M.S. Ramaiah Institute of Technology
(Autonomous Institute, Affiliated to VTU)
Department of Computer Science and Engineering

Course Name: Computer Organization and Architecture

Course Code: CS45

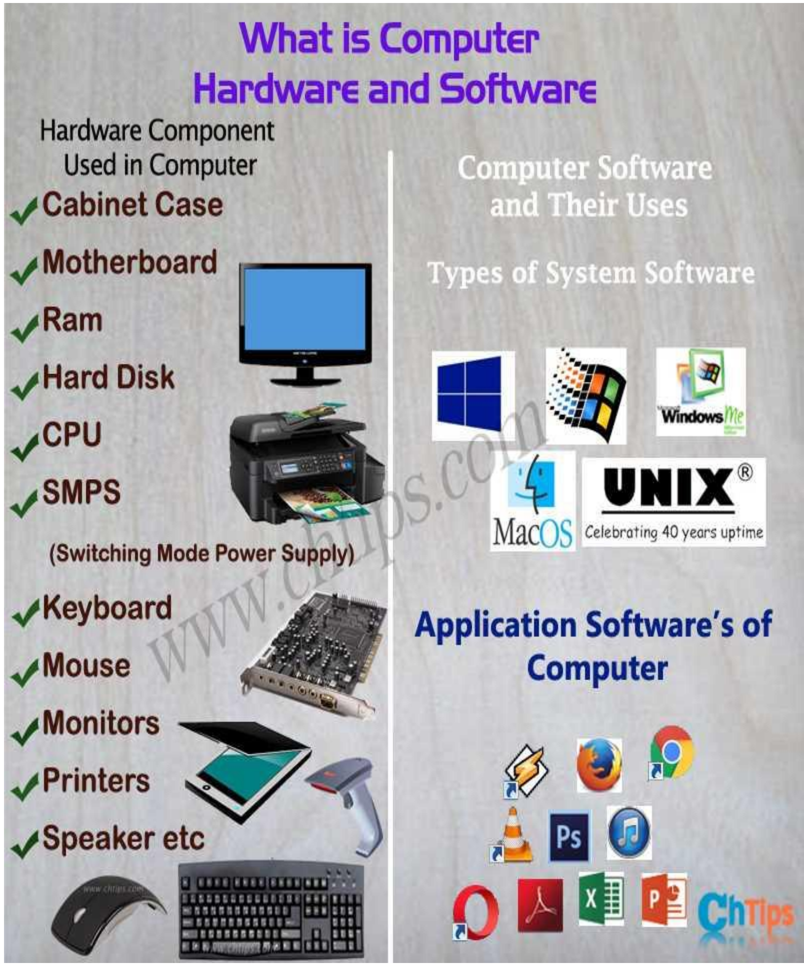
Unit -1

- Functional units, Bus structures, performance,
- **Overflow in integer arithmetic:** Numbers, Arithmetic operations and characters
- Memory locations and addresses,
- Memory operations,
- Instructions and instruction sequencing,
- Addressing modes,
- Subroutines and use of stack frames,
- Encoding of machine instructions.

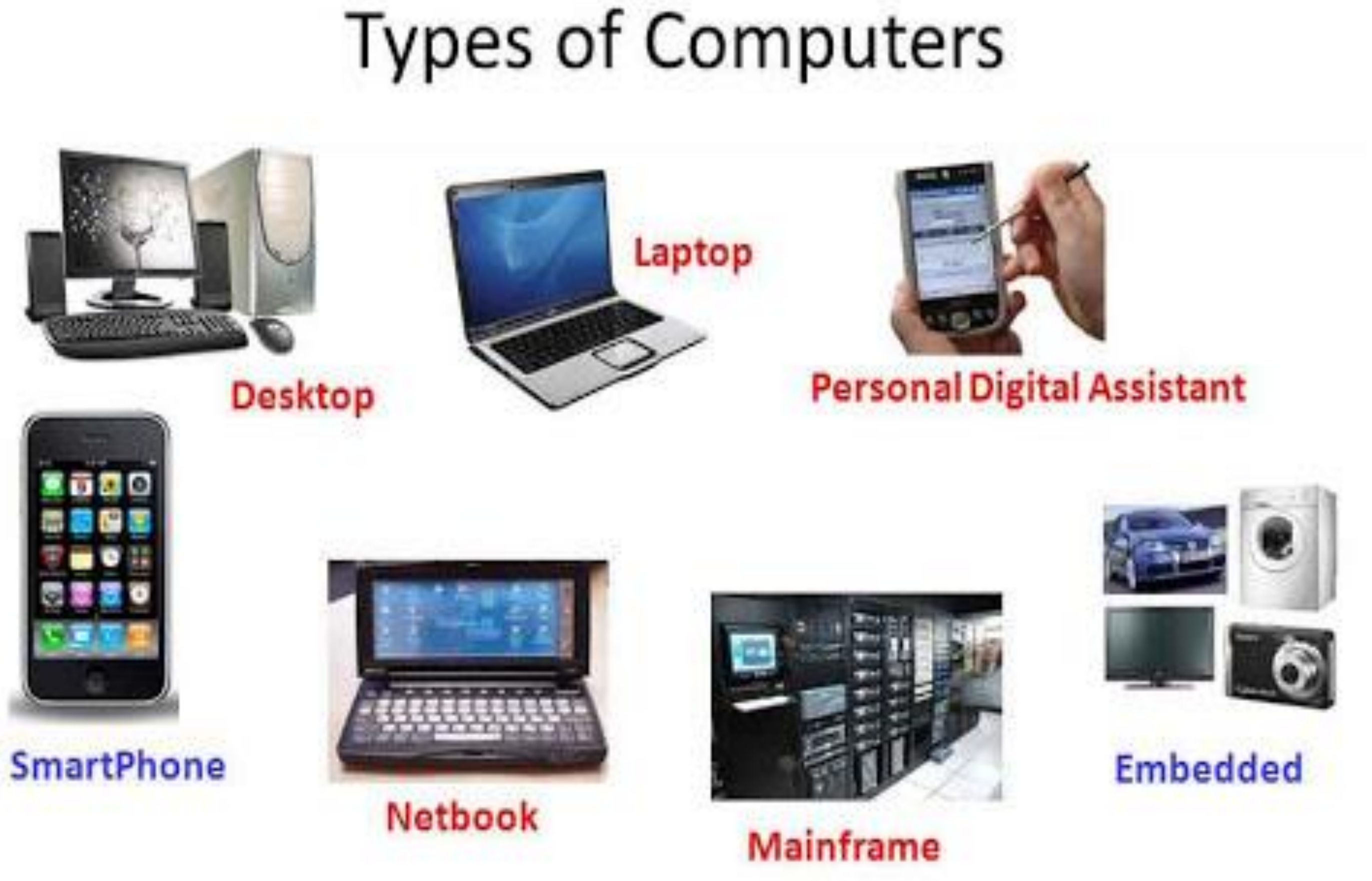
Introduction

What is Computer

- □ All computers have in common: **hardware** and **software.**
  - ? Hardware is any part of your computer that has a physical structure, such as the keyboard or mouse.
  - ? Software is any set of instructions that tells the hardware what to do and how to do it. Examples of software include web browsers, games, and word processors.



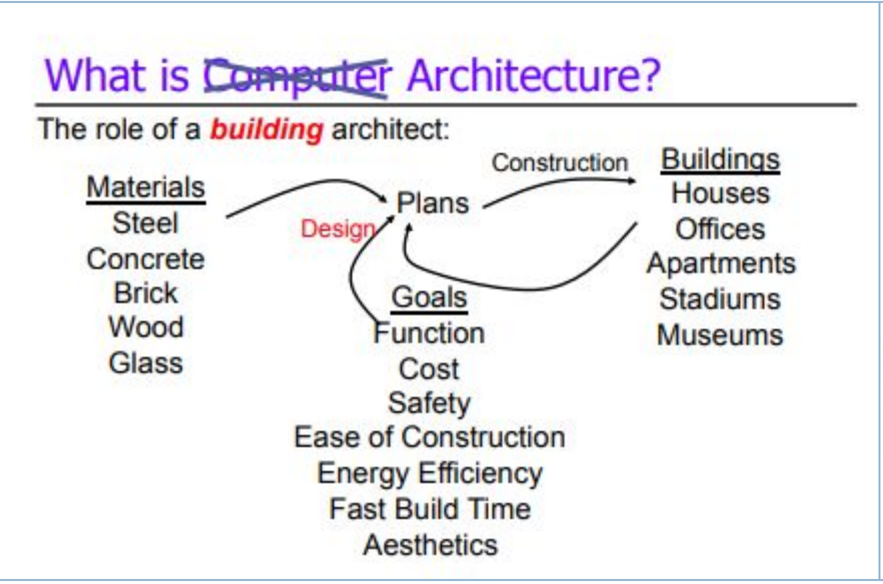
What are the different types of computers?



Analogy

- A chef prepares a certain recipe, then serves it to the customers.
- Chef knows how to prepare the food item whereas customer cares only about quality and taste of the food.
- **“customer” as computer organization**
- **“chef”** can referred to as **computer architecture**

Analogy



What is ~~Computer~~ Architecture?

The role of a ***building*** architect:

Materials
Steel
Concrete
Brick
Wood
Glass

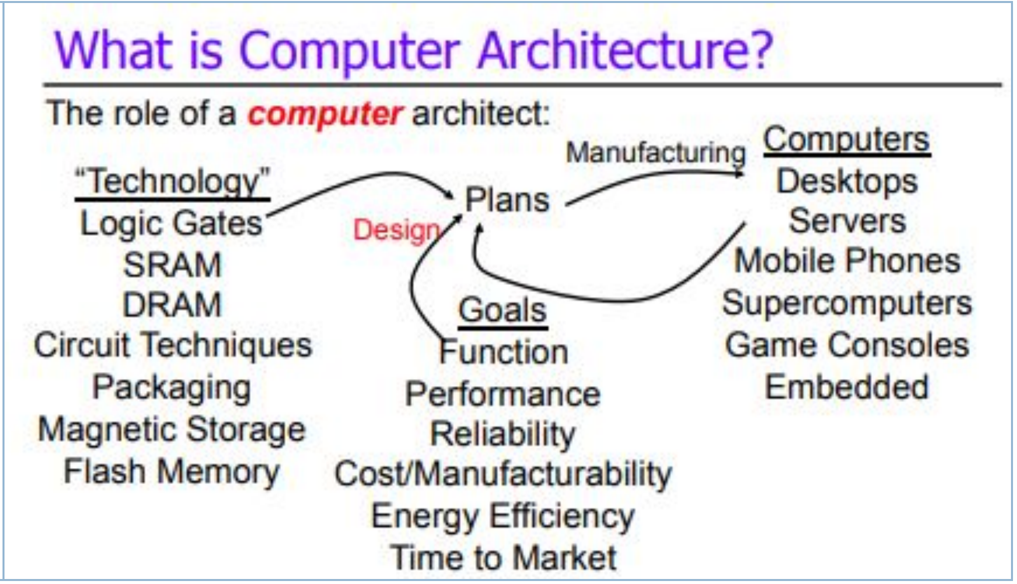
Design

Plans

Construction

Buildings
Houses
Offices
Apartments
Stadiums
Museums

Goals
Function
Cost
Safety
Ease of Construction
Energy Efficiency
Fast Build Time
Aesthetics



What is Computer Architecture?

