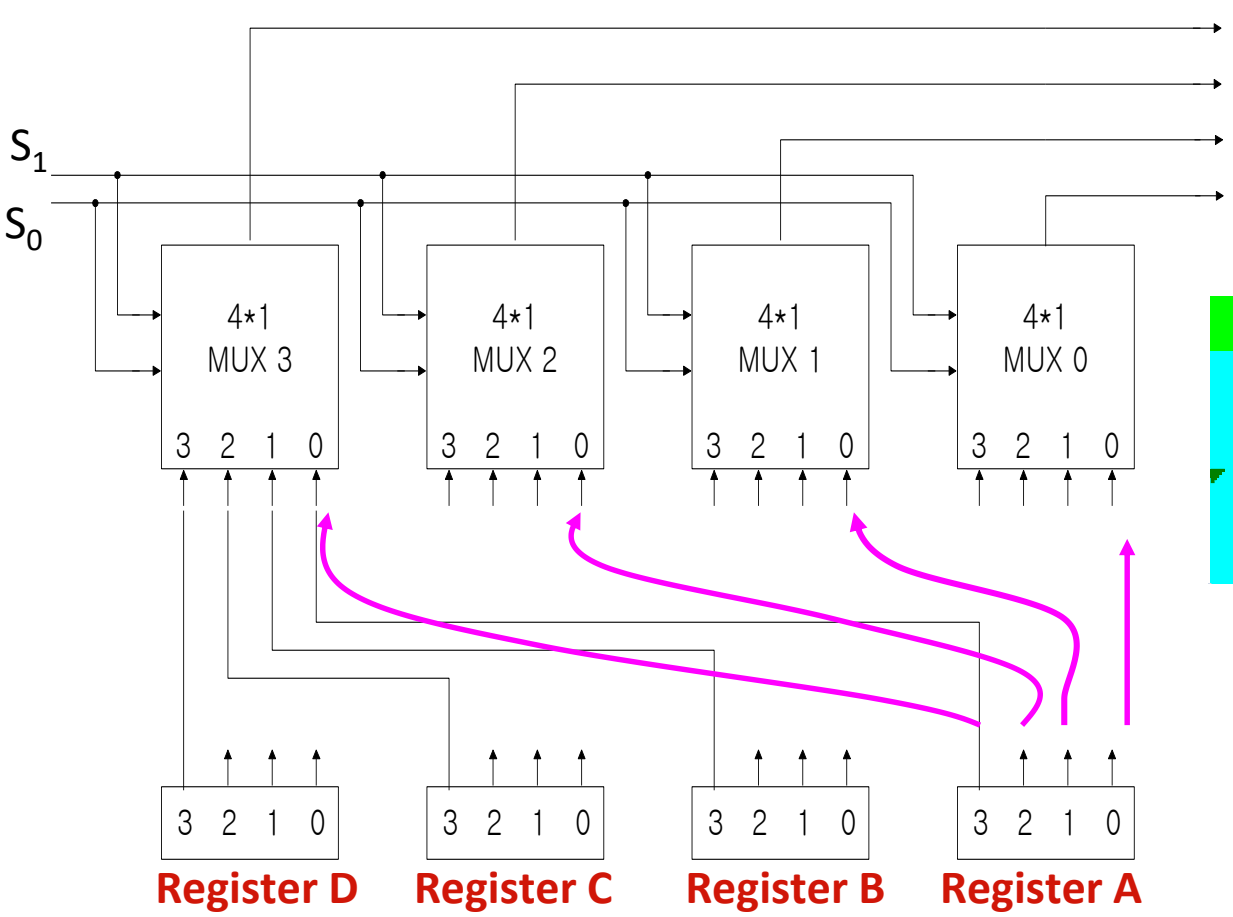
Basic Symbols for Register Transfer:

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

Bus and Memory Transfers:

- A typical digital computer has many registers, and paths must be provided to transfer information from one register to another.
- The number of wires will be excessive if separate lines are used between each register and all other registers in the system.
- The more efficient scheme for transferring information between registers in a multiple-register system is a common bus system.
- A bus structure consists of a set of common lines, one for each bit of a register, through which binary information is transferred one at a time.
- Control signals determine which register is selected by the bus during each particular register transfer.
- Common bus system can be constructed either by using multiplexers or by using three-state buffers.

Bus System for Four Registers using Multiplexers:



S1	S0	Register selected
0	0	A
0	1	B
1	0	C
1	1	D

- BUS ← C
- R1 ← BUS
- R1 ← C

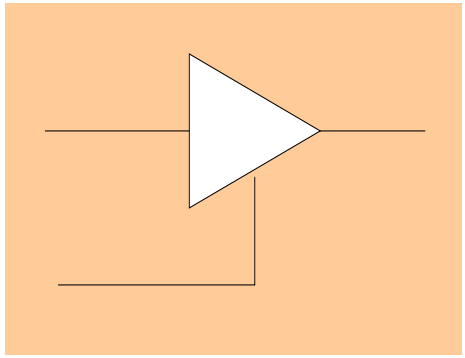
❖ A bus system will multiplex k registers of n bits each to produce an n-line common bus.

❖ The number of multiplexers needed to construct the bus is equal to n, the number of bits in each register.

❖ The size of each multiplexer must be k X 1, since it multiplexes k data lines.

Bus System for Four Registers using Three-State Buffers:

- A three state gate is a digital circuit that exhibits three states.
  - Logic 1 State
  - Logic 0 State
  - High Impedance State Z (Open Circuit)
- Buffer --- A device designed to be inserted between other devices to match impedance, to prevent mixed interactions, and to supply additional drive or relay capability
- Buffer types are classified as inverting or non-inverting.



If C=1, Output Y = A

If C=0, Output = High-impedance

Memory Transfer:

$$READ: \quad DR \leftarrow M[AR]$$
$$WRITE: \quad M[AR] \leftarrow R1$$

→ AR: Address Register.
DR: Data Register.
M : Memory Word (Data)

❖ **Memory Read : A transfer information into DR from the memory word M selected by the address in AR**

❖ **Memory Write : A transfer information from R1 into the memory word M selected by the address in AR**

Microoperations:

- A microoperation is an elementary operation performed with the data stored in registers.
- The microoperations performed in digital systems are classified into four categories:
  - Register transfer microoperations --- transfer binary information one register to another.
  - Arithmetic microoperations --- perform arithmetic operation on numeric data stored in registers.
  - Logic microoperations --- perform bit manipulation operations on numeric data stored in registers.
  - Shift microoperations --- perform shift operations on data stored in registers.

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
$R2 \leftarrow \overline{R2}$	Compliment the contents of R2 (1's compliment)
$R2 \leftarrow \overline{R2} + 1$	2's compliment the contents of R2 (negative)
$R3 \leftarrow R1 + \overline{R2} + 1$	R1 plus the 2's compliment of R2 (subtraction)
$R1 \leftarrow R1 + 1$	Increment the content of R1 by one
$R1 \leftarrow R1 - 1$	Decrement the content of R1 by one

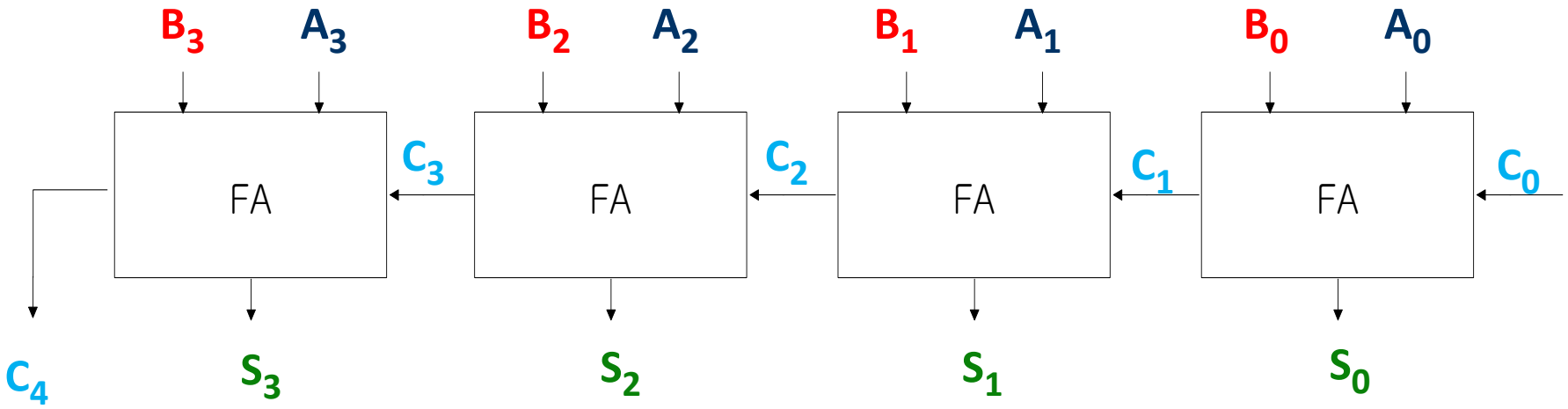
4-bit Binary Adder:

