JGI JAIN DEEMED-TO-BE UNIVERSITY | SCHOOL OF COMPUTER SCIENCE AND IT

Master of Computer Applications

23MCAC105 – Advanced Computer Architecture

Credits: 3

L: T: P:E – 3-0-0-3

Prepared by

Dr. A. Rengarajan, Professor

Course Outcome (CO)

CO1	Identify the different functional units and instruction format for building basic structure
CO2	Demonstrate the various arithmetic operations using data representation and number systems.
CO3	Classify data transfer types and memory types in transferring data among components.
CO4	Evaluate the problems in Pipelining and Data Hazards to avoid scheduling issues
CO5	Appraise multiprocessor architecture using parallel and centralized shared memory architecture.

Text Books:

1 M.Moris Mano, (2007), "Computer Systems Architecture", Pearson/PHI, IIIrd Edition (Module 1,2 & 3)

2 Richard Y. Kain, (2011), "Advanced Computer Architecture a Systems Design Approach", PHI (Module 4 &5)

Module – 1

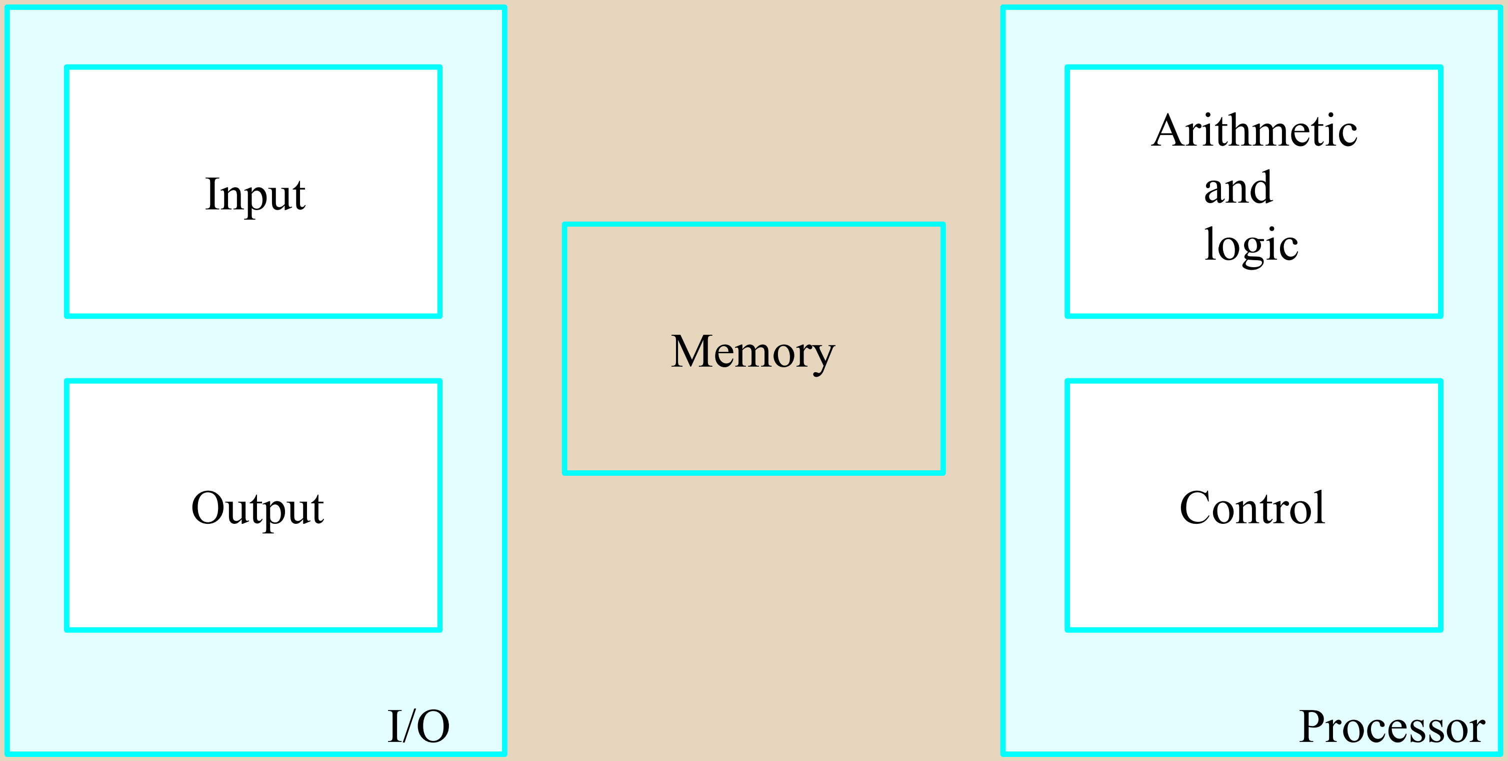
BASIC STRUCTURE OF COMPUTER

- Functional Units of a Digital Computer
- Hardware
- Software Interface
- Translation from a High Level Language to the Hardware Language
- Instruction Set Architecture
- Performance Metrics
- Amdahl’s Law
- General Register Organization
- Stack Organization
- Instruction Format
- Types of Instruction
- Addressing modes

Basic Structure of Computer

- ✓ **Computer Architecture** is a set of rules and methods that describe the functionality, organization, and implementation of computer systems.
- ✓ Some definitions of architecture define it as describing the capabilities and programming model of a computer but not a particular implementation.
- ✓ In other definitions computer architecture involves instruction set architecture design, micro architecture design, logic design, and implementation.

Functional Units of a Digital Computer 1/7



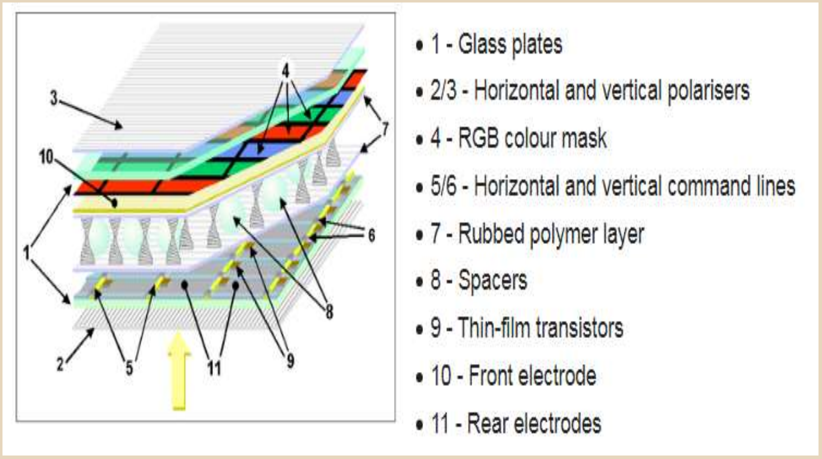
Functional Units of a Digital Computer 2/7

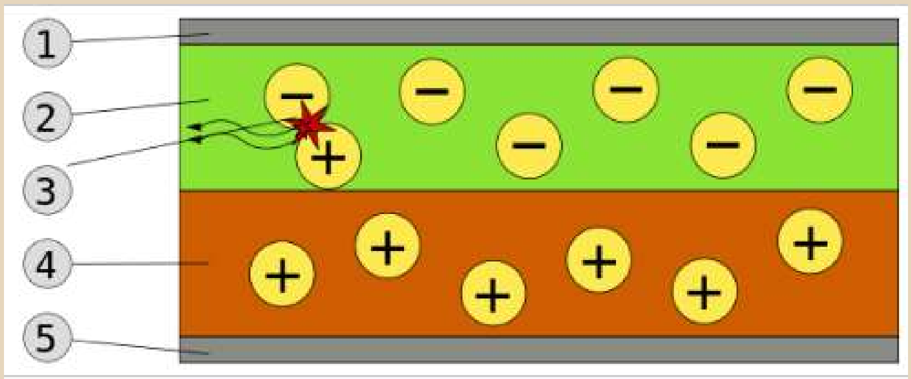
- ✓ **Input Unit:** Computer accepts encoded information through input unit. The standard input device is a keyboard. Whenever a key is pressed, keyboard controller sends the code to CPU/Memory.