The role of a ***computer*** architect:

"Technology"
Logic Gates
SRAM
DRAM
Circuit Techniques
Packaging
Magnetic Storage
Flash Memory

Design

Plans

Manufacturing

Computers
Desktops
Servers
Mobile Phones
Supercomputers
Game Consoles
Embedded

Goals
Function
Performance
Reliability
Cost/Manufacturability
Energy Efficiency
Time to Market

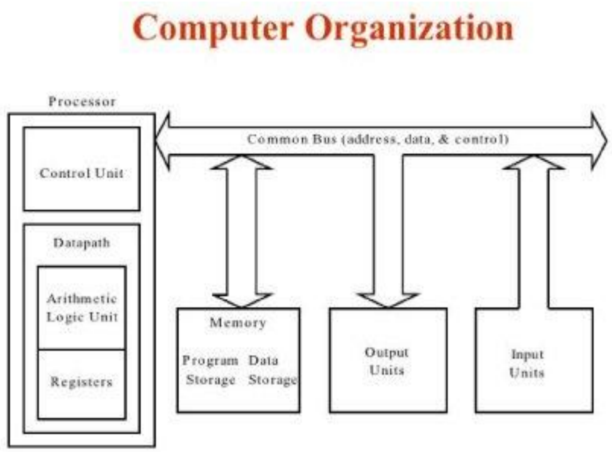
Computer organization

- **Computer Organization-** The way hardware components are connected to form a computer system
- In other words, it is mainly about the programmer’s or user point of view.
- Organization - physical design of a computer

  1.How many registers?

  2. What is a register?

  3. How many registers does a typical CPU have



Computer Architecture

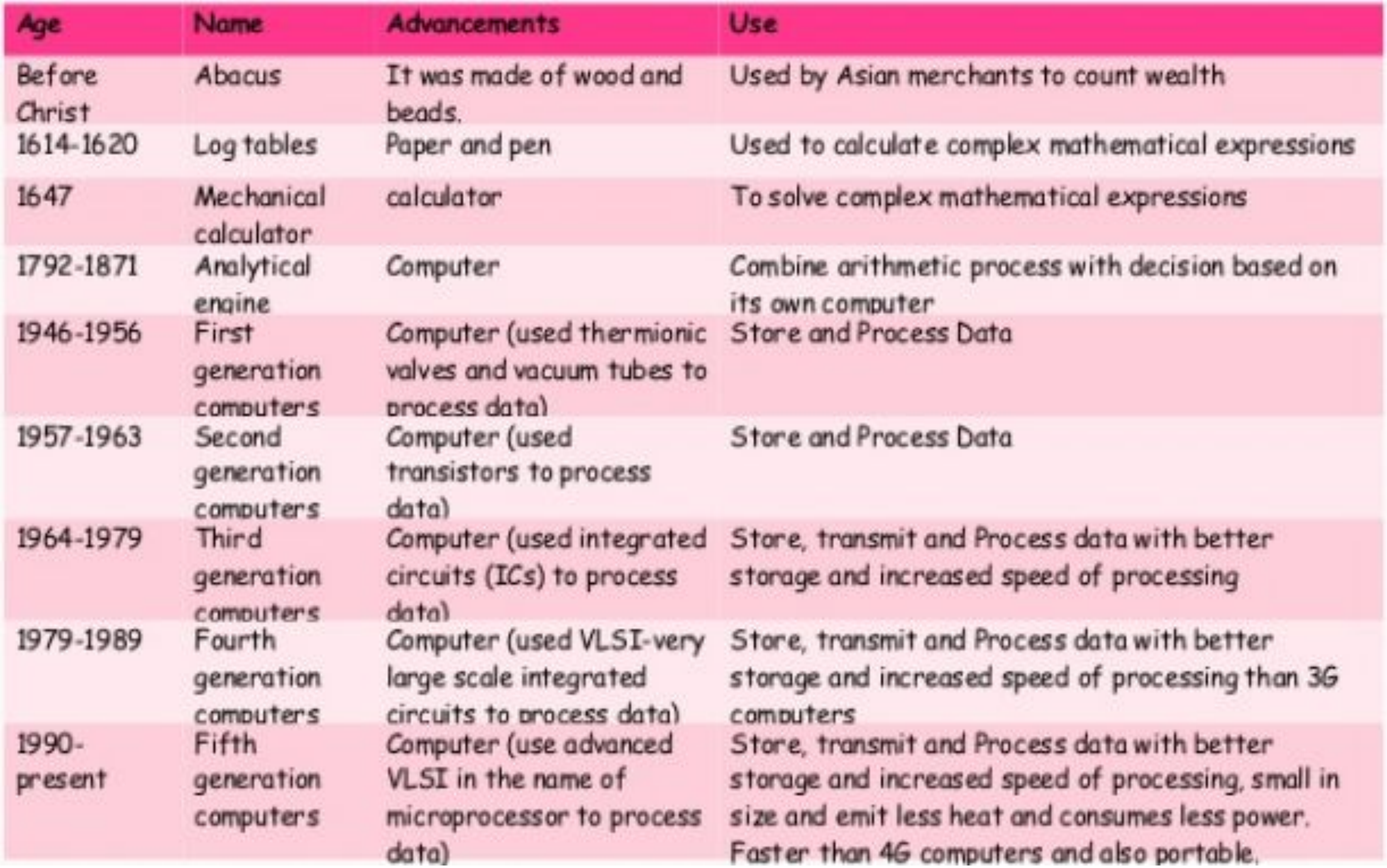
- Structure and behaviour of the various functional units of the computer and their interactions
- Basically, throws light on the designer's point of view.

Computer Architecture

- In a system, there are a set of instructions, it is enough for programmer or user to know what are a set of instructions present in case of **computer organization**

- System designer worries about how a set of instructions are implemented, algorithm of implementation is the emphasis in the case of architectural studies.

Generations of Computer



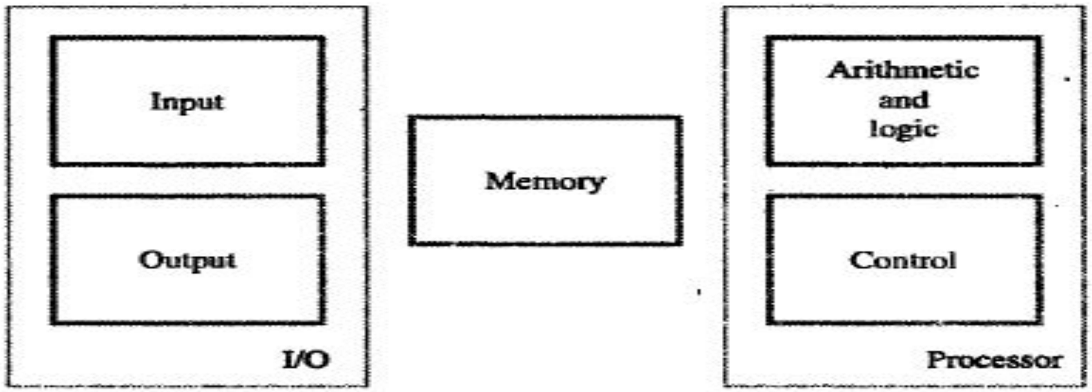
Age	Name	Advancements	Use
Before Christ	Abacus	It was made of wood and beads.	Used by Asian merchants to count wealth
1614-1620	Log tables	Paper and pen	Used to calculate complex mathematical expressions
1647	Mechanical calculator	calculator	To solve complex mathematical expressions
1792-1871	Analytical engine	Computer	Combine arithmetic process with decision based on its own computer
1946-1956	First generation computers	Computer (used thermionic valves and vacuum tubes to process data)	Store and Process Data
1957-1963	Second generation computers	Computer (used transistors to process data)	Store and Process Data
1964-1979	Third generation computers	Computer (used integrated circuits (ICs) to process data)	Store, transmit and Process data with better storage and increased speed of processing
1979-1989	Fourth generation computers	Computer (used VLSI-very large scale integrated circuits to process data)	Store, transmit and Process data with better storage and increased speed of processing than 3G computers
1990-present	Fifth generation computers	Computer (use advanced VLSI in the name of microprocessor to process data)	Store, transmit and Process data with better storage and increased speed of processing, small in size and emit less heat and consumes less power. Faster than 4G computers and also portable.

Unit -1

- Functional units, Bus structures, performance,
- **Overflow in integer arithmetic**: Numbers, Arithmetic operations and characters
- Memory locations and addresses,
- Memory operations,
- Instructions and instruction sequencing,
- Addressing modes,
- Subroutines and use of stack frames,
- Encoding of machine instructions.

Functional Units

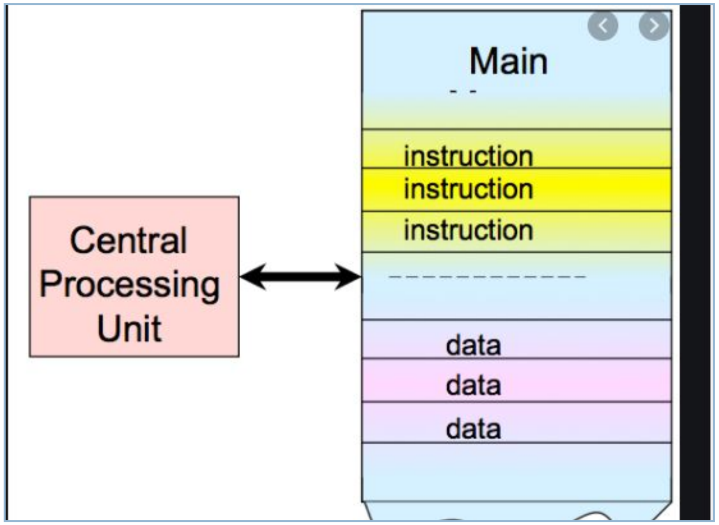
- A Computer consist of 5 main parts



Basic functional units of a computer.

Functional Units

- A **computer program** is a collection of instructions that can be executed by a **computer** to perform a specific task.