C_4 C_3 C_2 C_1 C_0

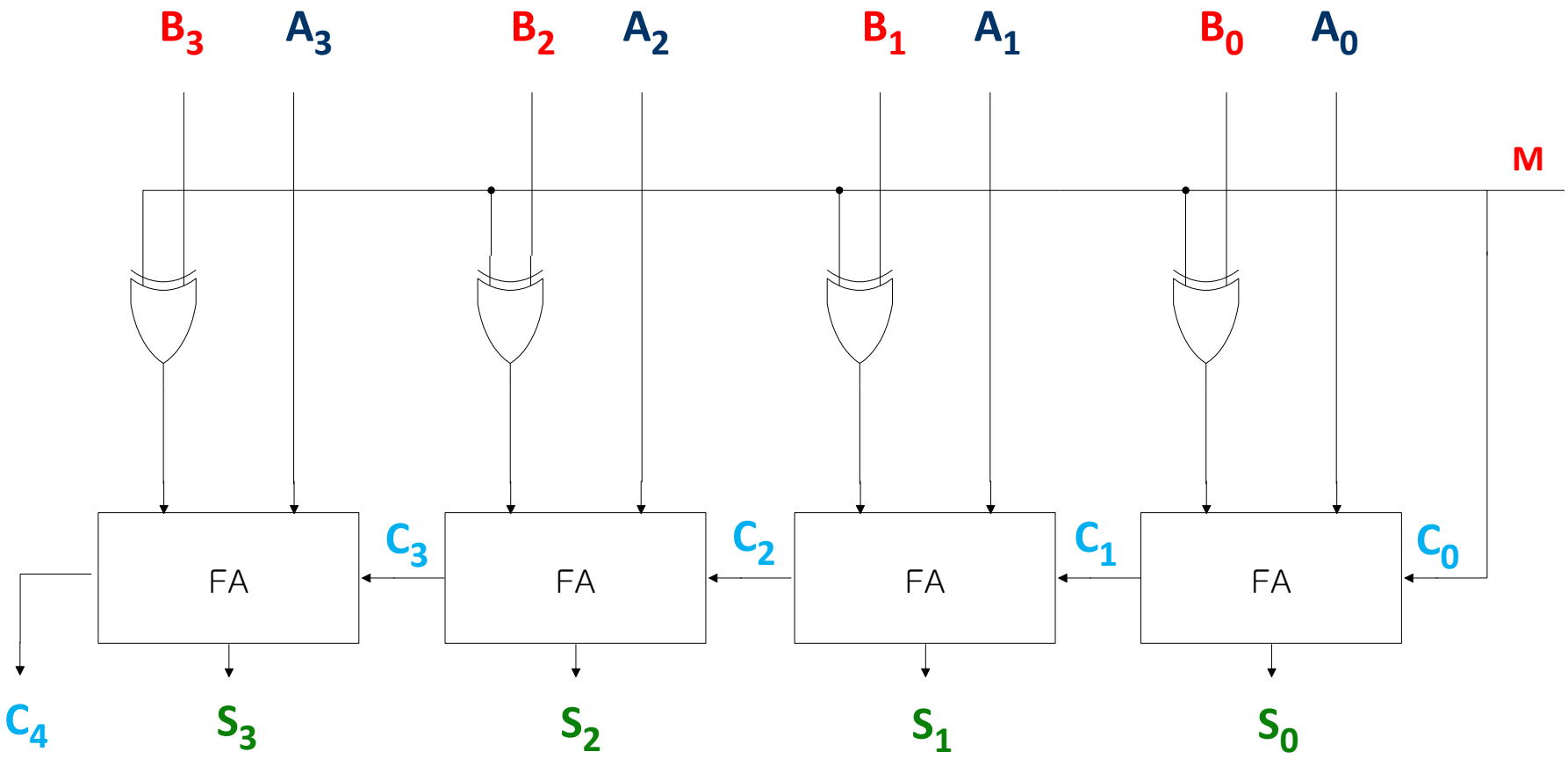
A_3 A_2 A_1 A_0

B_3 B_2 B_1 B_0

S_3 S_2 S_1 S_0



4-bit Binary Adder-Subtractor:



4-bit Binary Adder-Subtractor:

M = 0

C_4 C_3 C_2 C_1 C_0

A_3 A_2 A_1 A_0

B_3 B_2 B_1 B_0

S_3 S_2 S_1 S_0

A + B

M = 1

C_4 C_3 C_2 C_1 C_0

A_3 A_2 A_1 A_0

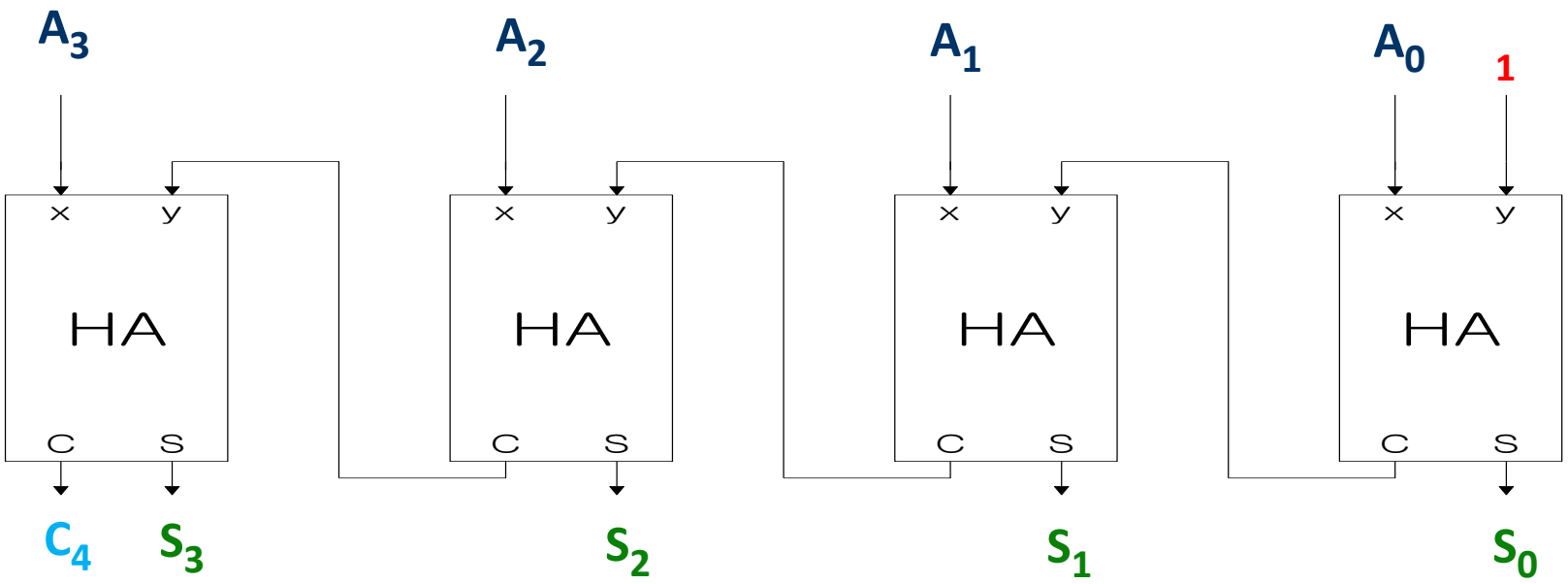
B_3' B_2' B_1' B_0'

S_3 S_2 S_1 S_0

A - B

4-bit Binary Incrementer:

C_4	C_3	C_2	C_1	
	A_3	A_2	A_1	A_0
				1
	S_3	S_2	S_1	S_0



Logic Microoperations:

❖ Logic microoperations specify binary operations for strings of bits stored in registers.

❖ These operations consider each bit of the register separately and treat them as binary variables.

❖ Example: $P: R1 \leftarrow R1 \oplus R2$

```
  1010  Content of R1
+ 1100  Content of R2
  0110  Content of R1 after P=1
```

❖ Special Symbols

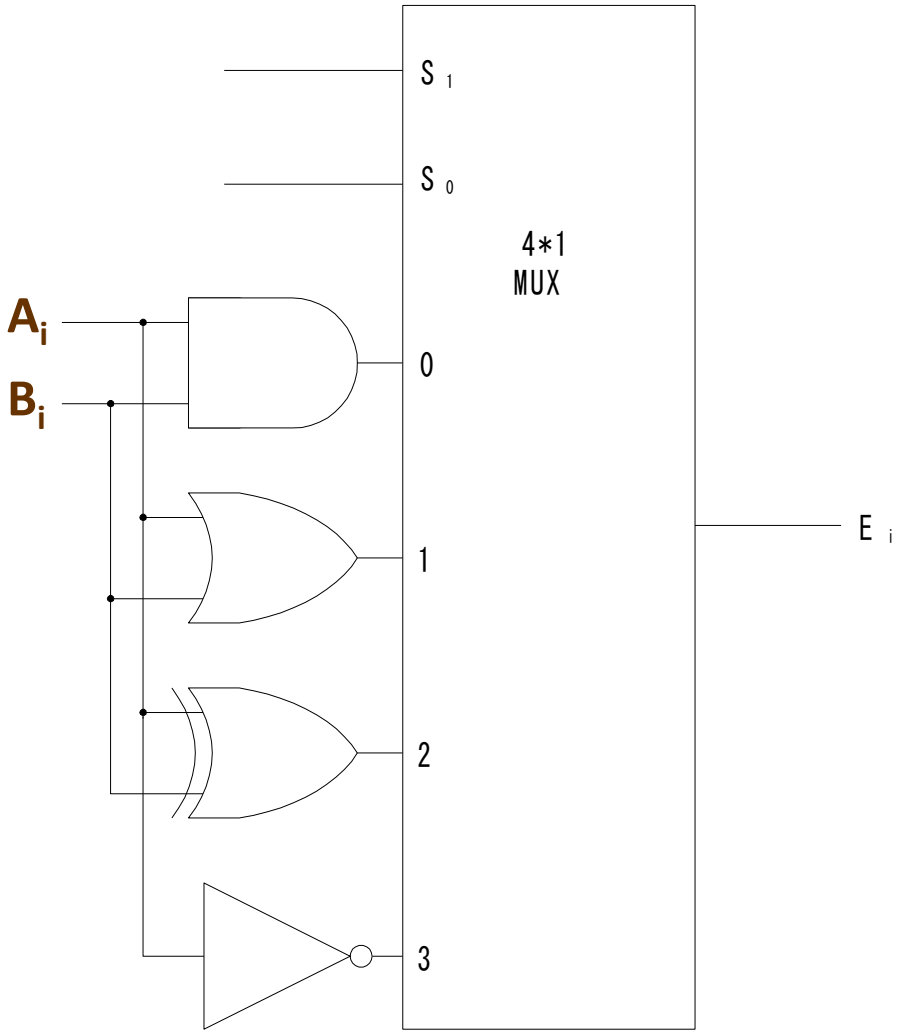
Special symbols will be adopted for the logic microoperations OR(V), AND(Λ), and complement (a bar on top), to distinguish them from the corresponding symbols used to express Boolean functions.

Logic Microoperations:

$$P+Q:R1 \leftarrow R2+R3,\ R4 \leftarrow R5 \vee R6$$

Logic OR

Arithmetic ADD



S_1	S_0	Output	Operation
0	0	E = A ∧ B	AND
0	1	E = A V B	OR
1	0	E = A ⊕ B	XOR
1	1	E = A’	Compliment NOT

Table #

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
$F_{10} = y'$	$F \leftarrow \overline{B}$	Compliment B
$F_{11} = x + y'$	$F \leftarrow A \vee \overline{B}$	
$F_{12} = x'$	$F \leftarrow \overline{A}$	Compliment 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

Logic Microoperations --- Applications:

❖ Logic microoperations are very useful for manipulating individual bits or a portion of a word stored in a register

❖ Used to change bit values, delete a group of bits, or insert new bit values

❖ Selective-set

❖ The selective-set operation sets to 1 the bits in register A where there are corresponding 1's in register B. It does not effect bit positions that have 0's in B.

$$A \leftarrow AVB$$

1010 A before
1100 B(Logic Operand)
1110 A After

❖ Selective-complement

❖The selective-complement operation complements bits in A where there are corresponding 1's in B. It does not effect bit positions that have 0's in B.

$$A \leftarrow A \oplus B$$

1010 A before
1100 B(Logic Operand)
0110 A After

Logic Microoperations --- Applications:

- **Selective-clear**
  - The selective-clear operation clears to 0 the bits in A only where there are corresponding 1's in B.

$$A \leftarrow A \wedge \overline{B}$$

```
1010  A before
1100  B(Logic Operand)
0010  A After
```

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

$$A \leftarrow A \wedge B$$

```
1010  A before
1100  B(Logic Operand)
1000  A After masking
```

- **Clear**
  - The clear operation compares the words in A and B and produces an all 0's result if the two numbers are equal

$$A \leftarrow A \oplus B$$

```
0110   A
0110   B
0000   A after clear
```

Shift Microoperations:

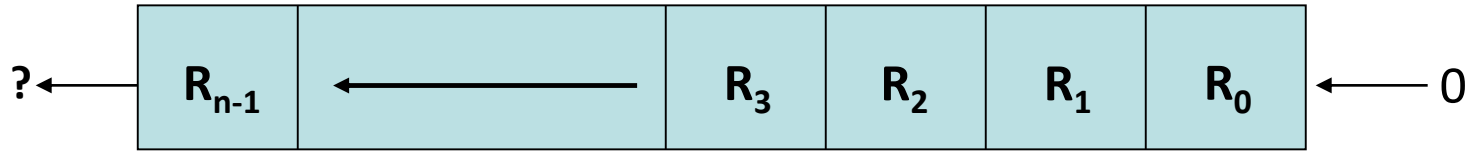
- Used for serial transfer of data.
- Three types of shift: Logical, Circular, and Arithmetic.