  Examples include Mouse, Joystick, Tracker ball, Light pen, Digitizer, Scanner etc.

Functional Units of a Digital Computer^{3/7}

- ✓ **Output Unit:** Computer after computation returns the computed results, error messages, etc. via output unit. The standard output device is a video monitor, LCD/TFT monitor. Other output devices are printers, plotters etc.





Schematic of a bilayer OLED: 1. Cathode (−), 2. Emissive Layer, 3. Emission of radiation, 4. Conductive layer, 5. Anode (+)

JAIN DEEMED-TO-BE UNIVERSITY | SCHOOL OF COMPUTER SCIENCE AND IT

Functional Units of a Digital Computer 4/7

- ✓ **Arithmetic and logic unit:** ALU consist of necessary logic circuits like adder, comparator etc., to perform operations of addition, multiplication, comparison of two numbers etc.

- ✓ **Control Unit:** Control unit co-ordinates activities of all units by issuing control signals. Control signals issued by control unit govern the data transfers and then appropriate operations take place. Control unit interprets or decides the operation/action to be performed.

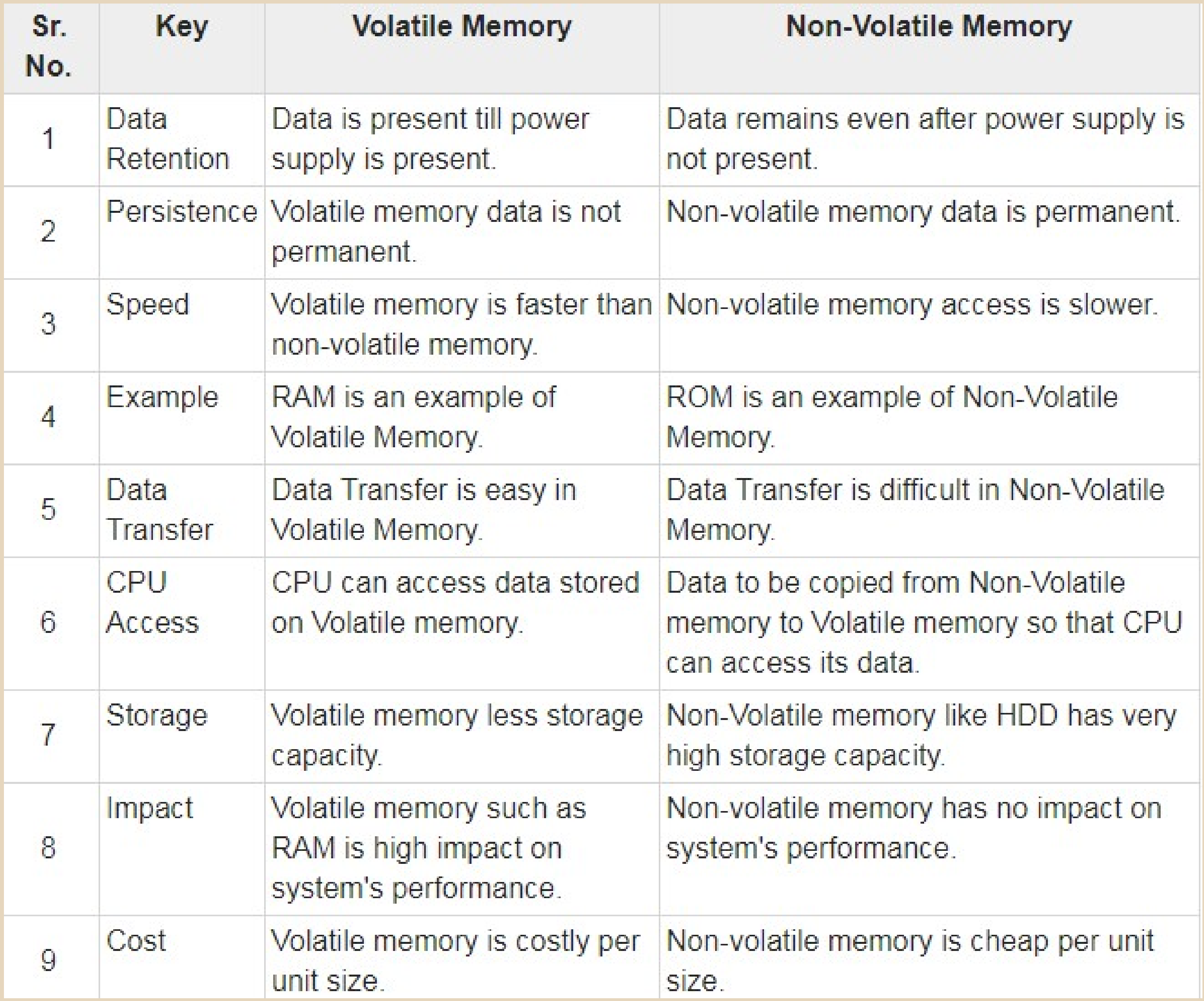
Functional Units of a Digital Computer 5/7

- ✓ **Memory Unit:** Memory unit stores the program instructions (Code), data and results of computations etc. Memory unit is classified as:
  - ❖ Primary /Main Memory
  - ❖ Secondary /Auxiliary Memory

Functional Units of a Digital Computer 6/7

- ✓ **Primary memory** is a semiconductor memory that provides access at high speed. Run time program instructions and operands are stored in the main memory.
- ✓ Main memory is classified again as ROM and RAM. ROM holds system programs and firmware routines such as BIOS, POST, I/O Drivers that are essential to manage the hardware of a computer.
- ✓ RAM is termed as Read/Write memory or user memory that holds run time program instruction and data. While primary storage is essential, it is volatile in nature and expensive.
- ✓ Additional requirement of memory could be supplied as auxiliary memory at cheaper cost.
- ✓ **Secondary memories** are non volatile in nature.

Volatile Vs. Non-Volatile



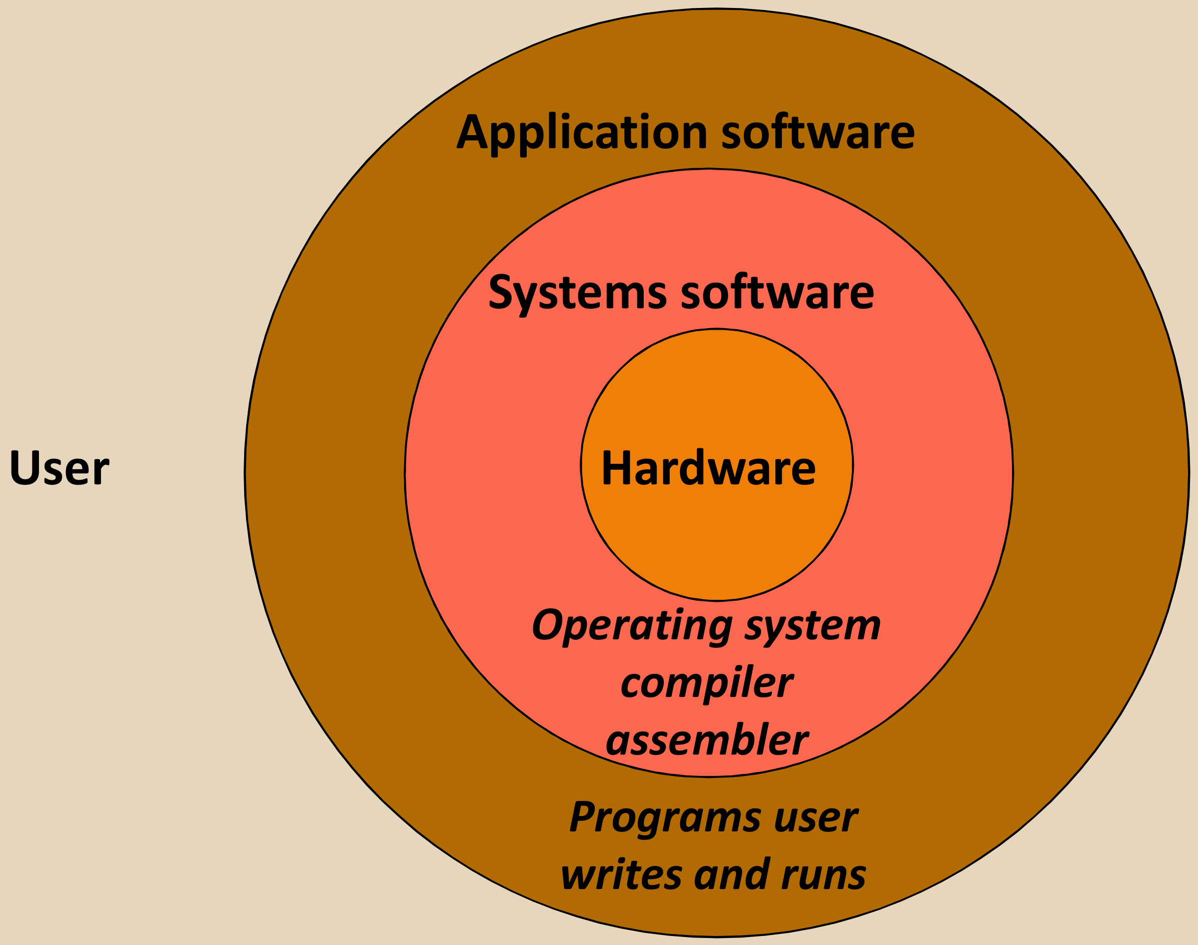
Sr. No.	Key	Volatile Memory	Non-Volatile Memory
1	Data Retention	Data is present till power supply is present.	Data remains even after power supply is not present.
2	Persistence	Volatile memory data is not permanent.	Non-volatile memory data is permanent.
3	Speed	Volatile memory is faster than non-volatile memory.	Non-volatile memory access is slower.
4	Example	RAM is an example of Volatile Memory.	ROM is an example of Non-Volatile Memory.
5	Data Transfer	Data Transfer is easy in Volatile Memory.	Data Transfer is difficult in Non-Volatile Memory.
6	CPU Access	CPU can access data stored on Volatile memory.	Data to be copied from Non-Volatile memory to Volatile memory so that CPU can access its data.
7	Storage	Volatile memory less storage capacity.	Non-Volatile memory like HDD has very high storage capacity.
8	Impact	Volatile memory such as RAM is high impact on system's performance.	Non-volatile memory has no impact on system's performance.
9	Cost	Volatile memory is costly per unit size.	Non-volatile memory is cheap per unit size.