Functional Units

- **Central Processing Unit (CPU) consists of the following features**
  - CPU is considered as the brain of the computer.
  - CPU performs all types of data processing operations.
  - It stores data, intermediate results, and instructions (program).
  - It controls the operation of all parts of the computer.
  - 3 components inside CPU
    - Registers
    - Control unit
    - ALU(Arithmetic Logic Unit)

Functional Units

ALU (Arithmetic Logic Unit)

This unit consists of two subsections namely,

? Arithmetic section

? LogicSection

- **Arithmetic section**

  The function of arithmetic section is to perform arithmetic operations like addition, subtraction, multiplication, and division.

- **Logic Section**

  The function of the logic section is to perform logic operations such as comparing, selecting, matching, and merging of data.

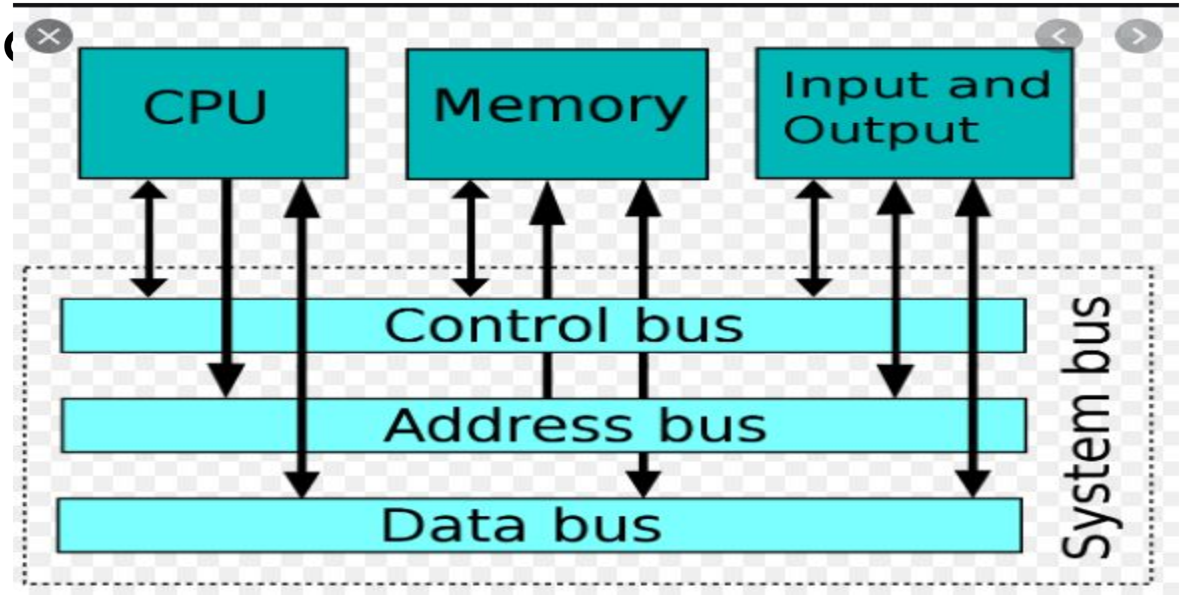
Functional Units

Control unit

- Functions of this unit are
- It is responsible for controlling the transfer of data and instructions among other units of a computer.
- It manages and coordinates all the units of the computer.
- It obtains the instructions from the memory, interprets them, and directs the operation of the computer.
- It communicates with Input/output devices for transfer of data or results from storage.
- It does not process or store data.

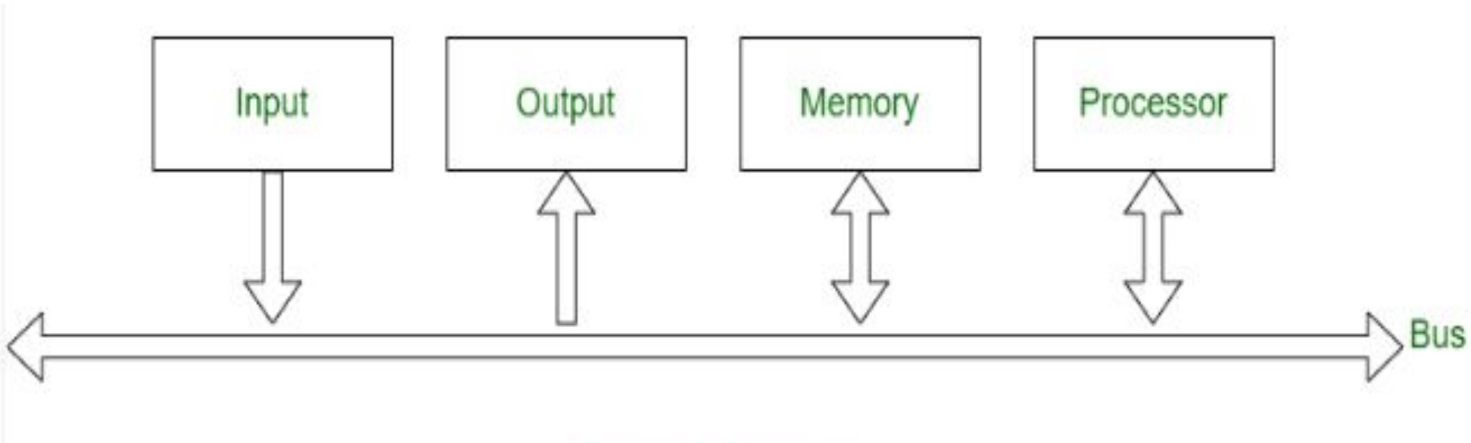
Bus Structure

- There are many ways to connect different parts inside a computer together.
- A group of lines that serves as a connecting path for several devices is called a *bus*.
- Address/



Bus Structure

- Single-bus



Single Bus Structure

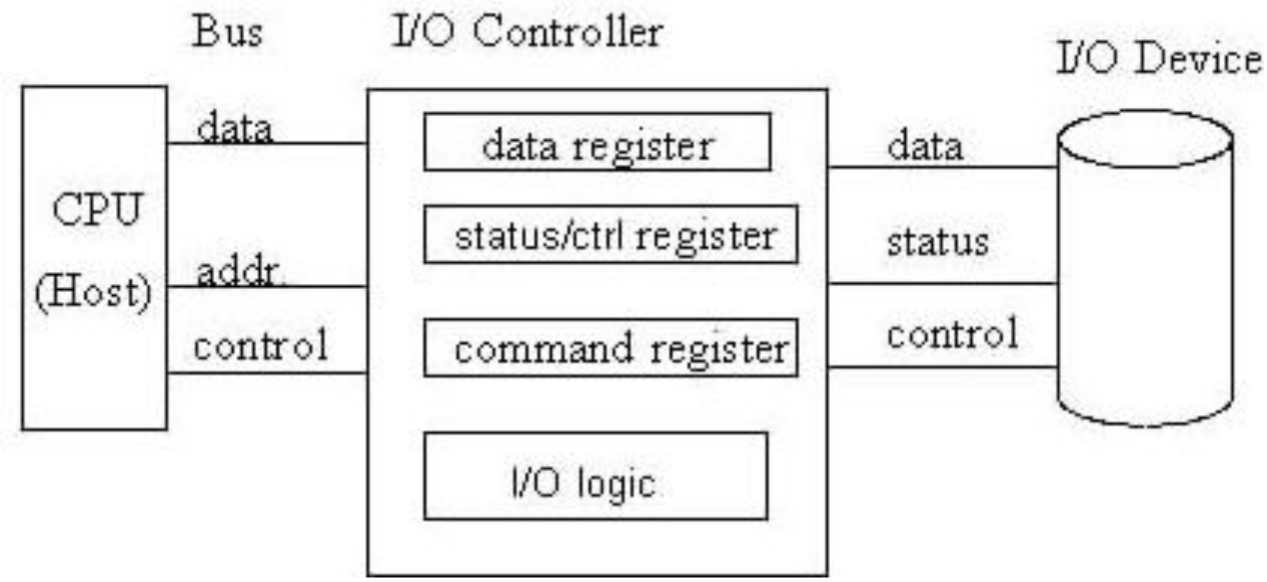
Bus Structure

Speed Issue

- Different devices have different transfer/operate speed.
  - Keyboard, printers are slow
  - Magnetic or optical disk are fast
  - Memory and processor are faster
- If the speed of bus is bounded by the slowest device connected to it, the efficiency will be very low.
- How to solve this?
  - ? A common approach – use buffers.

Bus Structure

- A common approach – use buffers
  - Buffer register with every device to hold information using data transfer