Symbolic Designation	Description
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 register R
R ← ashr R	Arithmetic shift right register R

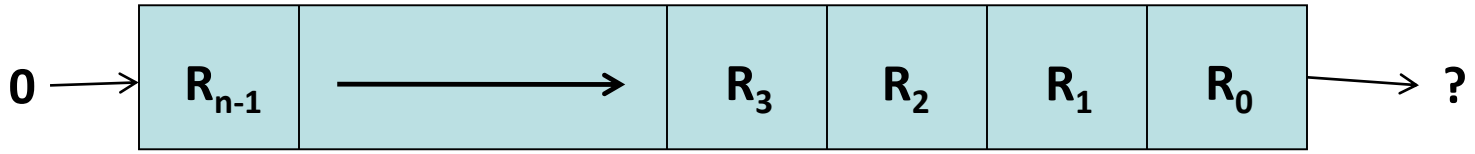
Logical Shift:

- ❖ Transfers 0 through the serial input.

Logical Shift Left:



Logical Shift Right:



Register Contents Before Shift Operation							
1	0	1	1	0	1	0	1

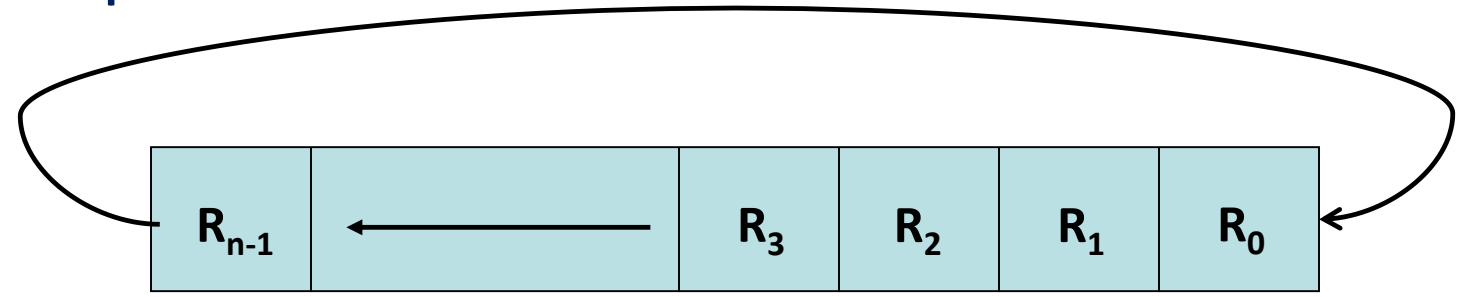
Register Contents After Shift left Operation							
0	1	1	0	1	0	1	0

Register Contents After Shift Right Operation							
0	1	0	1	1	0	1	0

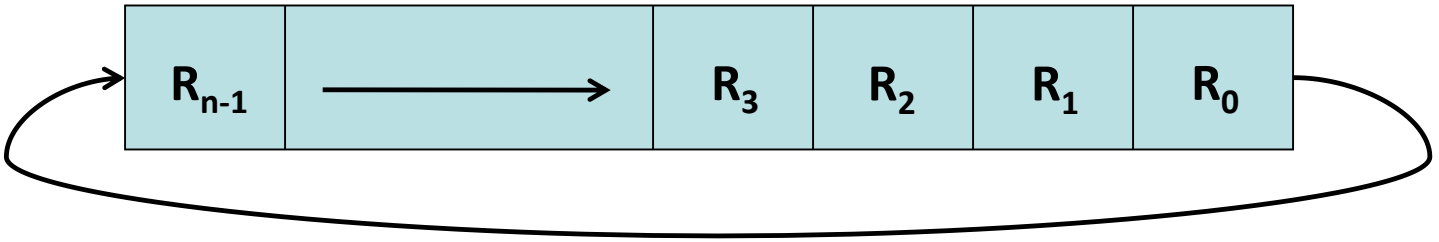
Circular Shift:

❖ Also known as rotate operation, circulates the bits of the register around the two ends without loss of information.

❖ This is accomplished by connecting the serial output of the shift register to its serial input.



Circular Shift Left



Circular Shift Right

Circular Shift Examples:

Register Contents Before Shift Operation							
1	0	1	1	0	1	0	1

Register Contents After Shift left Operation							
0	1	1	0	1	0	1	1

Register Contents After Shift Right Operation							
1	1	0	1	1	0	1	0

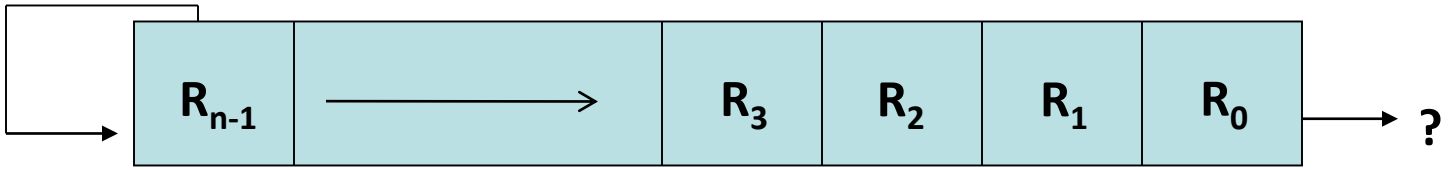
Arithmetic Shift:

- Shifts a signed binary number to the left or right.
- An arithmetic shift-left multiplies a signed binary number by 2:

  ashl (00100): 01000
- An arithmetic shift-right divides the number by 2:

  ashr (00100) : 00010

Arithmetic Shift Right:



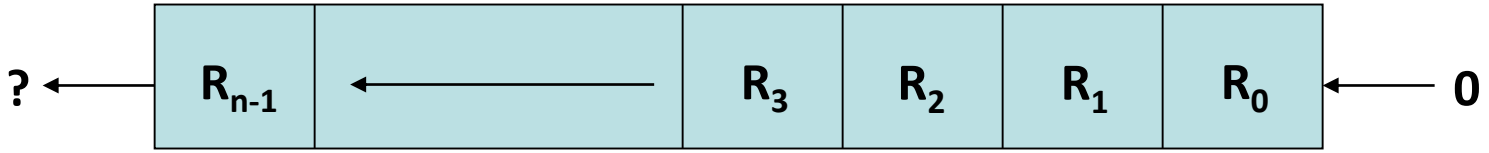
Sign Bit

Arithmetic Shift Right

Register Contents Before Arithmetic Shift Right Operation							
0	0	1	1	0	1	0	1
Register Contents After Arithmetic Shift Right Operation							
0	0	0	1	1	0	1	0
Register Contents Before Arithmetic Shift Right Operation							
1	0	1	1	0	1	0	1
Register Contents After Arithmetic Shift Right Operation							
1	1	0	1	1	0	1	0

Arithmetic Shift Left:

- The arithmetic shift-left inserts a 0 into R_0, and shifts all other bits to the left.
- The initial bit of R_{n-1} is lost and replaced by the bit from R_{n-2}.



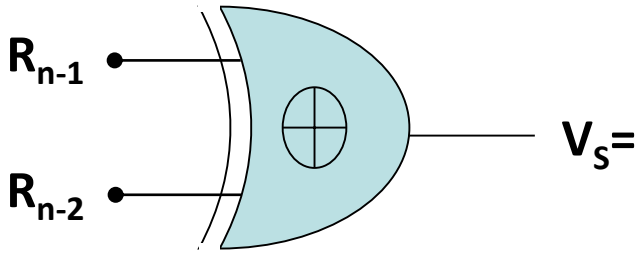
Sign Bit

Arithmetic Shift Left

Register Contents Before Arithmetic Shift Left Operation							
0	0	1	1	0	1	0	1
Register Contents After Arithmetic Shift Left Operation							
0	1	1	0	1	0	1	0
Register Contents Before Arithmetic Shift Left Operation							
1	0	1	1	0	1	0	1
Register Contents After Arithmetic Shift Left Operation							
0	1	1	0	1	0	1	0

Arithmetic Shift Left:

- A sign reversal occurs if the bit in R_{n-1} changes in value after the shift.
- This happens if the multiplication by 2 causes an overflow.
- An overflow flip-flop V_S can be used to detect an arithmetic shift-left overflow.
- $V_S = R_{n-1} \oplus R_{n-2}$



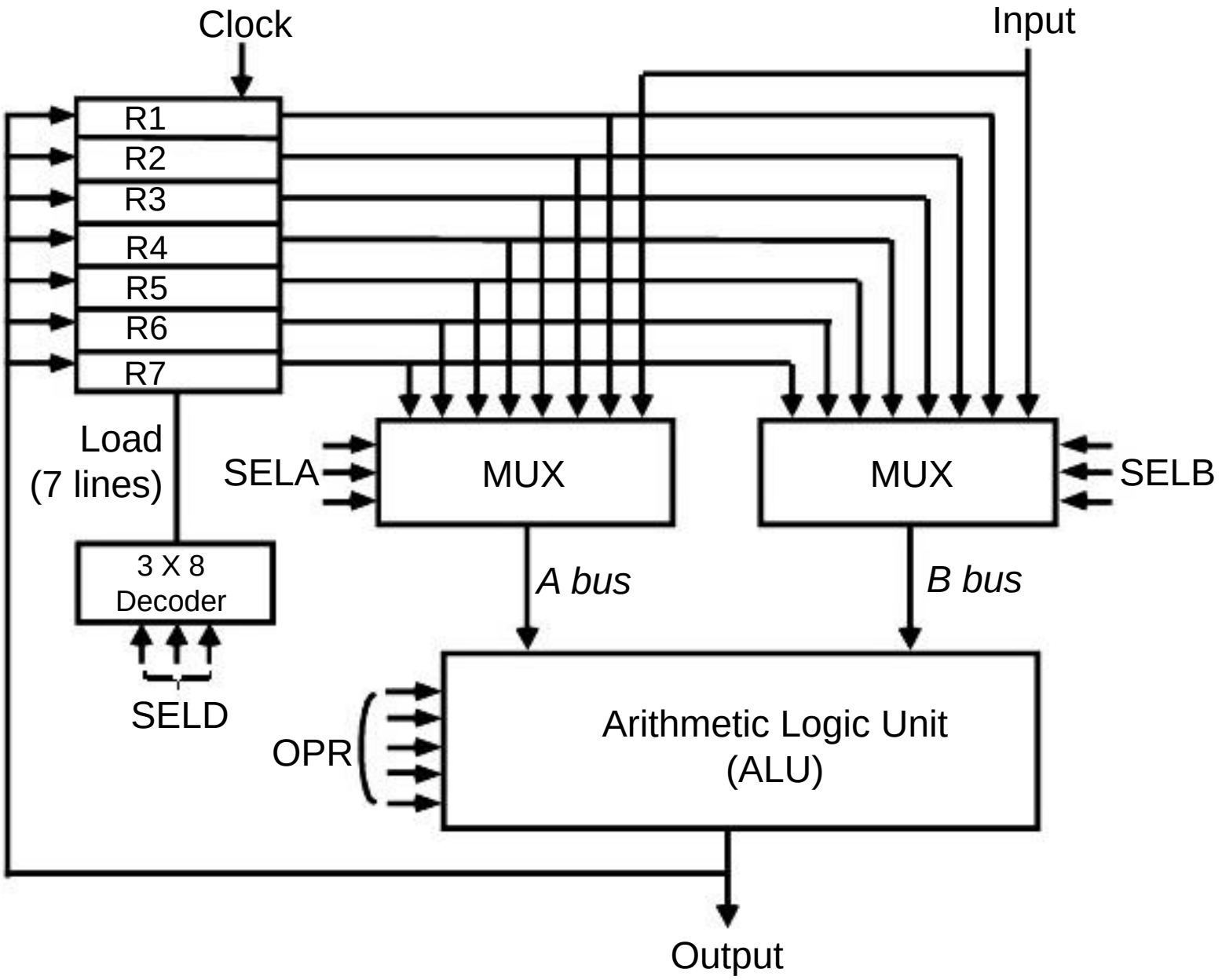
1 → overflow

0 → no overflow

Types of CPU Organizations:

- Single accumulator organization
- General register organization
- Stack organization

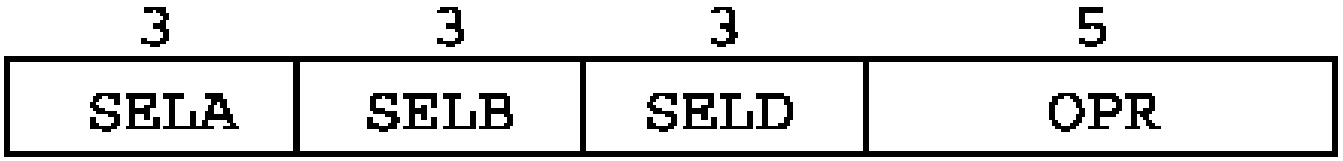
General Register Organization:



General Register Organization:

- 2 MUX: select one of 7 register or external data input by SELA and SELB.
- BUS A and BUS B form the inputs to a common ALU.
- ALU : An operation is selected by the ALU operation selector (OPR).
- The result of the microoperation is available for external data output and also goes into the inputs of all registers.
- The result of a microoperation is directed to a destination register selected by a decoder (SELD).
- Control word: The 14 binary selection inputs (3 bits for SELA, 3 for SELB, 3 for SELD, and 5 for OPR)

Control Word Format:



General Register Organization:

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

General Register Organization:

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	Compliment A	COMA
10000	Shift right A	SHRA
11000	Shift left A	SHLA

General Register Organization:

- Example: R1 ← R2 + R3
- MUX A selector (SELA - 010) : to place the content of R2 into BUS A.
- MUX B selector (SELB - 011) : to place the content of R3 into BUS B.
- ALU operation selector (OPR - 00010) : to provide the arithmetic addition R2 + R3.
- Decoder selector (SELD - 001) : to transfer the content of the output bus into R1.

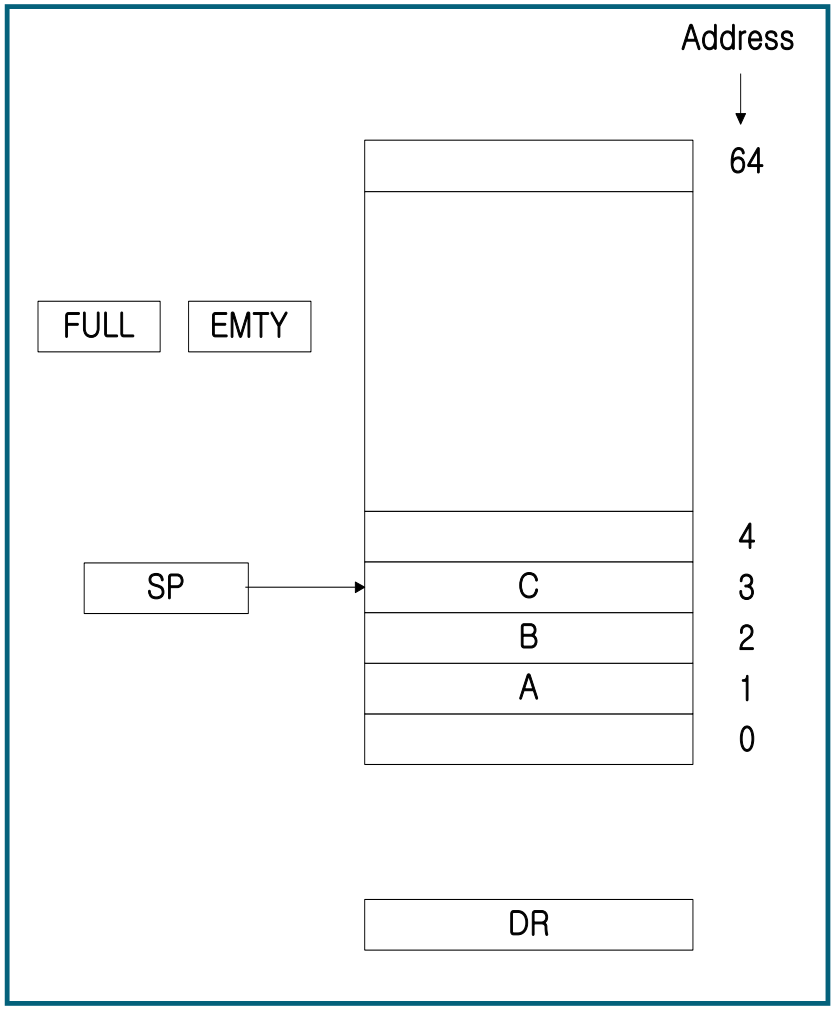
Field	SELA	SELB	SELD	OPR
Symbol	R2	R3	R1	ADD
Control Word:	010	011	001	00010

Control Word Format:

Stack Organization:

- Stack is a set of memory locations in R/W memory (or set of registers) used to store data temporarily during program execution.
- A stack stores the information in such a manner that the item stored last is the first item retrieved. i.e., LIFO; last-in, first-out.
- The address register that holds the address for the stack is called a stack pointer (SP) because its value always points at the top item in the stack.
- The two operations of the stack are the insertion and deletion of items.
- The operation of insertion is called *push* (or push-down).
- The operation deletion is called *pop* (or pop-up).
- A stack can be placed in a portion of s large memory (*memory stack*) or it can be organized as a collection of a finite number of memory words or registers (*register stack*).

Register Stack:



- A register stack is organized as a collection of a finite number of registers.
- In a 64-word stack, the stack pointer contains 6 bits.
- The one-bit register FULL is set to 1 when the stack is full; EMTY register is 1 when the stack is empty.
- The data register DR holds the data to be written into or read from the stack.
- Initially, SP is cleared to 0, EMTY is set to 1, and FULL is cleared to 0, so that SP points to the word at address 0.
- If the stack is not full (if FULL = 0), a new item is inserted with a *push* operation.
- A new item is deleted from the stack if the stack is not empty (if EMTY = 0) with a *pop* operation.

Register Stack:

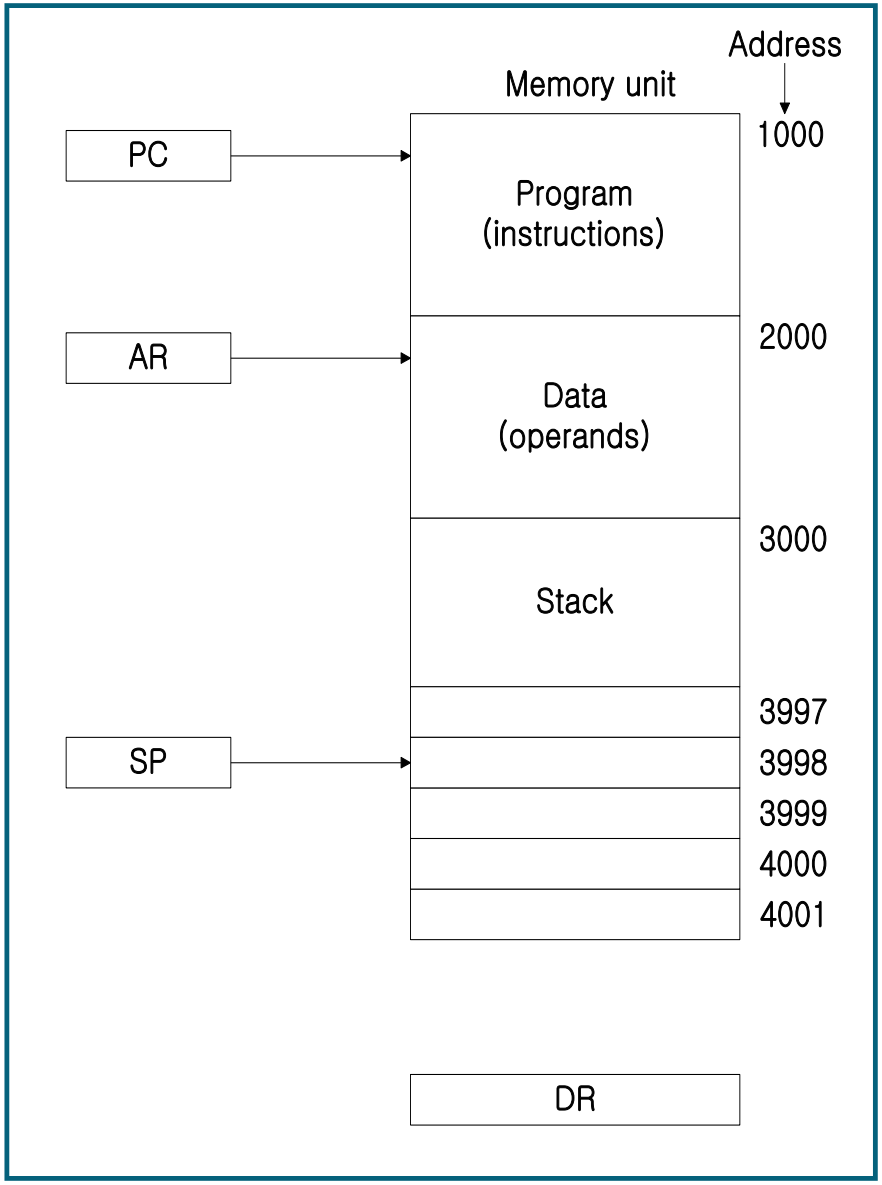
- ***push*** **operation:**

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

- ***pop*** **operation:**

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

Memory Stack:



- A memory stack is implemented by assigning a portion of memory to a stack operation and using a processor register as a stack pointer.
- The program counter PC points at the address of the next instruction in the program.
- The address register AR points at an array of data.
- The stack pointer SP points at the top of the stack.
- The three registers are connected to a common address bus, and either one can provide address for memory.
- Once the stack is initialized by loading address of stack top in SP, the stack grows with decreasing address.