Functional Units of a Digital Computer 7/7

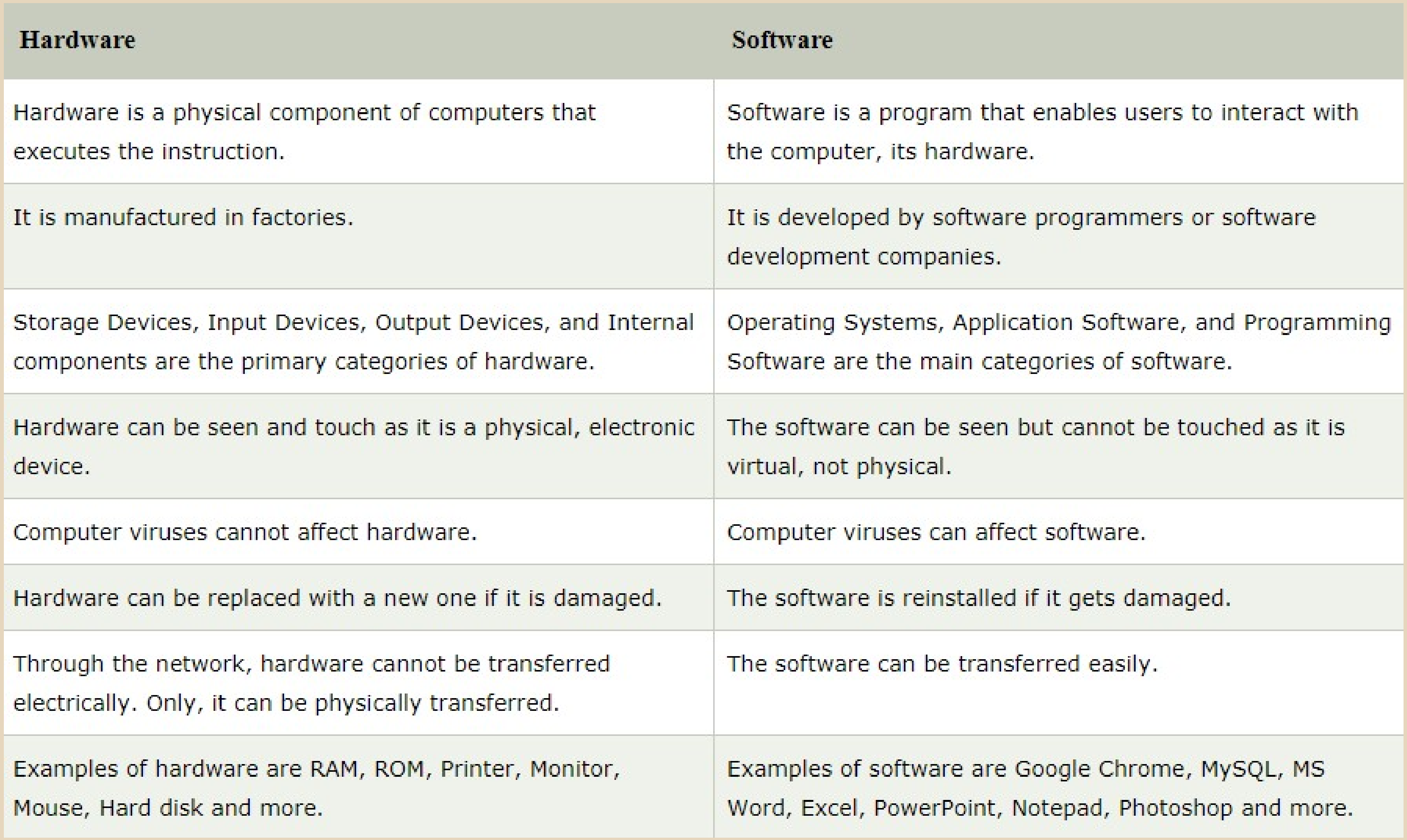
✓ The operations of a computer can be summarized as follows:

1. A set of instructions called a program reside in the main memory of computer.
2. The CPU fetches those instructions sequentially one-by-one from the main memory, decodes them and performs the specified operation on associated data operands in ALU.
3. Processed data and results will be displayed on an output unit.
4. All activities pertaining to processing and data movement inside the computer machine are governed by control unit.

Hardware – Software Interface



Hardware Vs Software



Hardware	Software
Hardware is a physical component of computers that executes the instruction.	Software is a program that enables users to interact with the computer, its hardware.
It is manufactured in factories.	It is developed by software programmers or software development companies.
Storage Devices, Input Devices, Output Devices, and Internal components are the primary categories of hardware.	Operating Systems, Application Software, and Programming Software are the main categories of software.
Hardware can be seen and touch as it is a physical, electronic device.	The software can be seen but cannot be touched as it is virtual, not physical.
Computer viruses cannot affect hardware.	Computer viruses can affect software.
Hardware can be replaced with a new one if it is damaged.	The software is reinstalled if it gets damaged.
Through the network, hardware cannot be transferred electrically. Only, it can be physically transferred.	The software can be transferred easily.
Examples of hardware are RAM, ROM, Printer, Monitor, Mouse, Hard disk and more.	Examples of software are Google Chrome, MySQL, MS Word, Excel, PowerPoint, Notepad, Photoshop and more.