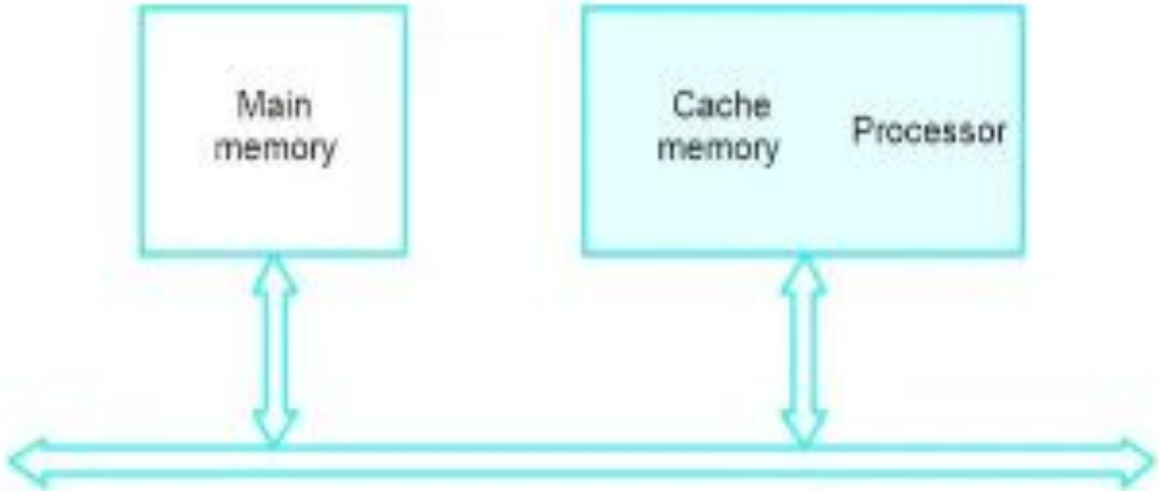


Performance

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

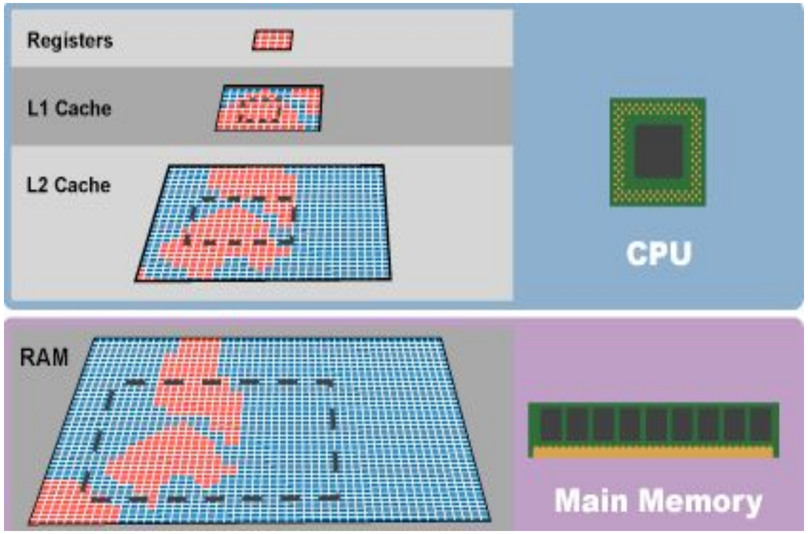
Performance – Hardware design

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



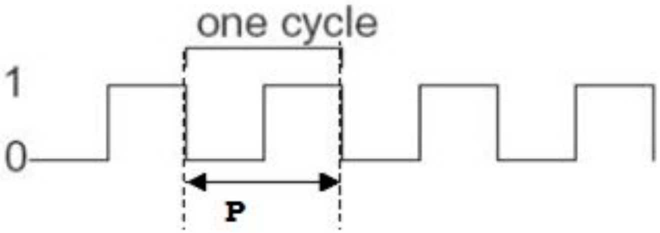
Performance – Hardware design

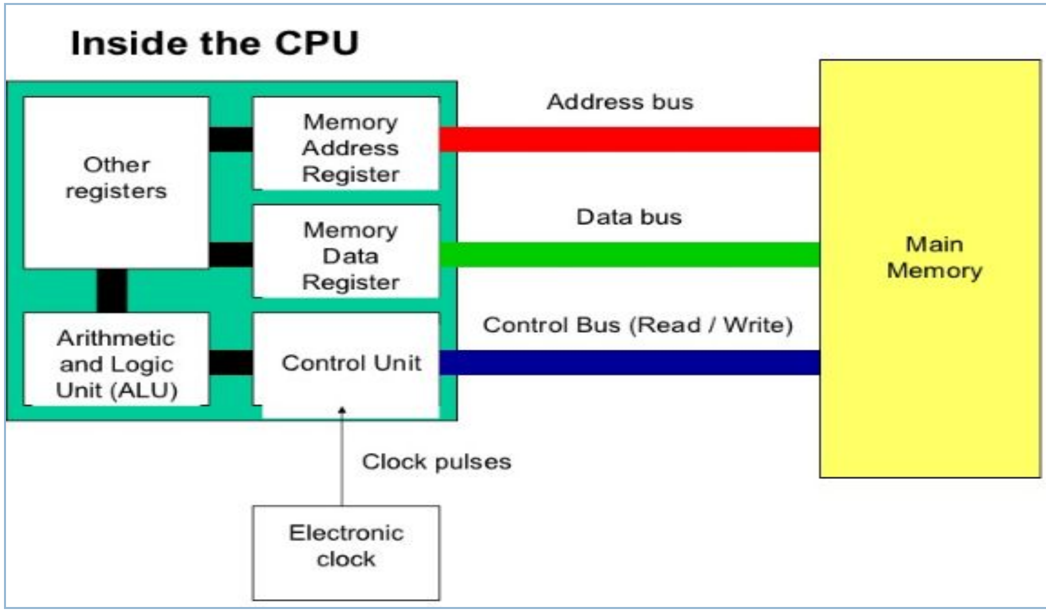
- The processor and a relatively small cache memory can be fabricated on a single integrated circuit chip.
- A cache is a special storage space for temporary files that makes a device run faster and more efficiently.
  - Speed
  - Cost
  - Memory management



Performance – Hardware design

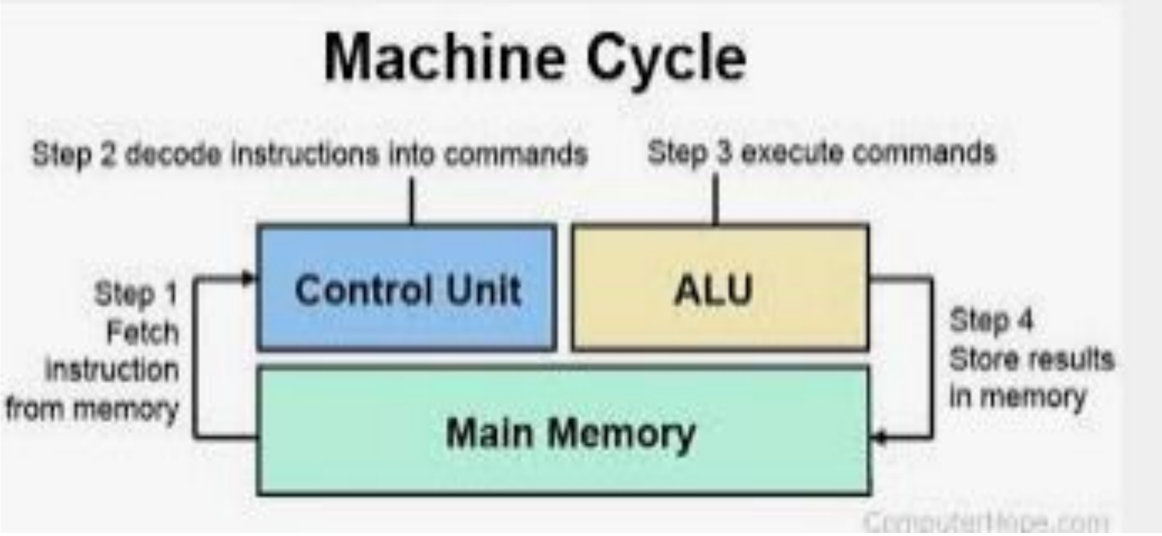
- Processor circuits are controlled by a timing signal called clock.
- Clock – defines regular time intervals called clock cycles
- P – length of once clock cycle





Performance – Hardware design

- To execute a machine instruction, processor divides the action into 4 steps.
- Each step take once clock cycle .


Machine Cycle
Step 2 decode instructions into commands
Step 3 execute commands
Control Unit
ALU
Step 1 Fetch instruction from memory
Step 4 Store results in memory
Main Memory
ComputerHope.com

Performance – Hardware design

- Clock rate(R)/Processor Speed
  - Is a measure of number of clock cycles per second.

$$R = \frac{1}{P}$$

- CPU clock speed, ate, is measured in Hertz — generally in gigahertz, or GHz.

Performance – Hardware design

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
- S – average number of basic steps needed to execute one machine instruction. Each step completes in one clock cycle
- R – clock rate

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

1) A program contains 1000 instruction out of that 25% instruction requires 4 clock cycles. 40% instructions requires 5 clock and remaining 3 clock cycles for execution. Find the total time required to execute the program running in a 1 GHz machine

Soln

$N = 1000$

$S =$ 25% of N ⇒ 250 instruction require 4 clock cycle
40% of N ⇒ 400 instruction require 5 clock cycle
35% of N ⇒ 350 instruction requires 3 clock cycle

$R = 1\,\text{GHz} \Rightarrow 1 \times 10^9\ \text{Hz}$

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

$$= \frac{(250 * 4) + (400 * 5) + (350 * 3)}{1 \times 10^9}$$

$$= 4050 / 1 \times 10^9 \quad = \quad 4.05 \times 10^{-6} \Rightarrow \boxed{4.05\,\mu s}$$

Performance – Hardware design

Basic Performance Equation

How to improve T?

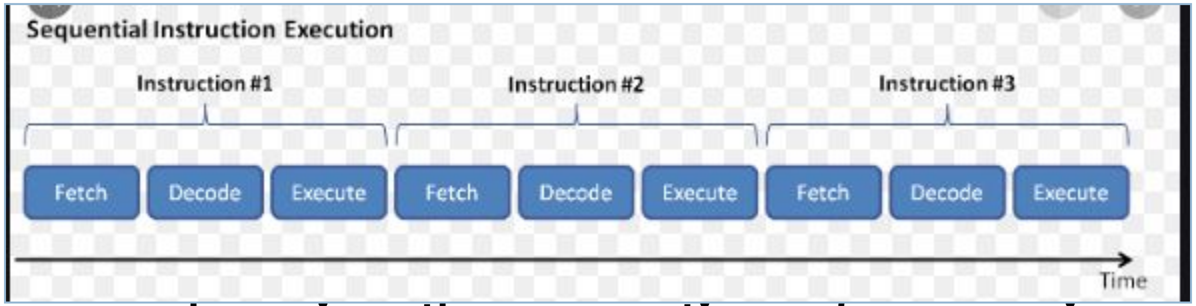
- Pipelining and Superscalar operation
- Clock rate

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

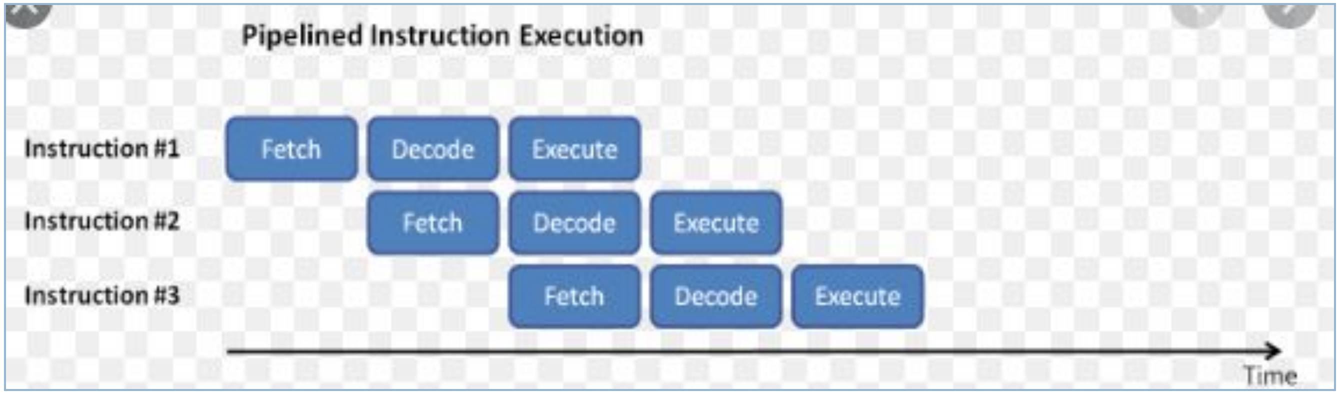
Performance – Hardware design

Pipelining

- Instructions are not necessarily executed one after another.