Memory Stack:

- *push* operation:
- SP ← SP - 1 — Decrement stack pointer
- M[SP] ← DR — Write item on top of the stack
- *pop* operation:
- DR ← M[SP] — Read item from top of stack
- SP ← SP +1 — Increment stack pointer

Arithmetic Expressions - Notation:

- A + B — Infix notation
- +AB — Prefix or Polish notation
- AB+ — Postfix or reverse Polish notation (RPN)
- A * B + C * D → AB*CD*+
- This expression is evaluated from left to right as follows:
- ⇒ (A* B)CD*+
- ⇒ (A* B)(C*D)+
- ⇒ (A* B) + (C*D)

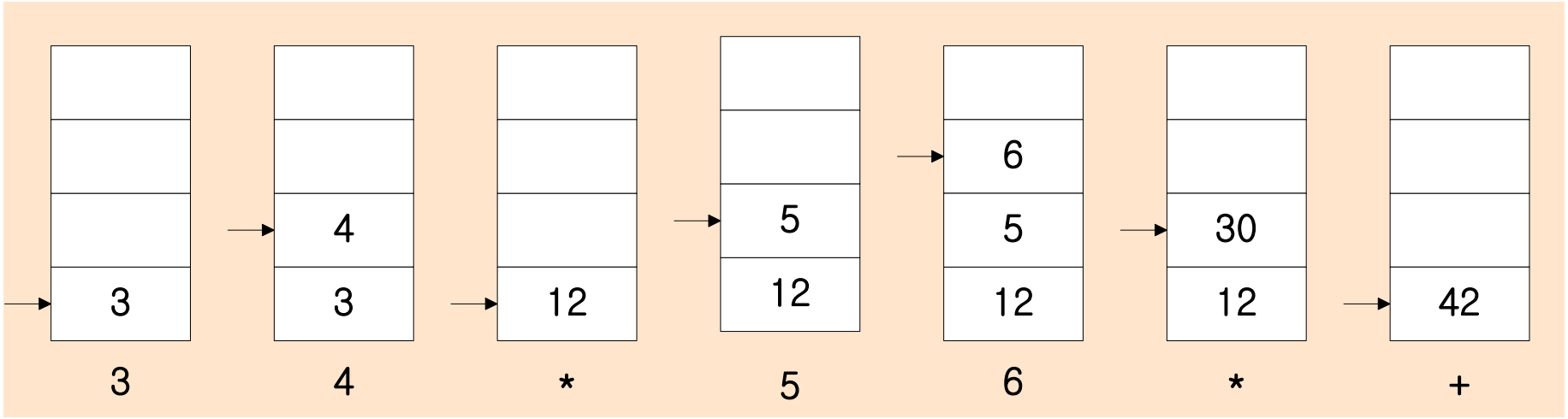
Conversion to RPN:

- **The conversion from infix to reverse Polish notation must follow the operational hierarchy adopted for infix notation.**
- **First perform all arithmetic inside inner parenthesis, then inside outer parenthesis, and do multiplication and division operations before addition and subtraction operations.**
- **Consider the expression (A + B) * [C * (D + E) + F]**
- **First we evaluate the arithmetic inside the inner parenthesis (A + B) and (D +E).**
- **Next calculate the expression inside the square brackets.**
- **The multiplication of C * (D + E) must be done prior to the addition of F since multiplication has precedence over addition.**
- **The last operation is the multiplication of the two terms between the parenthesis and brackets.**
- **The converted expression is AB + DE + C * F + ***

Evaluation of Arithmetic Expression:

- *Consider the example*

(3 * 4) + (5 * 6) → 34 * 56 * +



Instruction Formats:

- **The bits of an instruction are divided into three fields.**
  - ❖ **Operation Code Field : specify the operation to be performed.**
  - ❖ **Address Field : designate a memory address or a processor register.**
  - ❖ **Mode Field : specify the operand or the effective address (Addressing Mode).**

Instruction Codes:

- An instruction code is a group of bits that instruct that the computer to perform a specific operation.
- An instruction code is usually divided into two parts, operation code (opcode) and operand.
- The operation code is a group of bits that define such operations as add, subtract, multiply, shift, and compliment.
- An operation code is sometimes called a macrooperation because it specifies a set of micro-operations.

Instruction Formats:

- Three types of CPU organizations.
- The influence of the number of addresses on computer instructions
  - ❖ Consider the example X = (A + B)*(C + D)
- Three Address Instruction:

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

- Two Address Instruction:

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

- **One 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

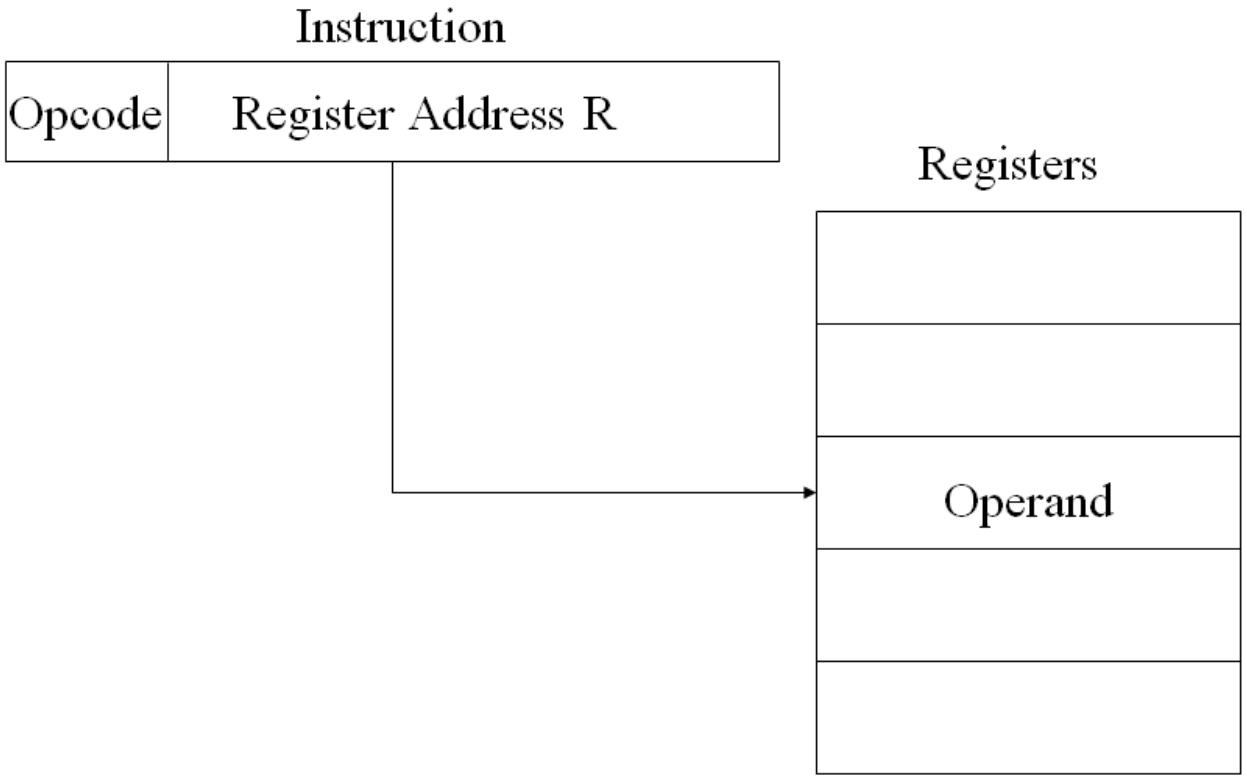
- **Zero Address Instructions:**

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[T] ← TOS

Addressing Modes:

- The way the operands are chose during program execution is dependent on the addressing mode of the instruction.
- The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually referenced.
- **Immediate Mode**: In this mode the operand is specified in the instruction itself i.e., an immediate-mode instruction has an operand field rather than an address field.
- Immediate mode instructions are useful for initializing registers to a constant value.

- Example: MOVE #100, R1 R1 ← 100
- **Register Mode:** In this mode the operands are in CPU registers.
- The particular register is selected from a register filed in the instruction.
- A k-bit field can specify any one of 2^k registers.

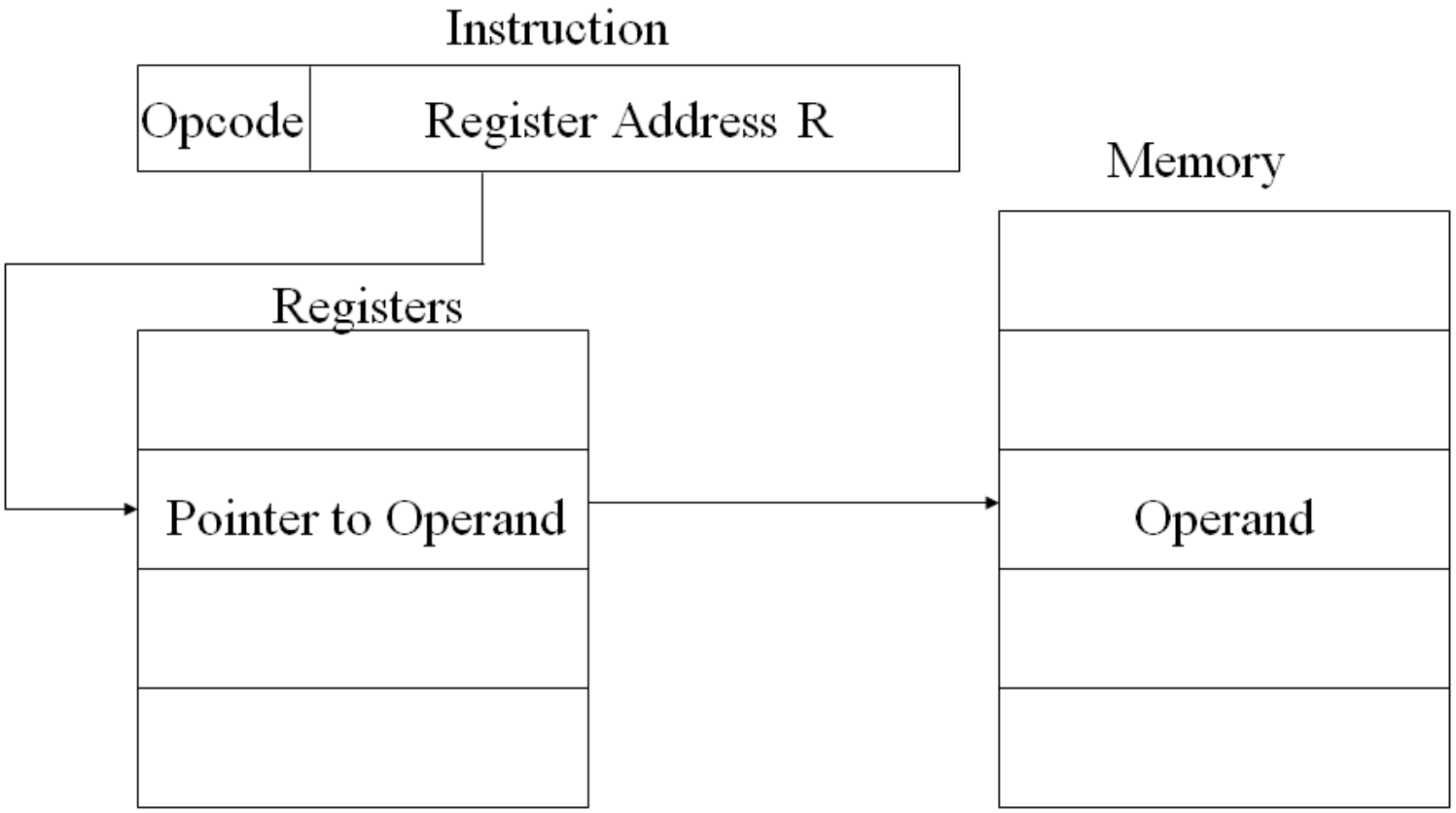


•**Register Indirect Mode:** **In this mode the instruction specifies a CPU register whose contents give the address of the operand in memory.**