Translation from a High Level Language to the Hardware Language

Compiler → **Assembler** →

Application software, a program in C:

```
swap (int  v[ ], int  k)
{int  temp;
            temp = v[k];
            v[k] = v[k+1];
            v[k+1] = temp;
}
```

MIPS compiler output, assembly language program:

```
swap;
            muli      $2,       $5, 4
            add       $2,       $4, $2
            lw        $15,       0 ($2)
            lw        $16,       4 ($2)
            sw        $16,       0 ($2)
            sw        $15,       4 ($2)
            jr        $31
```

MIPS binary machine code:

```
00000000101000010000000000011000
00000000000110000001100000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

Instruction Set Architecture (ISA)

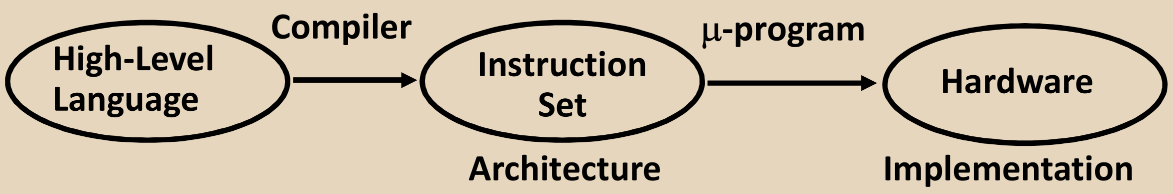
- ✓A set of assembly language instructions (ISA) provides a link between software and hardware.
- ✓Given an instruction set, software programmers and hardware engineers work more or less independently.
- ✓ISA is designed to extract the most performance out of the available hardware technology.
- ✓ Defines data transfer modes between registers, memory and I/O
- ✓ Types of ISA: RISC, CISC, VLIW, Superscalar
- ✓ Examples:
  - IBM370/X86/Pentium/K6 (CISC), PowerPC (Superscalar)
  - Alpha (Superscalar), MIPS (RISC and Superscalar)
  - Sparc (RISC), UltraSparc (Superscalar)

RISC: REDUCED INSTRUCTION SET COMPUTERS 1/5

Historical Background

IBM System/360, 1964

- **The real beginning of modern computer architecture**
- **Distinction between *Architecture* and *Implementation***
- **Architecture: The abstract structure of a computer seen by an assembly-language programmer**



Continuing growth in semiconductor memory and microprogramming
-> A much richer and complicated instruction sets
=> CISC(Complex Instruction Set Computer)

- Arguments advanced at that time
Richer instruction sets would simplify compilers
Richer instruction sets would alleviate the software crisis
- move as much functions to the hardware as possible
- close *Semantic Gap* between machine language and the high-level language
Richer instruction sets would improve the *architecture quality*

COMPLEX INSTRUCTION SET COMPUTERS: CISC

High Performance General Purpose Instructions

Characteristics of CISC:

1. A large number of instructions (from 100-250 usually)
2. Some instructions that performs a certain tasks are not used frequently.
3. Many addressing modes are used (5 to 20)
4. Variable length instruction format.
5. Instructions that manipulate operands in memory.

PHYLOSOPHY OF RISC

Reduce the semantic gap between machine instruction and microinstruction

1-Cycle instruction

Most of the instructions complete their execution in 1 CPU clock cycle - like a microoperation

*** Functions of the instruction (contrast to CISC)**
- Very simple functions
- Very simple instruction format
- Similar to microinstructions
=> No need for microprogrammed control

*** Register-Register Instructions**
- Avoid memory reference instructions except Load and Store instructions
- Most of the operands can be found in the registers instead of main memory
=> Shorter instructions
=> Uniform instruction cycle
=> Requirement of large number of registers

*** Employ instruction pipeline**

CHARACTERISTICS OF RISC

Common RISC Characteristics

- Operations are register-to-register, with only LOAD and STORE accessing memory

- The operations and addressing modes are reduced

Instruction formats are simple

CHARACTERISTICS OF RISC

5/5

RISC Characteristics

- Relatively few instructions
- Relatively few addressing modes
- Memory access limited to load and store instructions
- All operations done within the registers of the CPU
- Fixed-length, easily decoded instruction format
- Single-cycle instruction format
- Hardwired rather than microprogrammed control

More RISC Characteristics

-A relatively large numbers of registers in the processor unit.
-Efficient instruction pipeline
-Compiler support: provides efficient translation of high-level language programs into machine language programs.

Advantages of RISC

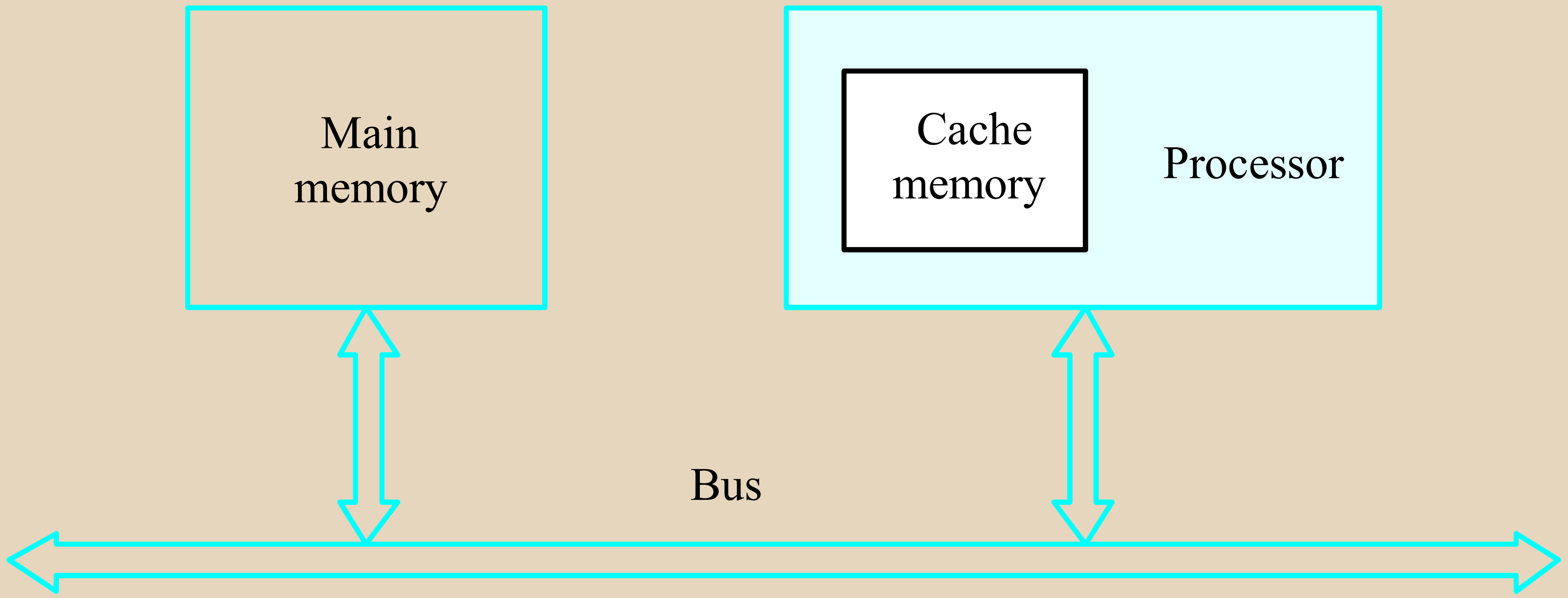
- VLSI Realization
- Computing Speed
- Design Costs and Reliability
- High Level Language Support

Performance

- ✓ The most important measure of a computer is how quickly it can execute programs.
- ✓ Three factors affect performance:
  - ❑ Hardware design
  - ❑ Instruction set
  - ❑ Compiler

Performance