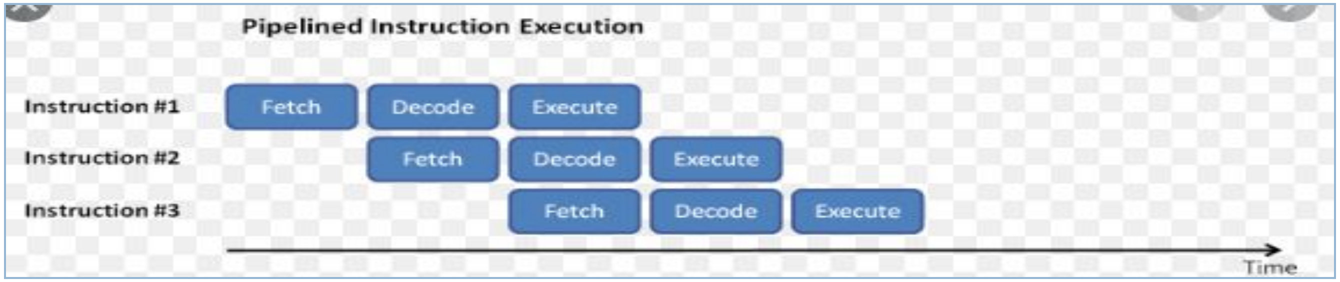


- Pipelining – overlapping the execution of successive instructions.



Performance – Hardware design

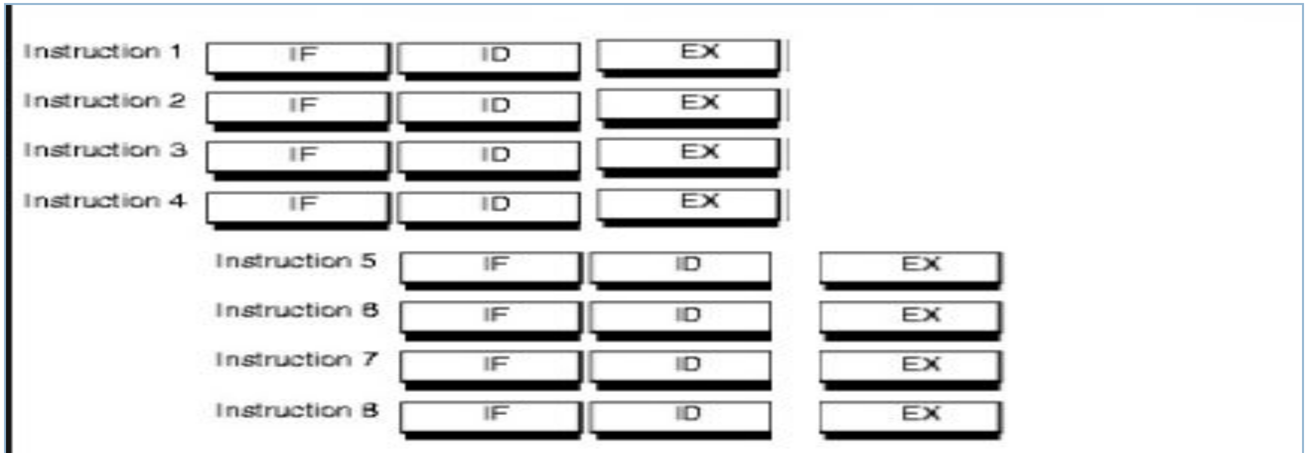
- Pipelining – overlapping the execution of successive instructions.



Pipelined Instruction Execution

Instruction #1	Fetch	Decode	Execute		
Instruction #2		Fetch	Decode	Execute	
Instruction #3			Fetch	Decode	Execute

- Superscalar – Different instruction can be executed in parallel



Instruction 1	IF	ID	EX	
Instruction 2	IF	ID	EX	
Instruction 3	IF	ID	EX	
Instruction 4	IF	ID	EX	
Instruction 5		IF	ID	EX
Instruction 6		IF	ID	EX
Instruction 7		IF	ID	EX
Instruction 8		IF	ID	EX

Performance – Hardware design

Clock Rate

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

- 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)

Performance – Instruction set

- Processor can have simple instruction and complex instruction
- Simple instruction requires a small number of steps to execute.
- Complex instruction involve a large number of steps

- The design of Instruction set of a Processor can be
  - Reduced Instruction Set Computer(RISC)
  - Complex Instruction Set Computer(CISC)

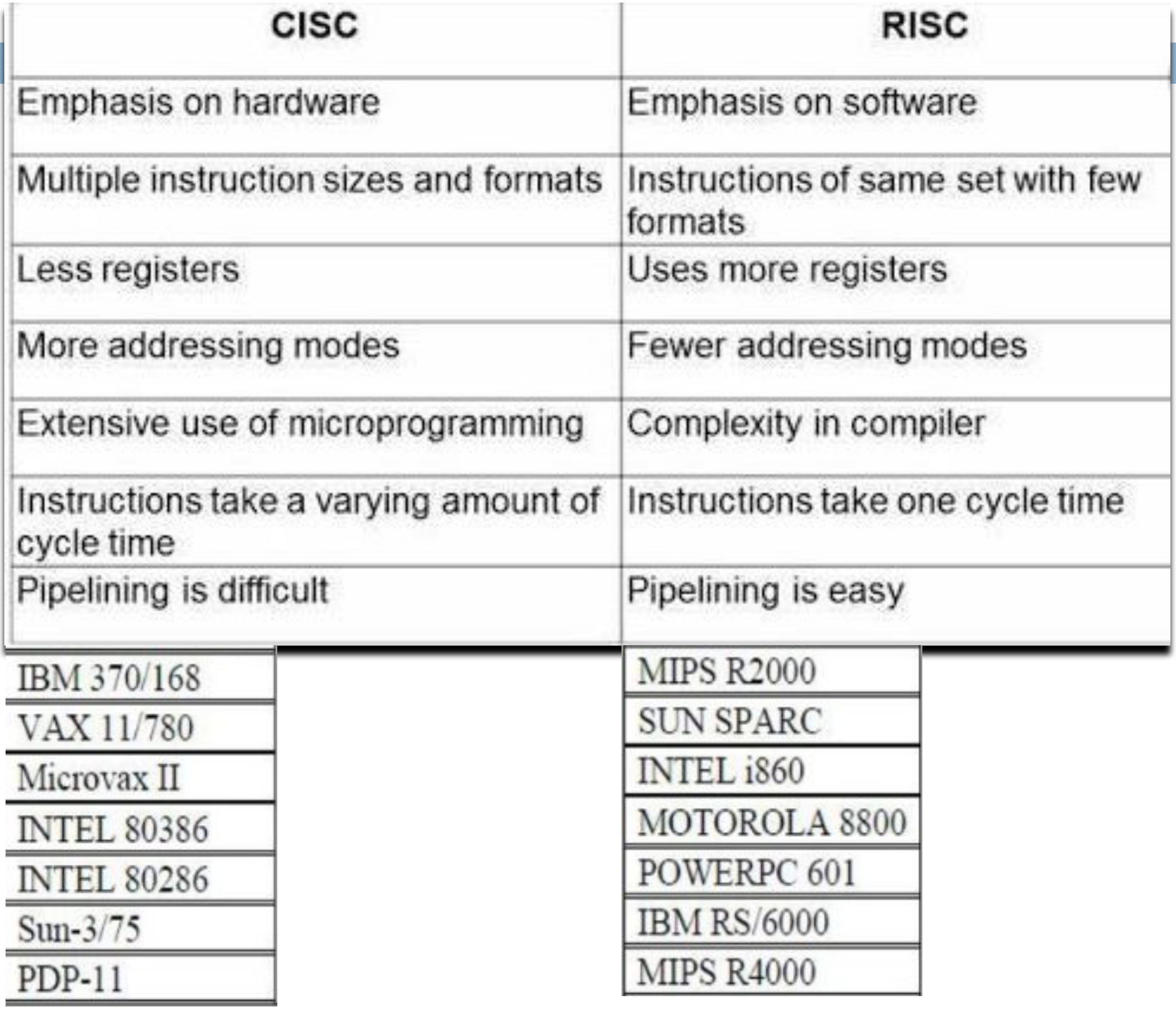
Performance – Instruction set

- If a processor has only Simple instructions – Program will have large number of instructions
  - N – large value
  - S – small value

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

- If a processor has only Complex instructions – Program will have lesser number of instructions
  - N – small value
  - S – large value

Performance – Instruction set



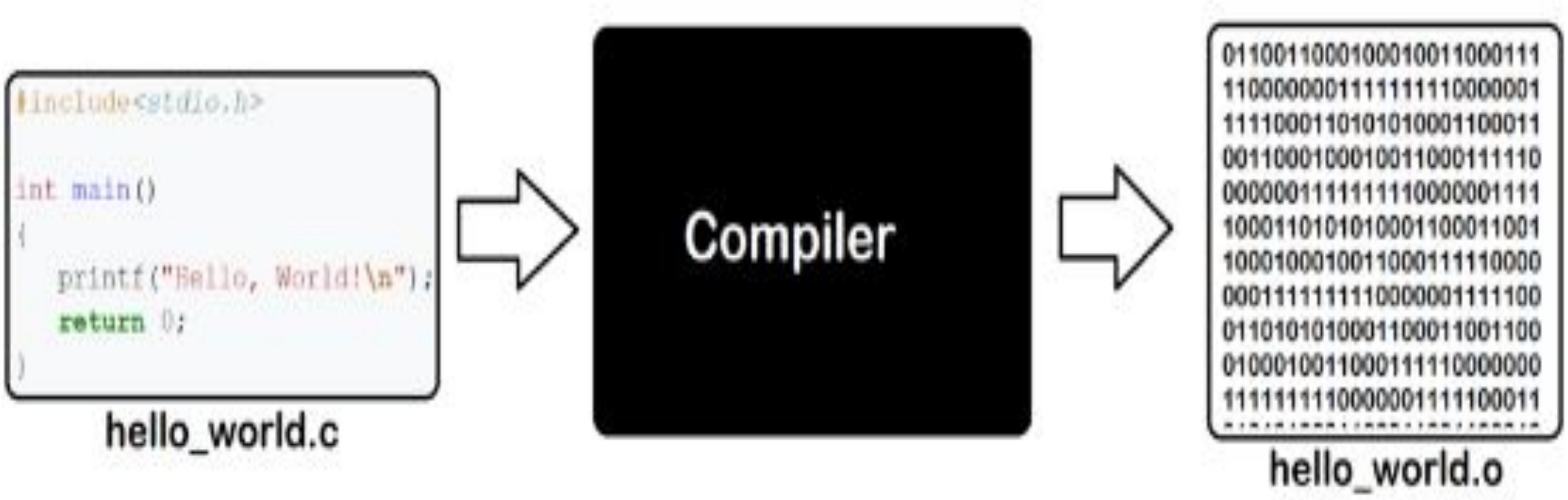
CISC	RISC
Emphasis on hardware	Emphasis on software
Multiple instruction sizes and formats	Instructions of same set with few formats
Less registers	Uses more registers
More addressing modes	Fewer addressing modes
Extensive use of microprogramming	Complexity in compiler
Instructions take a varying amount of cycle time	Instructions take one cycle time
Pipelining is difficult	Pipelining is easy