• **The advantage of a register indirect mode instruction is that the address field of the instruction uses fewer bits to select a register than a memory address directly.**

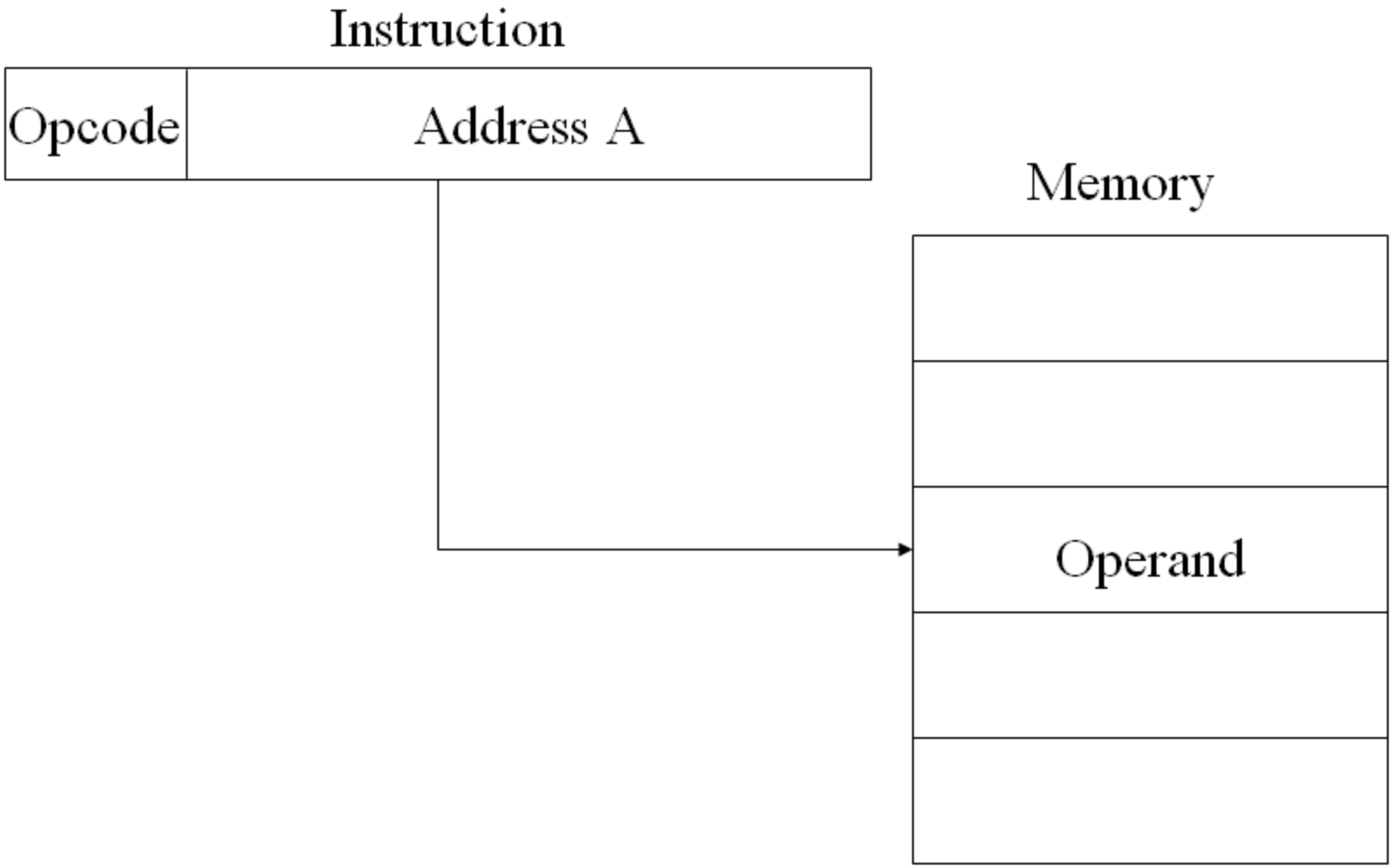
•**Autoincrement or Autodecrement Mode:** **This is similar to the register indirect mode except that the register is incremented or decremented after (or before) its value is used to access memory.**

• **This mode is used to refer to a table of data in memory.**

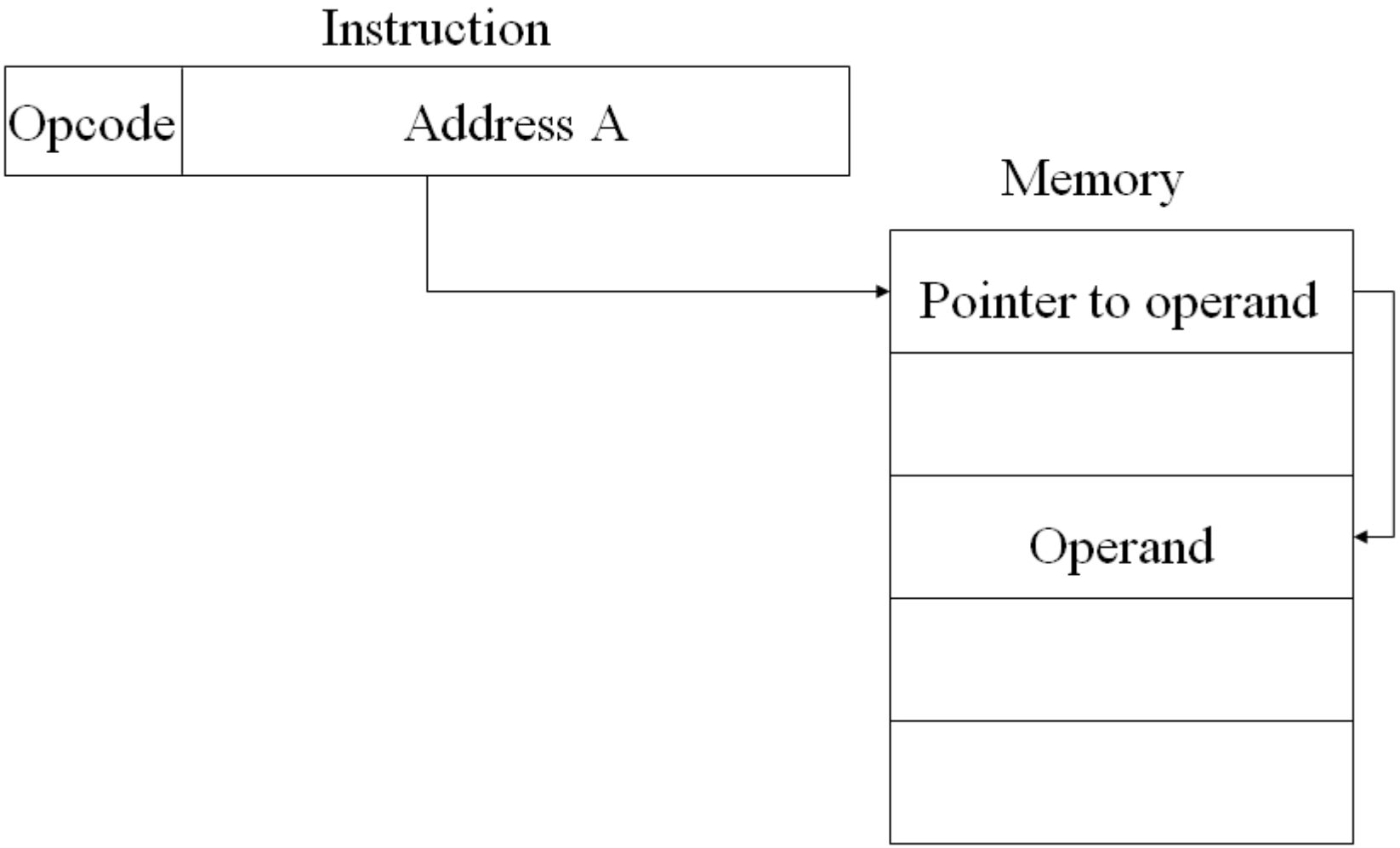


- **Direct Address Mode: In this mode the effective address is equal to the address part of the instruction.**
- **The operand resides in memory and its address is given directly by the address field of the instruction.**
- **Indirect Address Mode: In this mode the address field of the instruction gives the address where the effective address is stored in memory.**

Direct Address Mode:



Indirect Address Mode:



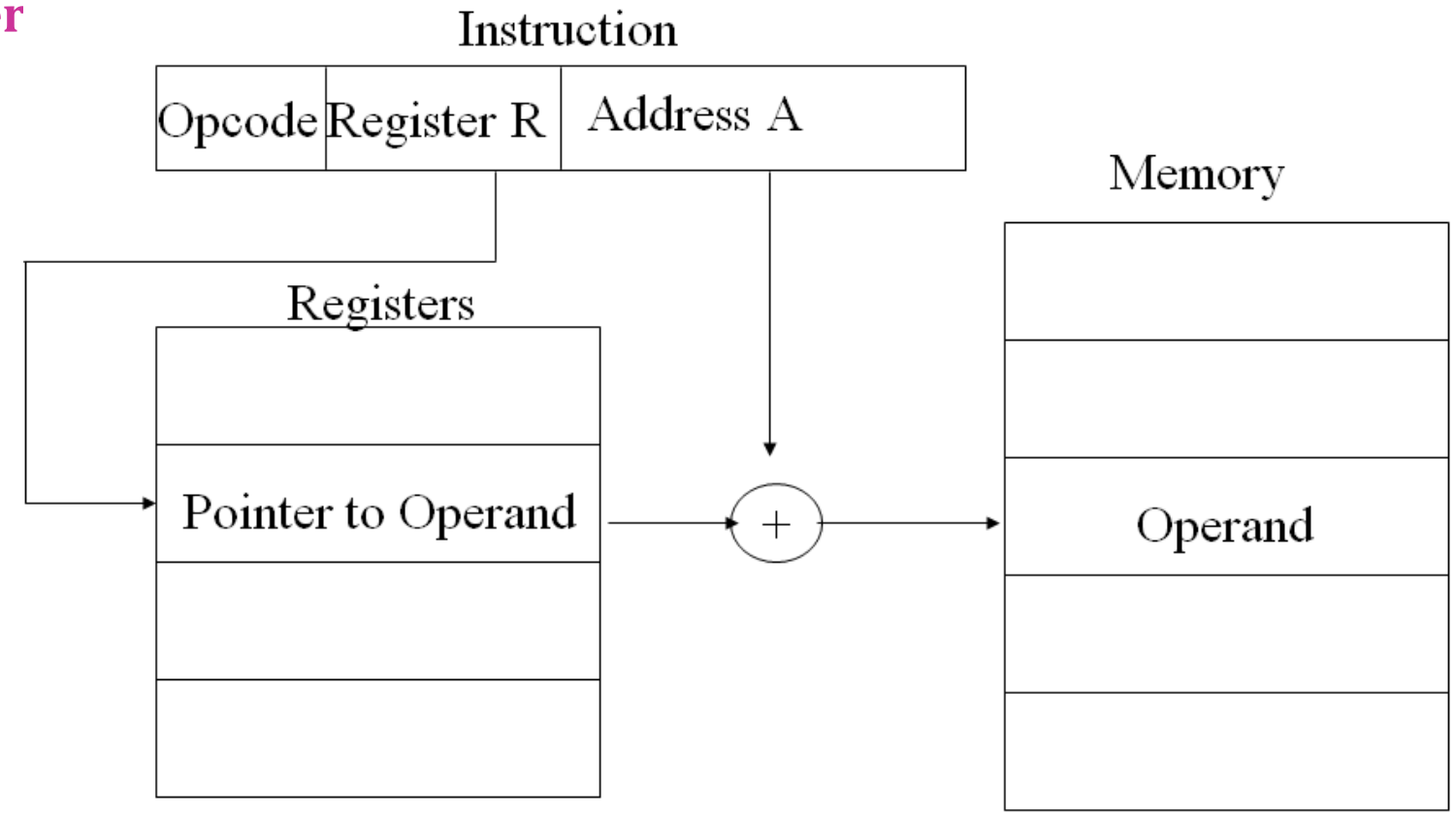
Addressing Modes:

•Indexed Addressing Mode: In this mode the content of an index register is added to the address part of the instruction to obtain the effective address.