✓Processor time to execute a program depends on the hardware involved in the execution of individual machine instructions.



The processor cache.

Performance

- The processor and a relatively small cache memory can be fabricated on a single integrated circuit chip.
- Speed
- Cost
- Memory management

Processor Clock

- ✓ Clock, clock cycle, and clock rate
- ✓ The execution of each instruction is divided into several steps, each of which completes in one clock cycle.
- ✓ Hertz – cycles per second

Basic Performance Equation

- T – processor time required to execute a program that has been prepared in high-level language
- N – number of actual machine language instructions needed to complete the execution (note: loop)
- S – average number of basic steps needed to execute one machine instruction. Each step completes in one clock cycle
- R – clock rate
- Note: these are not independent to each other

$$T = \frac{N \times S}{R}$$

How to improve T?

Pipeline and Superscalar Operation

- Instructions are not necessarily executed one after another.
- The value of S doesn't have to be the number of clock cycles to execute one instruction.
- Pipelining – overlapping the execution of successive instructions.
- Add R1, R2, R3
- Superscalar operation – multiple instruction pipelines are implemented in the processor.
- Goal – reduce S (could become <1!)

Clock Rate

- Increase clock rate
  - Improve the integrated-circuit (IC) technology to make the circuits faster
  - Reduce the amount of processing done in one basic step (however, this may increase the number of basic steps needed)
- Increases in R that are entirely caused by improvements in IC technology affect all aspects of the processor's operation equally except the time to access the main memory.

Compiler

- A compiler translates a high-level language program into a sequence of machine instructions.
- To reduce N, we need a suitable machine instruction set and a compiler that makes good use of it.
- Goal – reduce N×S
- A compiler may not be designed for a specific processor; however, a high-quality compiler is usually designed for, and with, a specific processor.

Performance Measurement

- T is difficult to compute.
- Measure computer performance using benchmark programs.
- System Performance Evaluation Corporation (SPEC) selects and publishes representative application programs for different application domains, together with test results for many commercially available computers.
- Compile and run (no simulation)
- Reference computer

$$SPEC\ rating = \frac{\text{Running time on the reference computer}}{\text{Running time on the computer under test}}$$

$$SPEC\ rating = \left(\prod_{i=1}^{n} SPEC_i\right)^{\frac{1}{n}}$$

Amdahl's Law

- ✓The execution time of the program after making the improvement is given by the following simple equation known as **Amdahl's Law:**
- ✓A rule stating that the performance enhancement possible with a given improvement is limited by the amount that the improved feature is used. It is a quantitative version of the law of diminishing returns.

$$\text{Execution time after improvement} = \frac{\text{Execution time affected by improvement}}{\text{Amount of improvement}} + \text{Execution time unaffected}$$

$$\text{Overall Speedup} = \frac{\text{Old execution time}}{\text{New execution time}} = \frac{1}{\left((1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}} \right)}$$

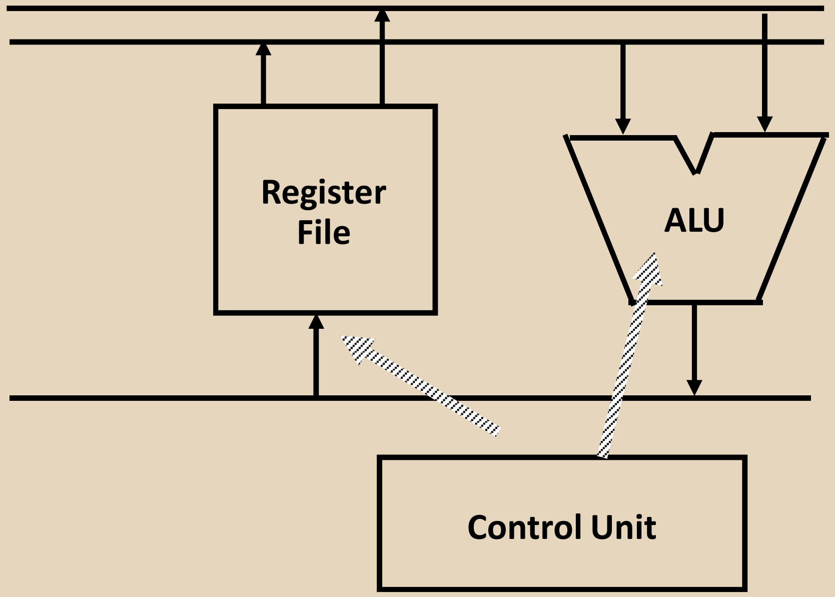
Major Components of CPU

Storage Components:
- Registers
- Flip-flops

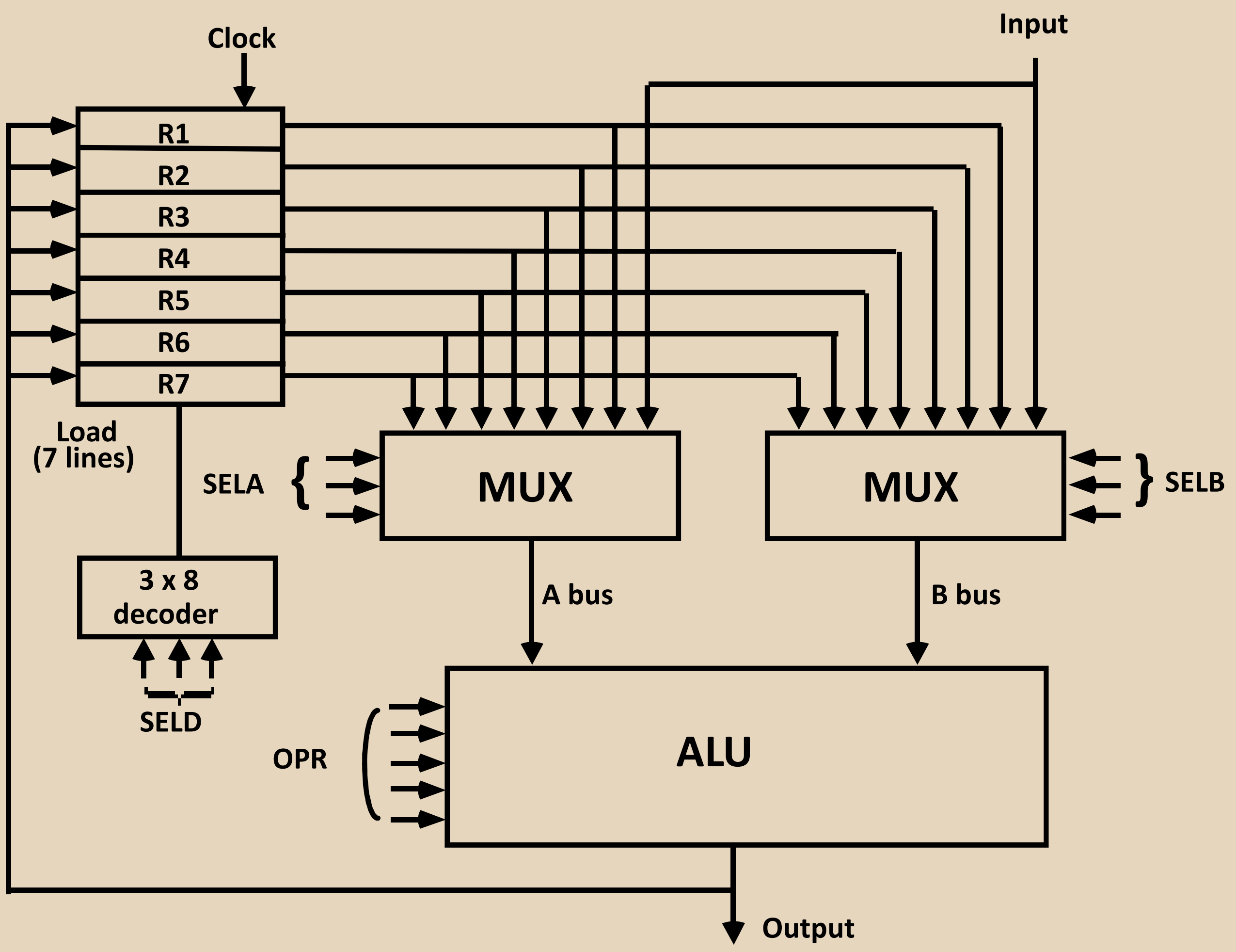
Execution (Processing) Components:
- Arithmetic Logic Unit (ALU):
- Arithmetic calculations, Logical computations, Shifts/Rotates

Transfer Components:
- Bus

Control Components:
- Control Unit



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 $\leftarrow$ R2

[2] MUX B selector (SELB): BUS B $\leftarrow$ R3

[3] ALU operation selector (OPR): ALU to ADD

[4] Decoder destination selector (SELD): R1 $\leftarrow$ Out Bus

Control Word

3	3	3	5
SELA	**SELB**	**SELD**	**OPR**

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

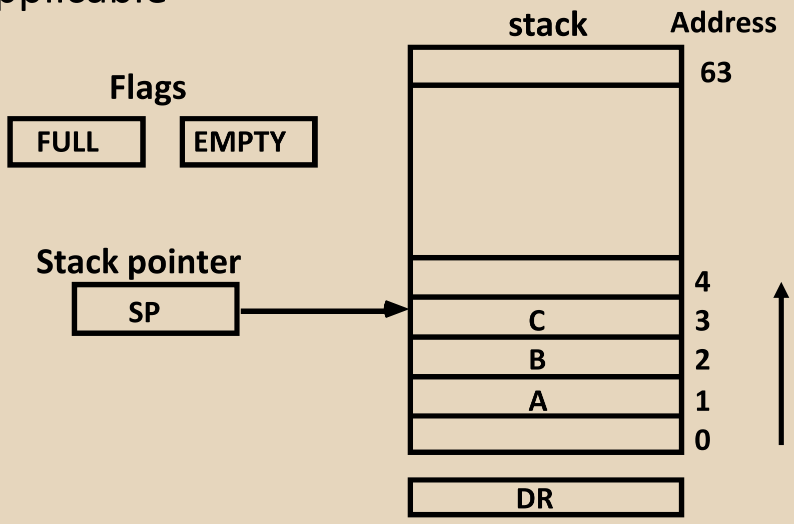
Examples of ALU Microoperations

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

Register Stack Organization

Stack
- Very useful feature for nested subroutines, nested loops control
- Also efficient for arithmetic expression evaluation
- Storage which can be accessed in LIFO
- Pointer: SP
- Only PUSH and POP operations are applicable

Register Stack



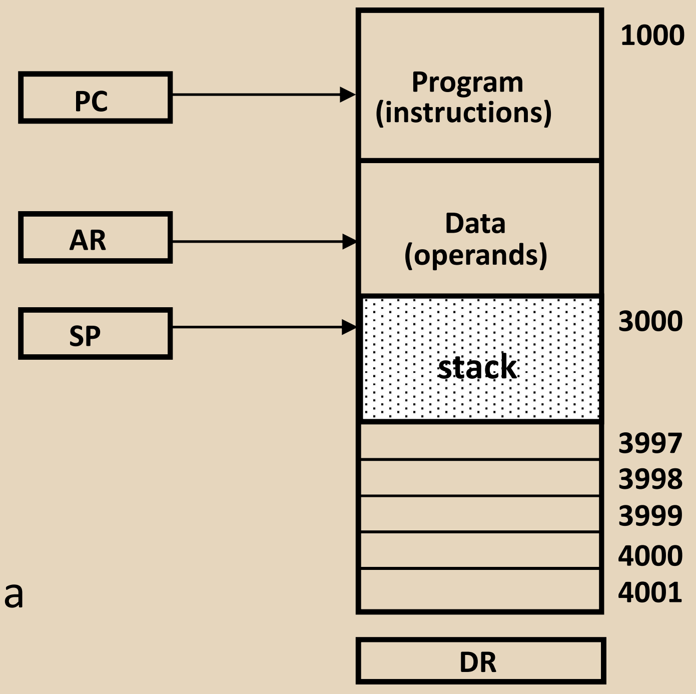
Push, Pop operations