IBM 370/168
VAX 11/780
Microvax II
INTEL 80386
INTEL 80286
Sun-3/75
PDP-11

MIPS R2000
SUN SPARC
INTEL i860
MOTOROLA 8800
POWERPC 601
IBM RS/6000
MIPS R4000

Performance – Compiler

- A compiler translates a high-level language program into a sequence of machine instructions.



```
#include<stdio.h>

int main()
{
   printf("Hello, World!\n");
   return 0;
}
```

```
0110011000100010011000111
110000000111111111110000001
1111000110101010001100011
0011000100010011000111110
00000011111111110000001111
1000110101010001100011001
1000100010011000111110000
00011111111110000001111100
0110101010001100011001100
0100010011000111110000000
11111111100000001111100011
```

Performance – Compiler

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

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

Performance – Measurement

- T is difficult to compute

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

- So, we can 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.

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

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

Performance Summary

METRICS

- Processor clock
- Basic performance equation
- Pipelining & super scalar operation
- Clock rate
- Instruction set CISC & RISC
- Compiler
- Performance measurement

Unit -1

- Functional units, Bus structures, performance,
- **Overflow in integer arithmetic:** Numbers, Arithmetic operations and characters
- Memory locations and addresses,
- Memory operations,
- Instructions and instruction sequencing,
- Addressing modes,
- Subroutines and use of stack frames,
- Encoding of machine instructions.

Number, Arithmetic Operations, and Characters

Let us learn

- Binary number representation
- Arithmetic operation on these number
- Character representation

Number, Arithmetic Operations, and Characters

- Computers are built using logic circuits that operate on information represented by two values 0 and 1.
- The bit of information stands for binary digit
- Represent number □ string of bits called binary number
- Represent Text character □ string of bits called character code

Number, Arithmetic Operations, and Characters

- Integer number are represented in two forms
  - Signed integer
  - Unsigned integer
- Unsigned integer number represent positive numbers.
- Computer does not have provision to represent negative sign, so various techniques are used

Number, Arithmetic Operations, and Characters

- Computers are built using logic circuits that operate on information represented by two values 0 and 1.
- The bit of information stands for binary digit
- Represent number □ string of bits called binary number
- Represent Text character □ string of bits called character code

Number, Arithmetic Operations, and Characters

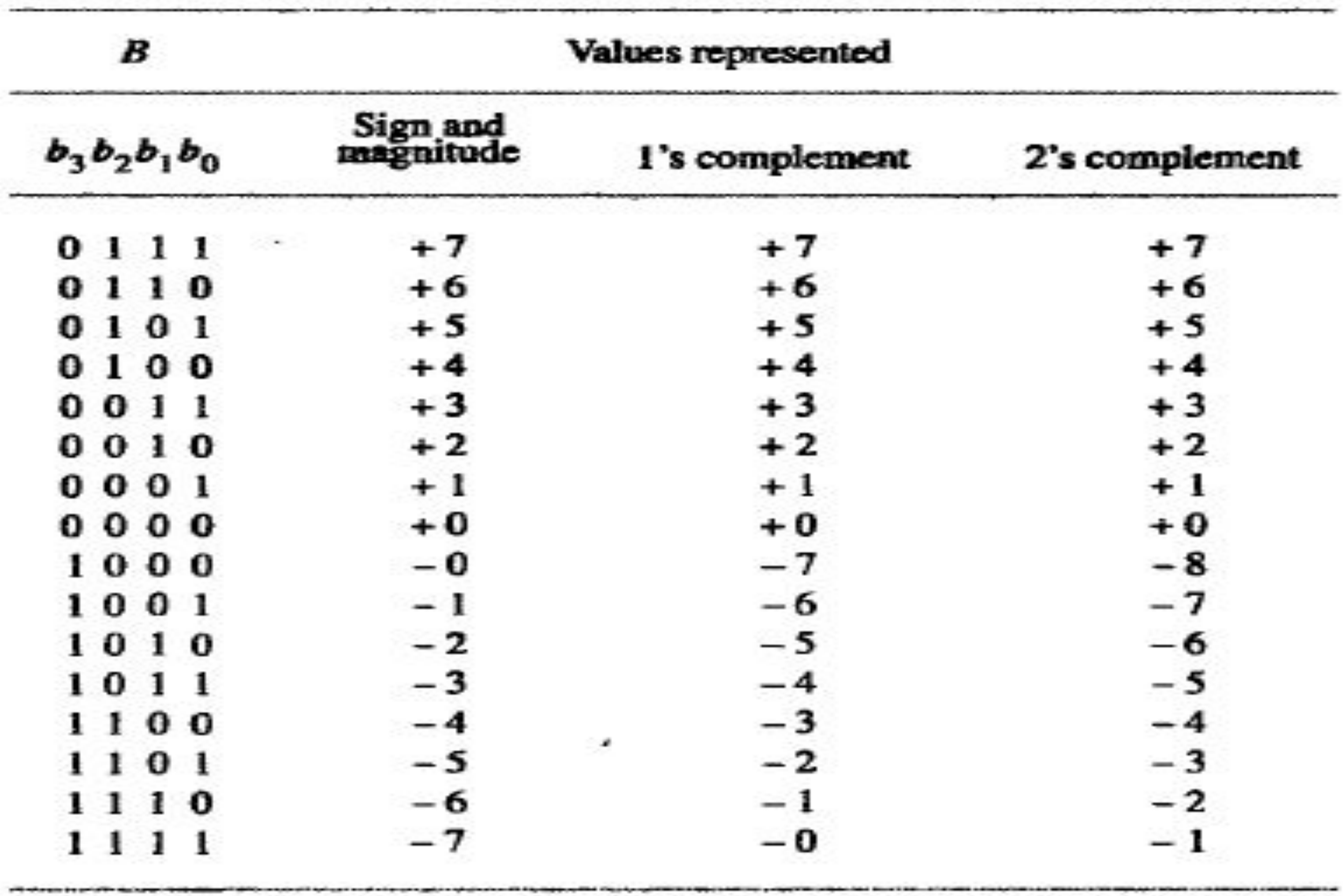
Number representation

- Various techniques to represent signed integer number are
  - Sign and magnitude
  - One’s complement
  - Two’s complement

Number, Arithmetic Operations, and Characters

- Integer number are represented in two forms
  - Signed integer
  - Unsigned integer
- Unsigned integer number represent positive numbers.
- Computer does not have provision to represent negative sign, so various techniques are used

Number, Arithmetic Operations, and Characters



B	Values represented		
$b_3 b_2 b_1 b_0$	Sign and magnitude	1's complement	2's complement
0 1 1 1	+7	+7	+7
0 1 1 0	+6	+6	+6
0 1 0 1	+5	+5	+5
0 1 0 0	+4	+4	+4
0 0 1 1	+3	+3	+3
0 0 1 0	+2	+2	+2
0 0 0 1	+1	+1	+1
0 0 0 0	+0	+0	+0
1 0 0 0	−0	−7	−8
1 0 0 1	−1	−6	−7
1 0 1 0	−2	−5	−6
1 0 1 1	−3	−4	−5
1 1 0 0	−4	−3	−4
1 1 0 1	−5	−2	−3
1 1 1 0	−6	−1	−2
1 1 1 1	−7	−0	−1

Number, Arithmetic Operations, and Characters

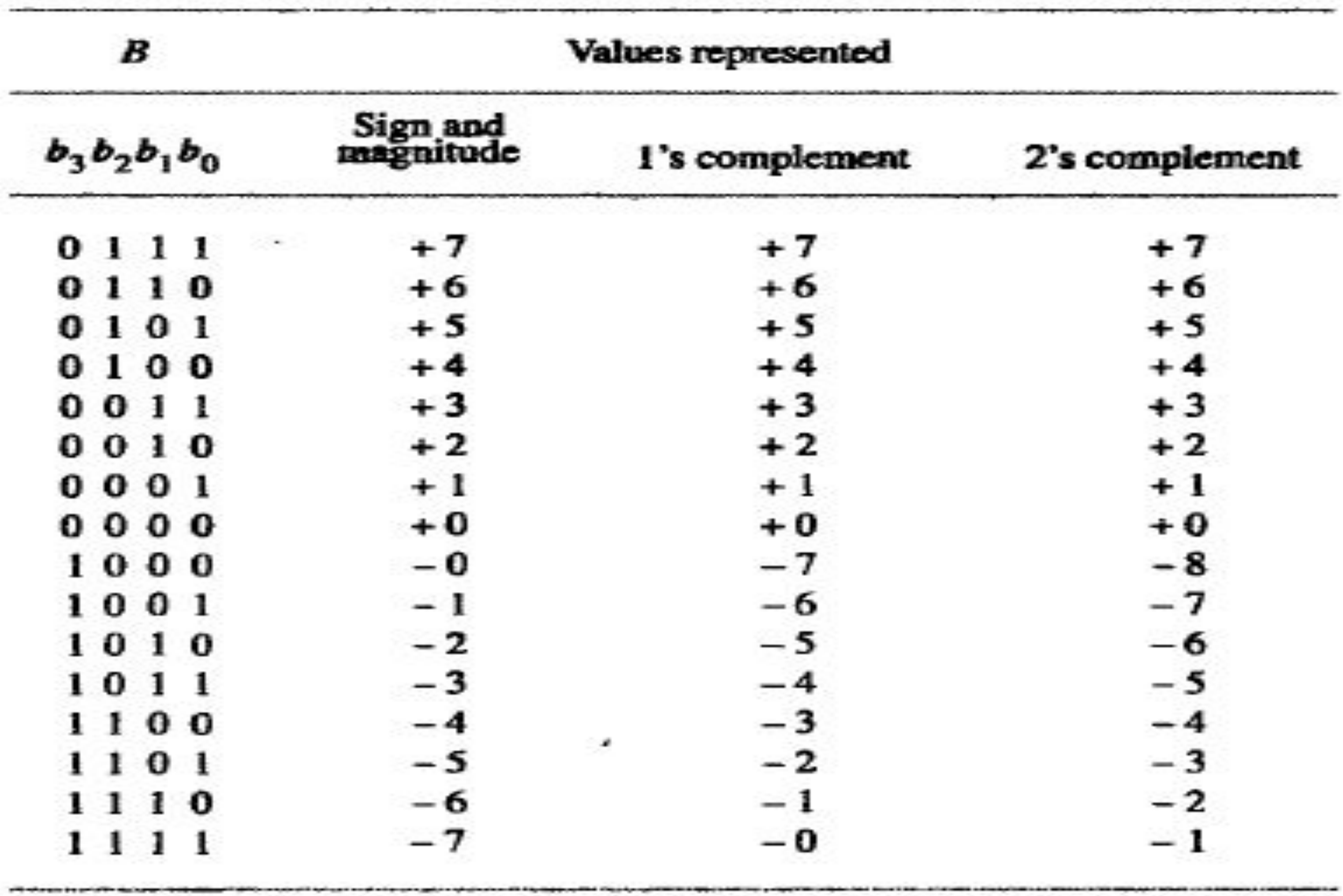
- Assumptions:

  4-bit machine word

  16 different values can be represented

  Roughly half are positive, half are negative

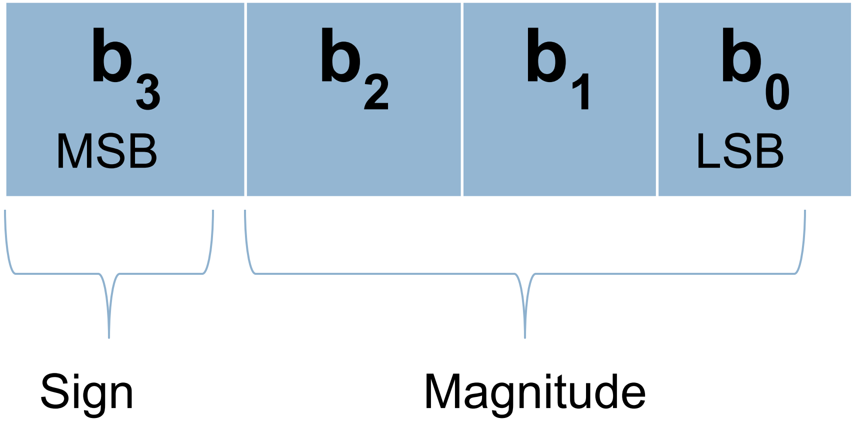
Number, Arithmetic Operations, and Characters



B	Values represented		
$b_3 b_2 b_1 b_0$	Sign and magnitude	1's complement	2's complement
0 1 1 1	+7	+7	+7
0 1 1 0	+6	+6	+6
0 1 0 1	+5	+5	+5
0 1 0 0	+4	+4	+4
0 0 1 1	+3	+3	+3
0 0 1 0	+2	+2	+2
0 0 0 1	+1	+1	+1
0 0 0 0	+0	+0	+0
1 0 0 0	−0	−7	−8
1 0 0 1	−1	−6	−7
1 0 1 0	−2	−5	−6
1 0 1 1	−3	−4	−5
1 1 0 0	−4	−3	−4
1 1 0 1	−5	−2	−3
1 1 1 0	−6	−1	−2
1 1 1 1	−7	−0	−1

Sign-and-magnitude system

- Plus(+) sign to represent positive number and Minus(-) sign to represent negative number
- Positive number and Negative number are represented with binary digits. The leftmost bit(sign bit) in the number represent sign of the number. The remaining bits represent magnitude of number
- Signed magnitude format for 4-bit signed number



b_3 MSB | b_2 | b_1 | b_0 LSB

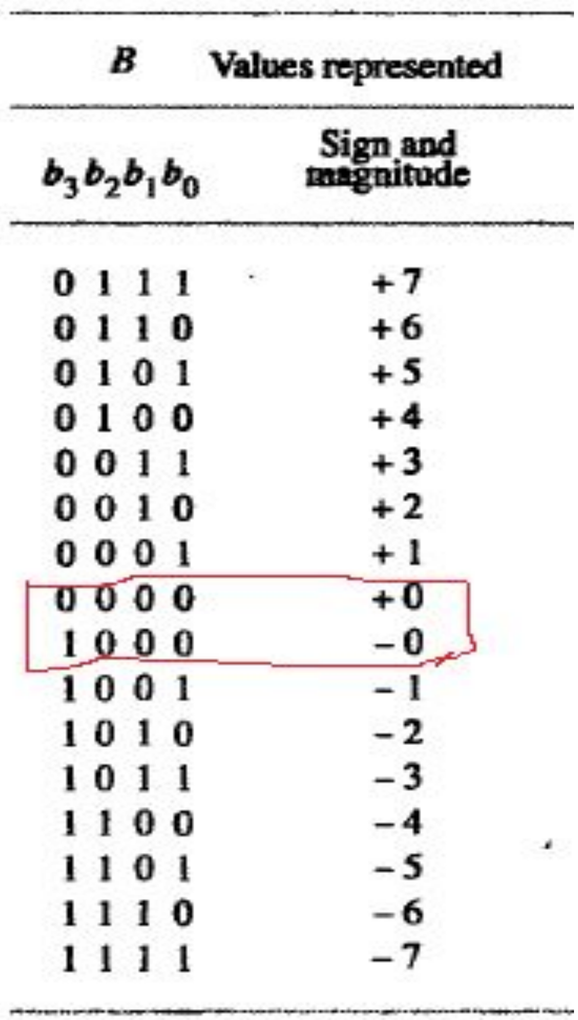
Sign | Magnitude

MSB = 0, then number is +ve

MSB = 1, then number is -ve

Sign-and-magnitude system

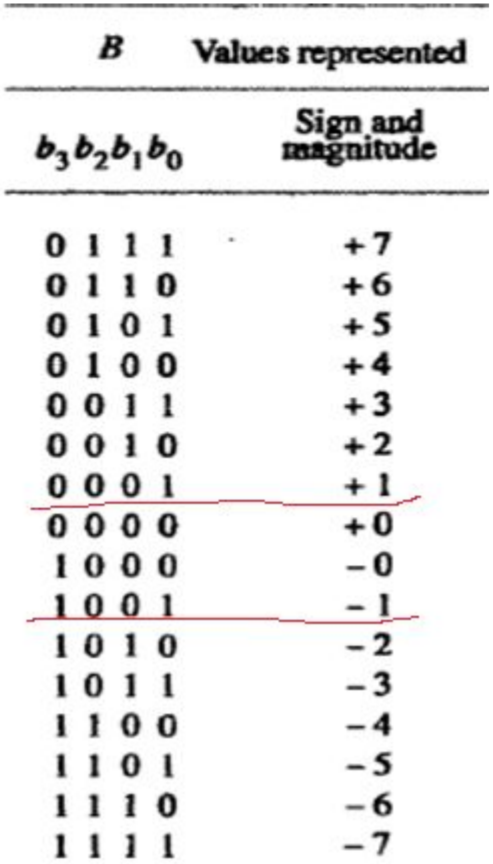
	b_3	b_2	b_1	b_0
+0	0	0	0	0
-0	**1**	**0**	**0**	**0**



B	Values represented
$b_3b_2b_1b_0$	Sign and magnitude
0 1 1 1	+7
0 1 1 0	+6
0 1 0 1	+5
0 1 0 0	+4
0 0 1 1	+3
0 0 1 0	+2
0 0 0 1	+1
0 0 0 0	+0
1 0 0 0	−0
1 0 0 1	−1
1 0 1 0	−2
1 0 1 1	−3
1 1 0 0	−4
1 1 0 1	−5
1 1 1 0	−6
1 1 1 1	−7

Sign-and-magnitude system

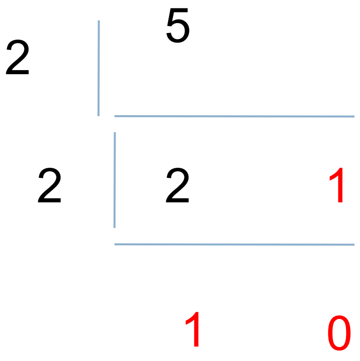
	b_3	b_2	b_1	b_0
+1	0	0	0	1
-1	**1**	**0**	**0**	**1**



B	Values represented
$b_3b_2b_1b_0$	Sign and magnitude
0 1 1 1	+7
0 1 1 0	+6
0 1 0 1	+5
0 1 0 0	+4
0 0 1 1	+3
0 0 1 0	+2
0 0 0 1	+1
0 0 0 0	+0
1 0 0 0	−0
1 0 0 1	−1
1 0 1 0	−2
1 0 1 1	−3
1 1 0 0	−4
1 1 0 1	−5
1 1 1 0	−6
1 1 1 1	−7

Sign-and-magnitude system

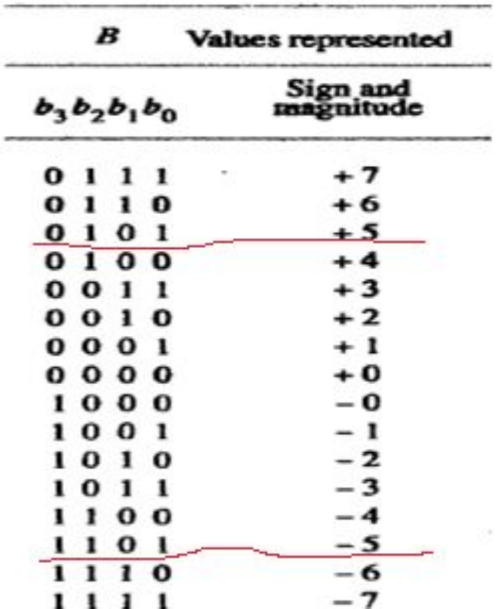
	b_3	b_2	b_1	b_0
+5	0	1	0	1
-5	**1**	**1**	**0**	**1**



2 | 5

2 | 2 1

1 0



B	Values represented
$b_3b_2b_1b_0$	Sign and magnitude
0 1 1 1	+7
0 1 1 0	+6
0 1 0 1	+5
0 1 0 0	+4
0 0 1 1	+3
0 0 1 0	+2
0 0 0 1	+1
0 0 0 0	+0
1 0 0 0	−0
1 0 0 1	−1
1 0 1 0	−2
1 0 1 1	−3
1 1 0 0	−4
1 1 0 1	−5
1 1 1 0	−6
1 1 1 1	−7