• The index register is a special CPU register that contains an index value.

• The address field of the instruction defines the beginning address of a data array in memory.

effective address = address part of instruction + content of CPU register



Addressing Modes:

•Relative address Mode: In this mode the content of the program counter is added to the address part of the instruction in order to obtain the effective address.

•Base Register Addressing Mode: In this mode the content of a base register is added to the address part of the instruction to obtain the effective address.

• The base register holds a base address and the address field of the instruction gives a displacement relative to the base address.

•Implied Mode: In this mode the operands are specified implicitly in the definition of the instruction.

• Examples: Compliment Accumulator

• All register-reference instructions that use an accumulator are implied-mode instructions.

• Zero address instructions in a stack-organized computer are implied-mode instructions since the operands are implied to be on top of the stack.

Numerical Example

To show the differences between the various modes, we will show the effect of the addressing modes on the instruction defined in fig:1. The two-word instruction at address 200 and 201 is a "load to AC" instruction with an address field equal to 500. The first word of the instruction specifies the operation code and mode, and the second word specifies the address part. PC has the value 200 for fetching this instruction. The content of the processor register R1 is 400,and the content of an index register XR is 100. AC receives the operand after the instruction is executed.

Address Memory

200	Load to Ac Mode
201	Address=500
202	Next instruction
399	450
400	700
500	800
600	900
702	325
800	300

al example for
es

ressing mode	Effective Address	Content of AC
ct Address	500	800
ediate and	201	500
ect Address	800	300
tive Address	702	325
xed Address	600	900
ster		400
ster Indirect	400	700
Increment	400	700
decrement	399	450

Table :Tabular List of Numerical Example

2. Write a program to evaluate the arithmetic statement:

$$X = \frac{A - B + C * (D * E - F)}{G + H * K}$$

a. Using a general register computer with three address instructions.

b. Using a general register computer with two address instructions.

c. Using an accumulator type computer with one address instructions.

d. Using a stack organized computer with zero-address operation instructions.

a) Three address instructions:

SUB R1, A, B	R1←M [A] - M [B]
MUL R2, D, E	R2←M [D] * M [E]
SUB R2, R2, F	R2← R2 – M [F]
MUL R2, R2, C	R2← R2*M [C]
ADD R1, R1, R2	R1← R1+R2
MUL R3, H, K	R3← M [H] + M [K]
ADD R3, R3, G	R3← R3+ M [G]
DIV X, R1, R3	X← R1/R3

b) Two address instructions:

MOV R1, A	R1←M [A]
SUB R1, B	R1←R1-M [B]
MOV R2, D	R2←M [D]
MUL R2, E	R2←R2*M [E]
SUB R2, F	R2←R2 – M [F]
MUL R2, C	R2←R2*M[C]
ADD R1, R2	R1←R1+R2
MOV R3, H	R3←M [H]
ADD R3, G	R3←R3+M [G]
DIV R1, R3	R1←R1/R3
MOV X, R1	M[X]←R1

c) One Address instructions:

LOAD A	AC← M [A]
SUB B	AC←AC-M [B]
STORE T	M [T]←AC
LOAD D	AC←M [D]
MUL E	AC←AC*M [E]
SUB F	AC←AC-M [F]
MUL C	AC←AC*M[C]
ADD T	AC←AC+M [T]
STORE T	M [T]←AC
LOAD H	AC←M [H]
MUL K	AC←AC*M [K]
ADD G	AC←AC+M [G]
STORE T1	M [T1]←AC
LOAD T	AC←M [T]
DIV T1	AC←AC/M [T1]
STORE X	M[X]←AC

d) Zero address instructions:

RPN: AB-CDE*F-*+GHK*+/

PUSH A	TOS←A
PUSH B	TOS←B
SUB	TOS←(A-B)
PUSH C	TOS←C
PUSH D	TOS←D
PUSH E	TOS←E
MUL	TOS← (D*E)
PUSH F	TOS← F
SUB	TOS←((D*E)-F)
MUL	TOS←C*((D*E)-F)
ADD	TOS←((A-B)+ C*((D*E)-F)
PUSH G	TOS← G
PUSH H	TOS←H
PUSH K	TOS←K
MUL	TOS←(H*K)
ADD	TOS←G+(H*K)
DIV	TOS←((A-B)+ C*((D*E)-F)/(G+(H*K))
POP X	M[X]←TOS

. The memory unit of a computer has 256K words of 32 bits each. The computer has an instruction format with four fields: an operation code field, a mode field to specify one of seven addressing modes, a register address field to specify one of 60 processor registers, and a memory address. Specify the instruction format and the number of bits in each field if the in instruction is in one memory word.

$256\ K = 2^8 \times 2^{10} = 2^{18}$

op code	Mode	Register	Address

5 3 6 18 = 32

Address = 18 bits

Mode = 3 bits

Register = 6 bits

27 bits

op code 5 bits

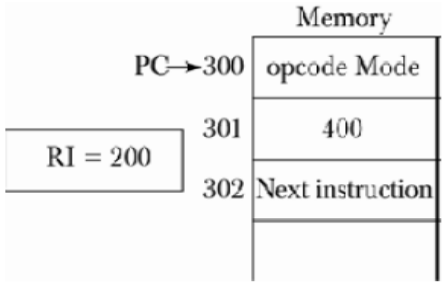
32 bits

8. An instruction is stored at location 300 with its address field at location 301. The address field has the value 400. A processor register R1 contains the number 200. Evaluate the effective address if the addressing mode of the instruction is (a) direct; (b) immediate; (c) relative; (d) register indirect; (e) index with R1 as the index register.

(a)direct addressing:

Direct addressing means that the address field contains the address of memory location the instruction is supposed to work with (where an operand "resides").

Effective address would therefore be 400.



(b) immediate addressing

Immediate addressing means that the address field contains the operand itself.

Effective address would therefore be 301.

(c) relative addressing

Relative addressing means that the address field contains offset to be added to the program counter to address a memory location of the operand.

Effective address would therefore be 302 + 400 = 702.

(d) register indirect addressing

Register indirect addressing means that the address of an operand is in the register. The address field in this case contains just another operand.

Effective address would therefore be in R1 = 200.

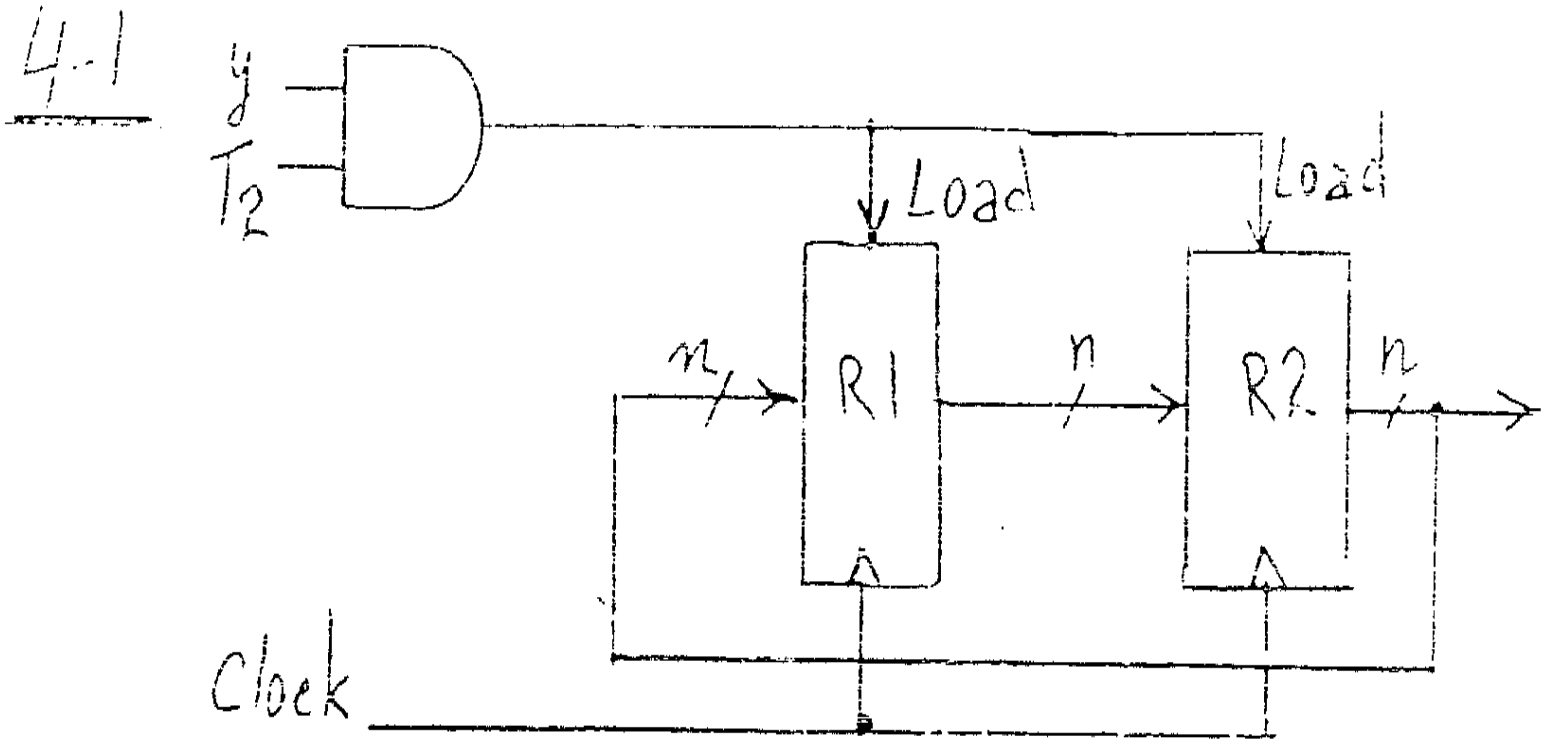
(e) indexed addressing with R1 as index register

In indexed absolute addressing the effective address is calculated by taking the contents of the address field and adding the contents of the index register.