```
/*  Initially, SP = 0, EMPTY = 1, FULL = 0  */

 PUSH                                    POP
SP ← SP + 1                   DR ← M[SP]
M[SP] ← DR                    SP ← SP - 1
If (SP = 0) then (FULL ← 1)             If (SP = 0) then (EMPTY ← 1)
EMPTY ← 0                     FULL ← 0
```

Memory Stack Organization

Memory with Program, Data, and Stack Segments



- A portion of memory is used as a stack with a processor register as a stack pointer

- PUSH: $SP \leftarrow SP - 1$
  $M[SP] \leftarrow DR$
- POP: $DR \leftarrow M[SP]$
  $SP \leftarrow SP + 1$

- Most computers do not provide hardware to check stack overflow (full stack) or underflow(empty stack)

Reverse Polish Notation

Arithmetic Expressions: A + B

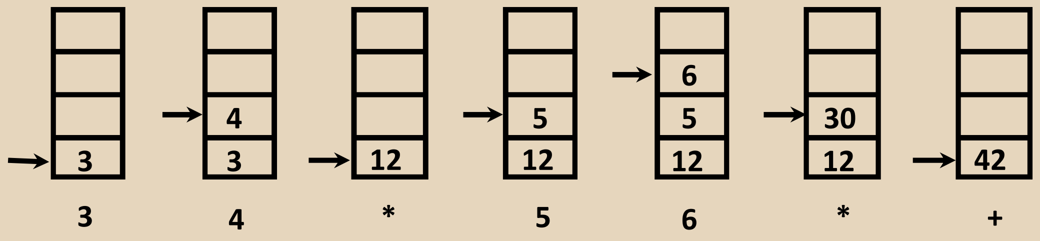
A + B	Infix notation
+ A B	Prefix or Polish notation
A B +	Postfix or reverse Polish notation

- The reverse Polish notation is very suitable for stack manipulation

Evaluation of Arithmetic Expressions

Any arithmetic expression can be expressed in parenthesis-free Polish notation, including reverse Polish notation

(3 * 4) + (5 * 6) ⇒ 3 4 * 5 6 * +



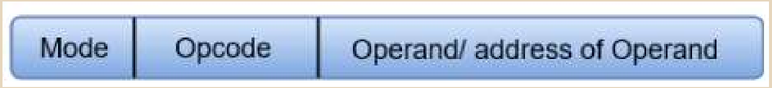
Instruction Format

Instruction Fields

OP-code field - specifies the operation to be performed

Address field - designates memory address(s) or a processor register(s)

Mode field - specifies the way the operand or the effective address is determined



Note: The number of address fields in the instruction format depends on the internal organization of CPU

- The three most common CPU organizations:

Single accumulator organization:

```
ADD     X              /* AC ← AC + M[X]  */
```

General register organization:

```
ADD     R1, R2, R3     /* R1 ← R2 + R3  */
ADD     R1, R2         /* R1 ← R1 + R2  */
MOV     R1, R2         /* R1 ← R2  */
ADD     R1, X          /* R1 ← R1 + M[X]  */
```

Stack organization:

```
PUSH    X              /* TOS ← M[X]  */
ADD
```

Types of Instructions Format

Three-Address Instructions:

Program to evaluate X = (A + B) * (C + D) :

```
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    */
```

- Results in short programs
- Instruction becomes long (many bits)

Two-Address Instructions:

Program to evaluate X = (A + B) * (C + D) :

```
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        */
```

Types of Instructions Format

One-Address Instructions:

- Use an implied AC register for all data manipulation
- Program to evaluate X = (A + B) * (C + D) :

```
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:

- Can be found in a stack-organized computer
- Program to evaluate X = (A + B) * (C + D) :

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

Data Transfer Instructions

Typical Data Transfer Instructions

Name	Mnemonic
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

Data Transfer Instructions with Different Addressing Modes

Mode	Assembly Convention	Register Transfer
Direct address	LD ADR	$AC \leftarrow M[ADR]$
Indirect address	LD @ADR	$AC \leftarrow M[M[ADR]]$
Relative address	LD $ADR	$AC \leftarrow M[PC + ADR]$
Immediate operand	LD #NBR	$AC \leftarrow NBR$
Index addressing	LD ADR(X)	$AC \leftarrow M[ADR + XR]$
Register	LD R1	$AC \leftarrow R1$
Register indirect	LD (R1)	$AC \leftarrow M[R1]$
Autoincrement	LD (R1)+	$AC \leftarrow M[R1], R1 \leftarrow R1 + 1$
Autodecrement	LD -(R1)	$R1 \leftarrow R1 - 1, AC \leftarrow M[R1]$

Data Manipulation Instructions

JAIN DEEMED-TO-BE UNIVERSITY SCHOOL OF COMPUTER SCIENCE AND IT

5/9

Three Basic Types:	Arithmetic instructions
	Logical and bit manipulation instructions
	Shift instructions

Arithmetic Instructions

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with Carry	ADDC
Subtract with Borrow	SUBB
Negate(2's Complement)	NEG

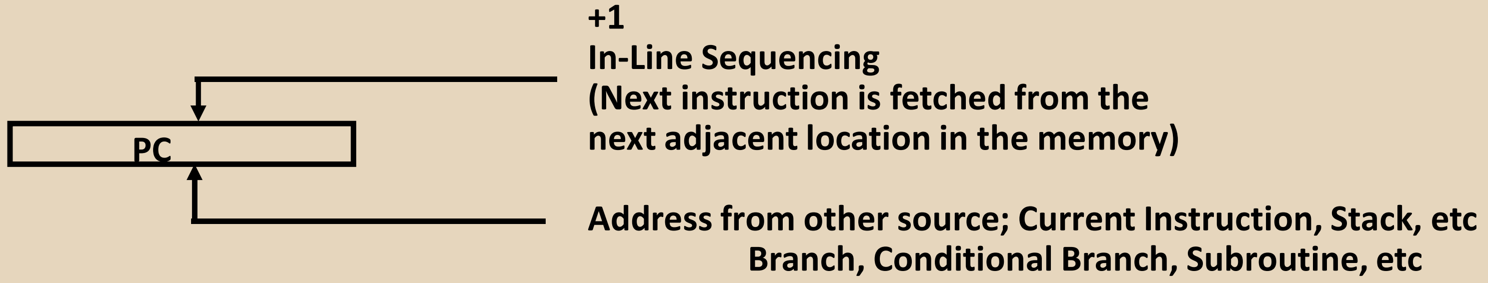
Logical and Bit Manipulation Instructions

Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-OR	XOR
Clear carry	CLRC
Set carry	SETC
Complement carry	COMC
Enable interrupt	EI
Disable interrupt	DI

Shift Instructions

Name	Mnemonic
Logical shift right	SHR
Logical shift left	SHL
Arithmetic shift right	SHRA
Arithmetic shift left	SHLA
Rotate right	ROR
Rotate left	ROL
Rotate right thru carry	RORC
Rotate left thru carry	ROLC

Program Control Instructions

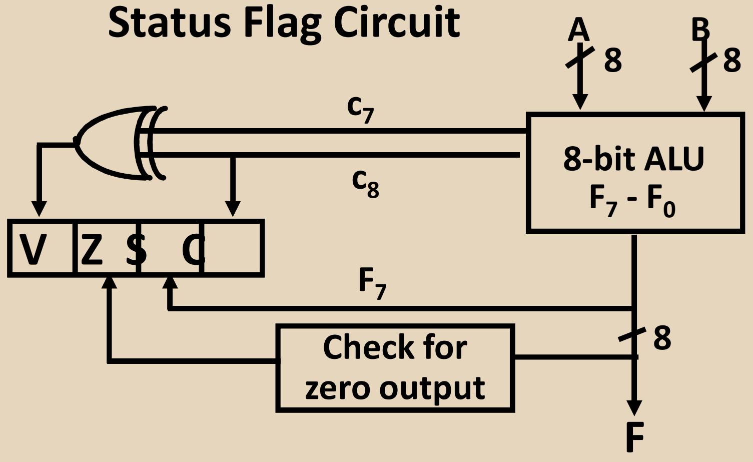

PC
+1
In-Line Sequencing
(Next instruction is fetched from the
next adjacent location in the memory)
Address from other source; Current Instruction, Stack, etc
Branch, Conditional Branch, Subroutine, etc

Program Control Instructions

Name	Mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RTN
Compare(by -)	CMP
Test (by AND)	TST

* CMP and TST instructions do not retain their results of operations(- and AND, respectively). They only set or clear certain Flags.

Status Flag Circuit



Conditional Branch Instructions

7/9

Mnemonic	Branch condition	Tested condition
BZ	Branch if zero	$Z = 1$
BNZ	Branch if not zero	$Z = 0$
BC	Branch if carry	$C = 1$
BNC	Branch if no carry	$C = 0$
BP	Branch if plus	$S = 0$
BM	Branch if minus	$S = 1$
BV	Branch if overflow	$V = 1$
BNV	Branch if no overflow	$V = 0$
Unsigned compare conditions (A - B)		
BHI	Branch if higher	$A > B$
BHE	Branch if higher or equal	$A \geq B$
BLO	Branch if lower	$A < B$
BLOE	Branch if lower or equal	$A \leq B$
BE	Branch if equal	$A = B$
BNE	Branch if not equal	$A \neq B$
Signed compare conditions (A - B)		
BGT	Branch if greater than	$A > B$
BGE	Branch if greater or equal	$A \geq B$
BLT	Branch if less than	$A < B$
BLE	Branch if less or equal	$A \leq B$
BE	Branch if equal	$A = B$
BNE	Branch if not equal	$A \neq B$

Subroutine Call And Return

SUBROUTINE CALL

Call subroutine
Jump to subroutine
Branch to subroutine
Branch and save return address

Two Most Important Operations are Implied;

* Branch to the beginning of the Subroutine
- Same as the Branch or Conditional Branch

* Save the Return Address to get the address of the location in the Calling Program upon exit from the Subroutine
- Locations for storing Return Address:
  - Fixed Location in the subroutine(Memory)
  - Fixed Location in memory
  - In a processor Register
  - In a memory stack
    - most efficient way

CALL
$SP \leftarrow SP - 1$
$M[SP] \leftarrow PC$
$PC \leftarrow EA$

RTN
$PC \leftarrow M[SP]$
$SP \leftarrow SP + 1$

Program Interrupt

Types of Interrupts:

External interrupts

External Interrupts initiated from the outside of CPU and Memory

- I/O Device -> Data transfer request or Data transfer complete
- Timing Device -> Timeout
- Power Failure

Internal interrupts (traps)

Internal Interrupts are caused by the currently running program

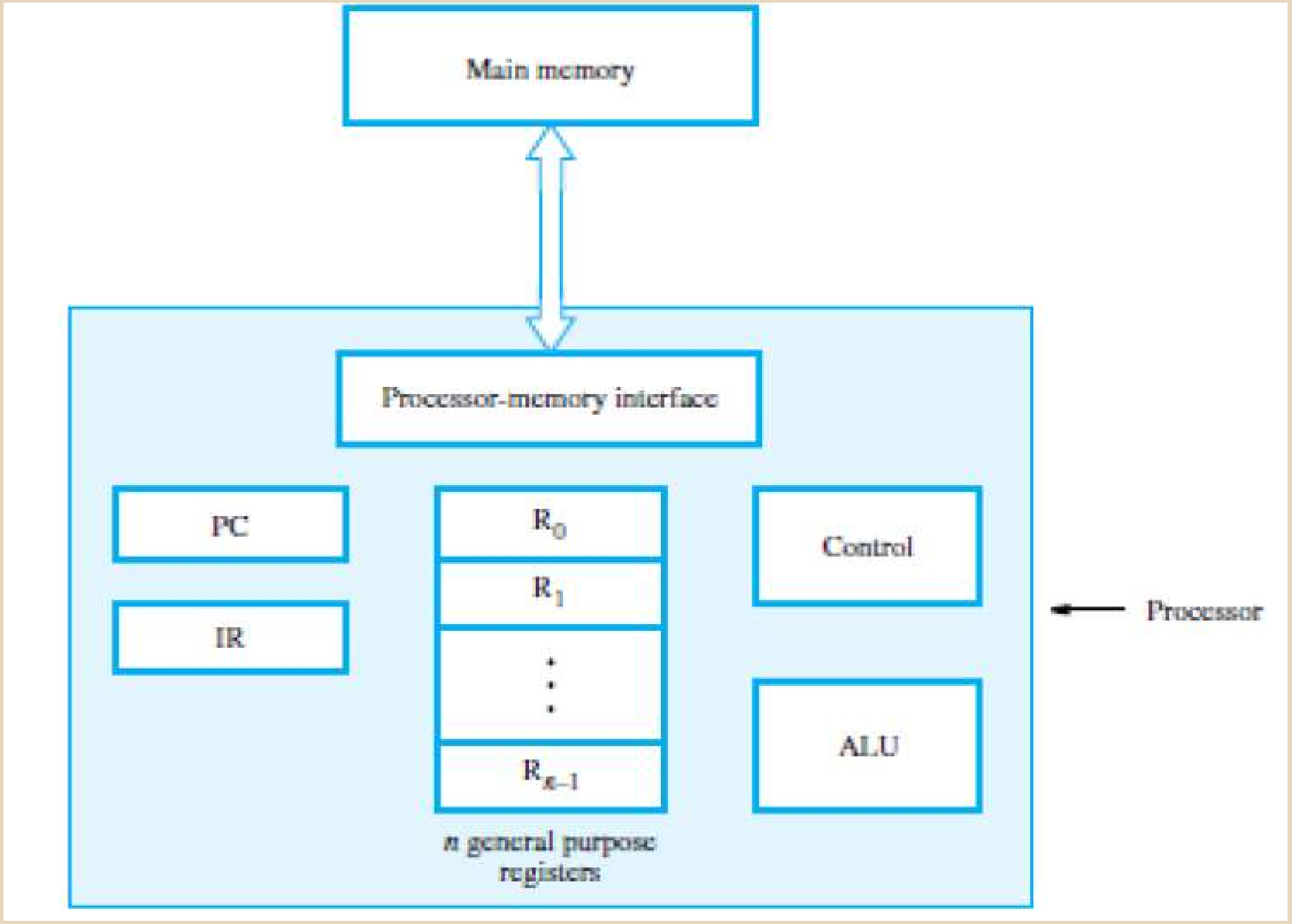
- Register, Stack Overflow
- Divide by zero
- OP-code Violation
- Protection Violation