Effective address would therefore be 400 + R1 = 400 + 200 = 600.

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

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



4-6. A digital computer has a common bus system for 16 registers of 32 bits each. The bus is constructed with multiplexers.

a. How many selection inputs are there in each multiplexer?

b. What size of multiplexers are needed?

c. How many multiplexers are there in the bus?

(a) 4 selection lines to select one of 16 registers,

(b) 16 x 1 multiplexers

(c) 32 multiplexers, one for each bit of the registers.

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

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

4-3 ...

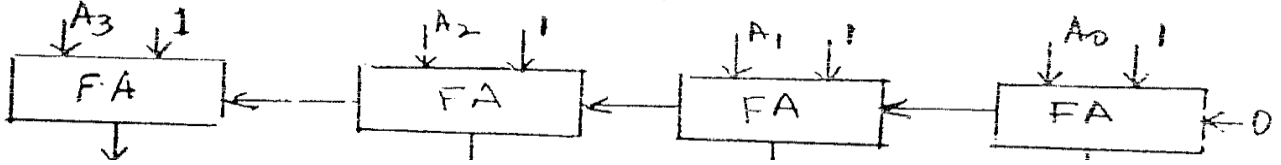
P: $R1 \leftarrow R2$

P'Q: $R1 \leftarrow R3$

4-13. Design a 4-bit combinational circuit decrementer using four full-adder circuits.

4-13

A − 1 = A + 2's complement of 1 = A + 1111



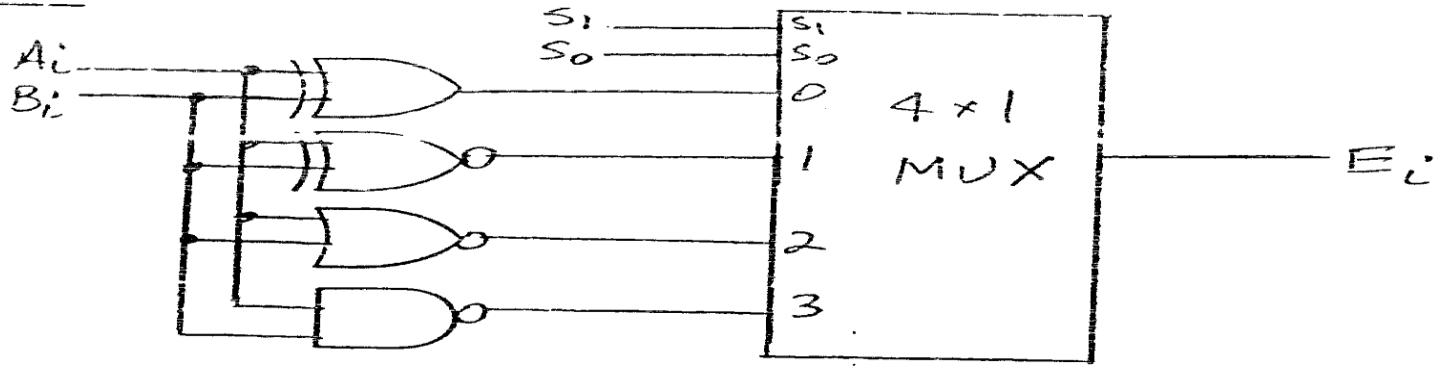
4-17. Design a digital circuit that performs the four logic operations of exclusive-OR, exclusive-NOR, NOR, and NAND. Use two selection variables. Show the logic diagram of one typical stage.

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

4-17



4-18

(a) A = 11011001

B = 10110100 ⊕

A ← A⊕B 01101101

A = 11011001

B = 11111101 (OR)

A ← A∨B 11111101

4-19. The 8-bit registers *AR*, *BR*, *CR*, and *DR* initially have the following values:

$$AR = 11110010$$
$$BR = 11111111$$
$$CR = 10111001$$
$$DR = 11101010$$

Determine the 8-bit values in each register after the execution of the following sequence of microoperations.

$AR \leftarrow AR + BR$	Add *BR* to *AR*
$CR \leftarrow CR \wedge DR,\ BR \leftarrow BR + 1$	AND *DR* to *CR*, increment *BR*
$AR \leftarrow AR - CR$	Subtract *CR* from *AR*

4-19

(a) AR = 11110010
BR = 11111111 (+)
AR = 11110001 BR = 11111111 CR = 10111001 DR = 11101010

(b) CR = 10111001 BR = 11111111
DR = 11101010 (AND) + 1
CR = 10101000 BR = 00000000 AR = 11110001 DR = 11101010

(c) AR = 11110001
CR = 10101000 (−)
AR = 01001001 ; BR = 00000000 ; CR = 10101000 ; DR = 11101010

4-20. An 8-bit register contains the binary value 10011100. What is the register value after an arithmetic shift right? Starting from the initial number 10011100, determine the register value after an arithmetic shift left, and state whether there is an overflow.

4-21. Starting from an initial value of $R = 11011101$, determine the sequence of binary values in R after a logical shift-left, followed by a circular shift-right, followed by a logical shift-right and a circular shift-left.

4-20

R = 10011100

Arithmetic shift right : 11001110

Arithmetic shift left : 00111000 overflow because a negative number changed to positive

4-21

R = 11011101

logical shift left : 10111010

Circular shift right : 01011101

logical shift right : 00101110

Circular shift left : 01011100

5-1. A computer uses a memory unit with 256K words of 32 bits each. A binary instruction code is stored in one word of memory. The instruction has four parts: an indirect bit, an operation code, a register code part to specify one of 64 registers, and an address part.

a. How many bits are there in the operation code, the register code part, and the address part?

b. Draw the instruction word format and indicate the number of bits in each part.

c. How many bits are there in the data and address inputs of the memory?

5-1

$256K = 2^8 \times 2^{10} = 2^{18}$

$64 = 2^6$

(a) Address: 18 bits

Register code: 6 bits

Indirect bit: 1 bit

25

$32 - 25 = 7$ bits for opcode.

(b)

1	7	6	18
I	opcode	Register	Address

$= 32$ bits

(c) Data: 32 bits; address: 18 bits.

8-1. A bus-organized CPU similar to Fig. 8-2 has 16 registers with 32 bits in each, an ALU, and a destination decoder.

a. How many multiplexers are there in the *A* bus, and what is the size of each multiplexer?
b. How many selection inputs are needed for MUX A and MUX B?
c. How many inputs and outputs are there in the decoder?
d. How many inputs and outputs are there in the ALU for data, including input and output carries?
e. Formulate a control word for the system assuming that the ALU has 35 operations.

8-1

(a) 32 multiplexers, each of size 16 × 1.

(b) 4 inputs each, to select one of 16 registers.

(c) 4-to-16-line decoder

(d) 32 + 32 + 1 = 65 data input lines

32 + 1 = 33 data output lines.

(e)

4	4	4	6
SELA	SELB	SELD	OPR

= 18 bits

8-5. Let $SP = 000000$ in the stack of Fig. 8-3. How many items are there in the stack if:

a. FULL = 1 and EMTY = 0?

b. FULL = 0 and EMTY = 1?

8-6. A stack is organized such that *SP* always points at the next empty location on the stack. This means that *SP* can be initialized to 4000 in Fig. 8-4 and the first item in the stack is stored in location 4000. List the microoperations for the push and pop operations.

8-5

(a) Stack full with 64 items.

(b) Stack empty.

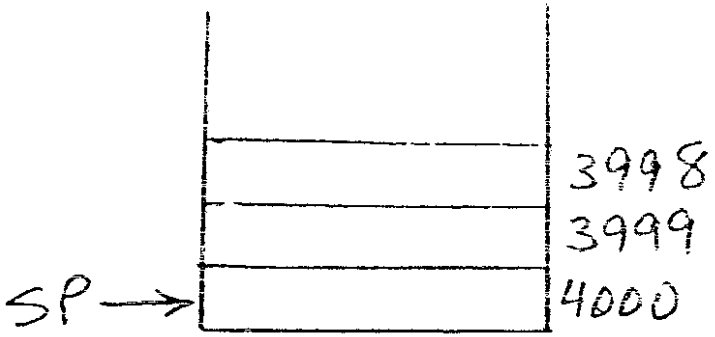
8-6

PUSH : $M[SP] \leftarrow DR$

$SP \leftarrow SP-1$

POP : $SP \leftarrow SP+1$

$DR \leftarrow M[SP]$



8-7. Convert the following arithmetic expressions from infix to reverse Polish notation.

a. $A * B + C * D + E * F$

b. $A * B + A * (B * D + C * E)$

c. $A + B * [C * D + E * (F + G)]$

d. $\dfrac{A * [B + C * (D + E)]}{F * (G + H)}$

8-8. Convert the following arithmetic expressions from reverse Polish notation to infix notation.

a. $A\ B\ C\ D\ E + * - /$

b. $A\ B\ C\ D\ E * / - +$

c. $A\ B\ C * /\ D - E\ F\ /\ +$

d. $A\ B\ C\ D\ E\ F\ G + * + * + *$

8-9. Convert the following numerical arithmetic expression into reverse Polish notation and show the stack operations for evaluating the numerical result.

$$(3 + 4)[10(2 + 6) + 8]$$

8-7

(a) $AB * CD * EF * + +$

(b) $AB * ABD * CE * + * +$

(c) $FG + E * CD * + B * A +$

(d) $ABCDE + * + * FGH + * /$

8-8

(a) $\dfrac{A}{B - (D+E) * C}$ (b) $A + B - \dfrac{C}{D * E}$

(c) $\dfrac{A}{B * C} - D + \dfrac{E}{F}$ (d) $(((F+G) * E + D) * C + B) * A$

8-9

$(3+4)[10(2+6)+8] = 616$

RPN: $3\ 4\ +\ 2\ 6\ +\ 10\ *\ 8\ +\ *$

				6		10		8		
	4		2	2	8	8	80	80	88	
3	3	7	7	7	7	7	7	7	7	616
3	4	+	2	6	+	10	*	8	+	*

8-13. The memory unit of a computer has 256K words of 32 bits each. The computer has an instruction format with four fields: an operation code field, a mode field to specify one of seven addressing modes, a register address field to specify one of 60 processor registers, and a memory address. Specify the instruction format and the number of bits in each field if the in instruction is in one memory word.

8-14. A two-word instruction is stored in memory at an address designated by the symbol W. The address field of the instruction (stored at W + 1) is designated by the symbol Y. The operand used during the execution of the instruction is stored at an address symbolized by Z. An index register contains the value X. State how Z is calculated from the other addresses if the addressing mode of the instruction is

a. direct
b. indirect
c. relative
d. indexed

8-13

$256K = 2^8 \times 2^{10} = 2^{18}$

address = 18 bits
Mode = 3 "
Register = 6 "
27 bits
op code 5
32 bits

opcode	Mode	Register	Address
5	3	6	18 = 32

8-14

Z = Effective address

(a) Direct : Z = Y
(b) Indirect : Z = M[Y]
(c) Relative : Z = Y + W + 2
(d) Indexed : Z = Y + X