Software Interrupts

Both External and Internal Interrupts are initiated by the computer Hardware.
Software Interrupts are initiated by executing an instruction.

- Supervisor Call -> Switching from a user mode to the supervisor mode
  -> Allows to execute a certain class of operations
  which are not allowed in the user mode

Connection b/w Processor and Memory 1/2



Connection b/w Processor and Memory

- ✓ The **instruction register** (IR) holds the instruction that is currently being executed. Its output is available to the control circuits, which generate the timing signals that control the various processing elements involved in executing the instruction.
- ✓ The **program counter** (PC) is another specialized register. It contains the memory address of the next instruction to be fetched and executed. During the execution of an instruction, the contents of the PC are updated to correspond to the address of the next instruction to be executed.
- ✓ The **general-purpose registers** (GPRs) R0 through Rn−1, often called processor registers

Addressing Modes

Addressing Modes:

* Specifies a rule for interpreting or modifying the address field of the instruction (before the operand is actually referenced)

* Variety of addressing modes

- to give programming flexibility to the user
- to use the bits in the address field of the instruction efficiently

Types Of Addressing Modes

2/7

Implied Mode

Address of the operands are specified implicitly
in the definition of the instruction
- No need to specify address in the instruction
- EA = AC, or EA = Stack[SP], **EA: Effective Address.**

Immediate Mode

Instead of specifying the address of the operand,
operand itself is specified
- No need to specify address in the instruction
- However, operand itself needs to be specified
- Sometimes, require more bits than the address
- Fast to acquire an operand

Register Mode

Address specified in the instruction is the register address
- Designated operand need to be in a register
- Shorter address than the memory address
- Saving address field in the instruction
- Faster to acquire an operand than the memory addressing
- EA = IR(R) (IR(R): Register field of IR)

Types Of Addressing Modes

3/7

Register Indirect Mode

Instruction specifies a register which contains the memory address of the operand

- Saving instruction bits since register address is shorter than the memory address
- Slower to acquire an operand than both the register addressing or memory addressing
- EA = [IR(R)] ([x]: Content of x)

Auto-increment or Auto-decrement features:

Same as the Register Indirect, but:

- When the address in the register is used to access memory, the value in the register is incremented or decremented by 1 (after or before the execution of the instruction)

Types Of Addressing Modes

4/7

Direct Address Mode

Instruction specifies the memory address which can be used directly to the physical memory

- Faster than the other memory addressing modes
- Too many bits are needed to specify the address for a large physical memory space
- EA = IR(address), (IR(address): address field of IR)

Indirect Addressing Mode

The address field of an instruction specifies the address of a memory location that contains the address of the operand

- When the abbreviated address is used, large physical memory can be addressed with a relatively small number of bits
- Slow to acquire an operand because of an additional memory access
- EA = M[IR(address)]

Types Of Addressing Modes

Relative Addressing Modes

The Address fields of an instruction specifies the part of the address (abbreviated address) which can be used along with a designated register to calculate the address of the operand

PC Relative Addressing Mode(R = PC)
- EA = PC + IR(address)

- Address field of the instruction is short
- Large physical memory can be accessed with a small number of address bits

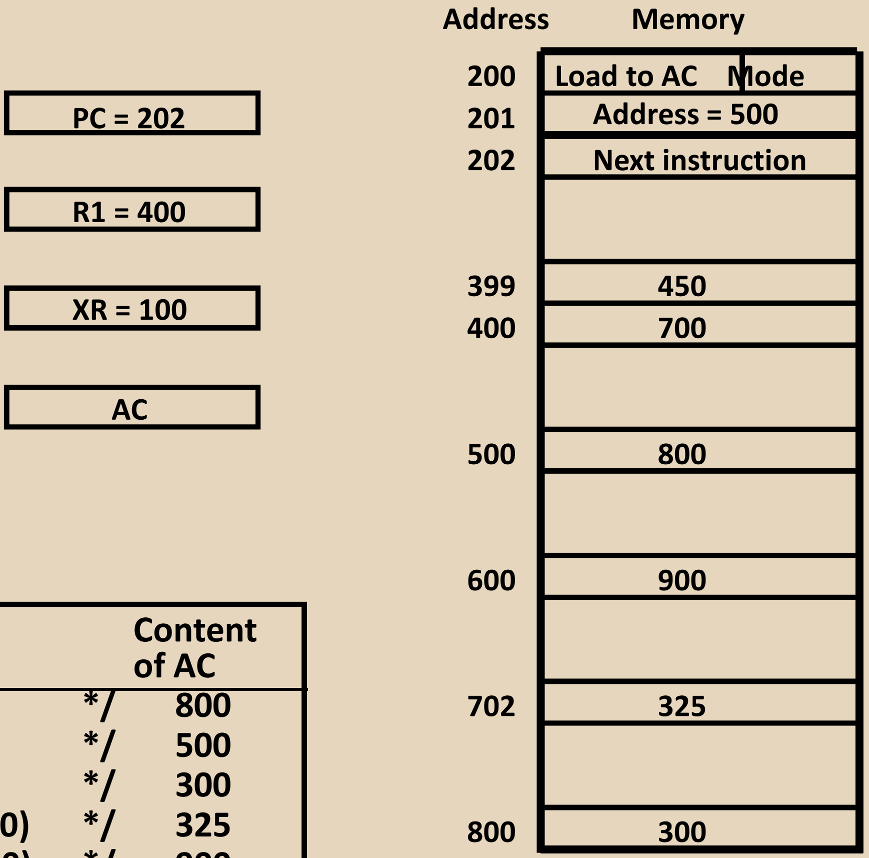
Indexed Addressing Mode

XR: Index Register:
- EA = XR + IR(address)

Base Register Addressing Mode

BAR: Base Address Register:
- EA = BAR + IR(address)

Addressing Modes - Examples



Addressing Mode	Effective Address		Content of AC
Direct address	500	/* AC ← (500) */	800
Immediate operand	-	/* AC ← 500 */	500
Indirect address	800	/* AC ← ((500)) */	300
Relative address	702	/* AC ← (PC+500) */	325
Indexed address	600	/* AC ← (XR+500) */	900
Register	-	/* AC ← R1 */	400
Register indirect	400	/* AC ← (R1) */	700
Auto increment	400	/* AC ← (R1)+ */	700
Auto decrement	399	/* AC ← -(R) */	450

Addressing Mode – Applications

7/7

1. **Immediate addressing mode:** Used to set an initial value for a register. The value is usually a constant
2. **Register addressing mode/direct addressing mode:** Used to implement variables and access static data
3. **Register indirect addressing mode/indirect addressing mode:** Used to pass an array as a parameter and to implement pointers
4. **Relative addressing mode:** Used to relocate programs at run time and to change the execution order of instruction
5. **Index addressing mode:** Used to implement arrays
6. **Base register addressing mode:** Used to write codes that are relocatable and for handling recursion
7. **Auto-increment/decrement addressing mode:** Used to implement loops and stacks