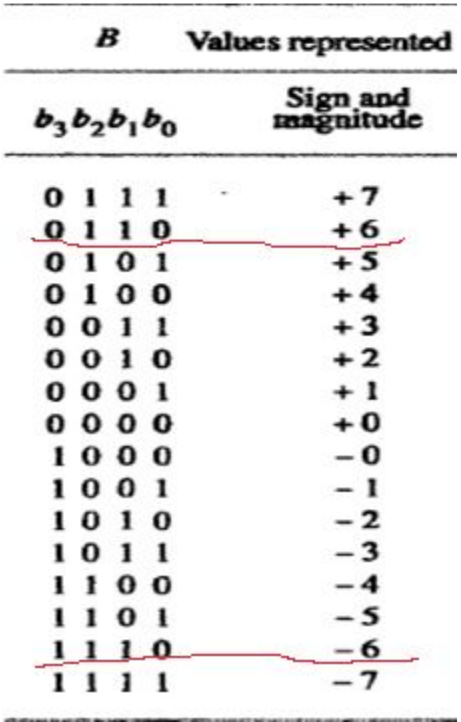
Sign-and-magnitude system

	b_3	b_2	b_1	b_0
+6	0	1	1	0
-6	1	1	1	0

2 | 6

2 | 3 0

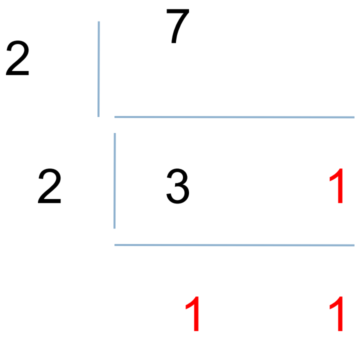
1 1



B	Values represented
$b_3b_2b_1b_0$	Sign and magnitude
0 1 1 1	+7
0 1 1 0	+6
0 1 0 1	+5
0 1 0 0	+4
0 0 1 1	+3
0 0 1 0	+2
0 0 0 1	+1
0 0 0 0	+0
1 0 0 0	−0
1 0 0 1	−1
1 0 1 0	−2
1 0 1 1	−3
1 1 0 0	−4
1 1 0 1	−5
1 1 1 0	−6
1 1 1 1	−7

Sign-and-magnitude system

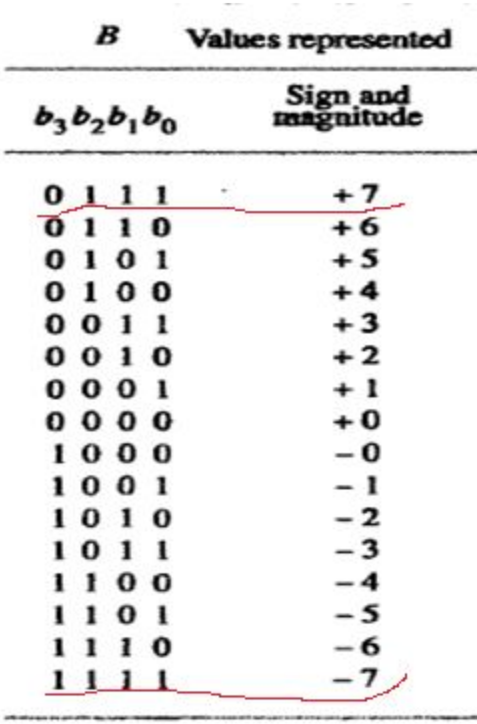
	b_3	b_2	b_1	b_0
+7	0	1	1	1
-7	1	1	1	1



2 | 7

2 | 3 1

1 1

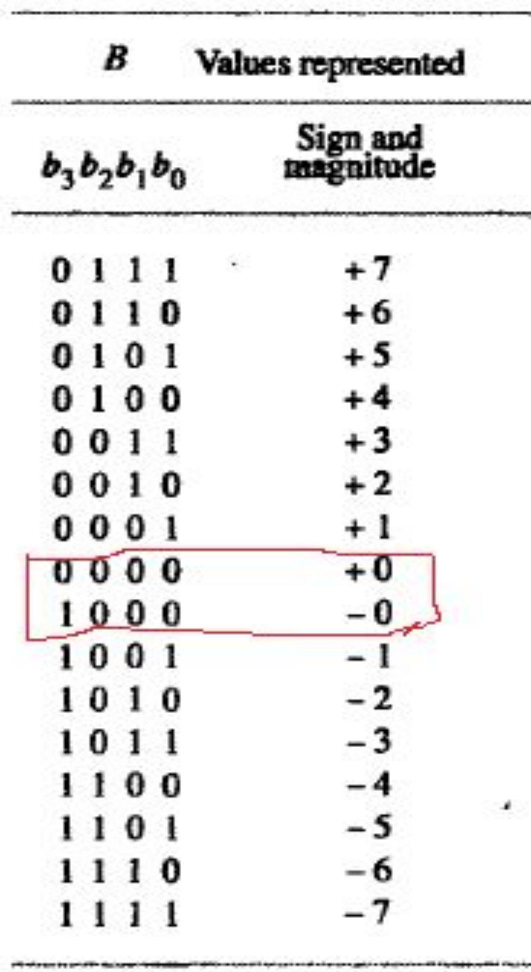


B	Values represented
$b_3b_2b_1b_0$	Sign and magnitude
0 1 1 1	+7
0 1 1 0	+6
0 1 0 1	+5
0 1 0 0	+4
0 0 1 1	+3
0 0 1 0	+2
0 0 0 1	+1
0 0 0 0	+0
1 0 0 0	−0
1 0 0 1	−1
1 0 1 0	−2
1 0 1 1	−3
1 1 0 0	−4
1 1 0 1	−5
1 1 1 0	−6
1 1 1 1	−7

Sign-and-magnitude system

- For addition and subtraction, it is necessary to consider signs of both the number and their relative magnitude in order to carry out the required arithmetic operation
- There are two representation of zero(0)
  - +0 □ 0000
  - -0 □ 1000
- Due to this, it is difficult to test for zero operation frequently

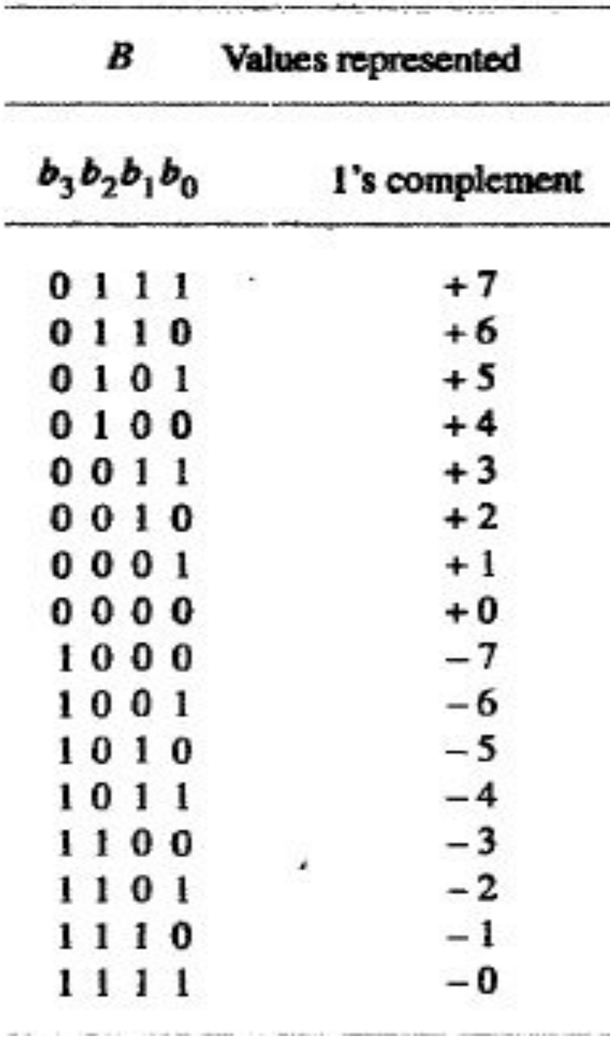
performed by computer



B	Values represented
$b_3b_2b_1b_0$	Sign and magnitude
0 1 1 1	+7
0 1 1 0	+6
0 1 0 1	+5
0 1 0 0	+4
0 0 1 1	+3
0 0 1 0	+2
0 0 0 1	+1
0 0 0 0	+0
1 0 0 0	−0
1 0 0 1	−1
1 0 1 0	−2
1 0 1 1	−3
1 1 0 0	−4
1 1 0 1	−5
1 1 1 0	−6
1 1 1 1	−7

1's complement system

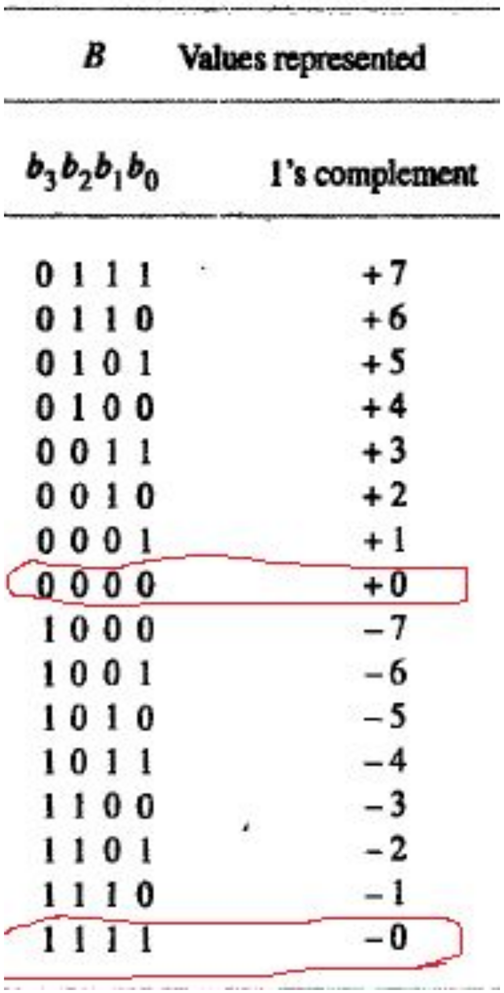
- The 1’s complement of a binary number is the number that results when we change all 1’s to zeros and the zeros to 1’s.



B	Values represented
$b_3b_2b_1b_0$	1's complement
0 1 1 1	+7
0 1 1 0	+6
0 1 0 1	+5
0 1 0 0	+4
0 0 1 1	+3
0 0 1 0	+2
0 0 0 1	+1
0 0 0 0	+0
1 0 0 0	−7
1 0 0 1	−6
1 0 1 0	−5
1 0 1 1	−4
1 1 0 0	−3
1 1 0 1	−2
1 1 1 0	−1
1 1 1 1	−0

1's complement system

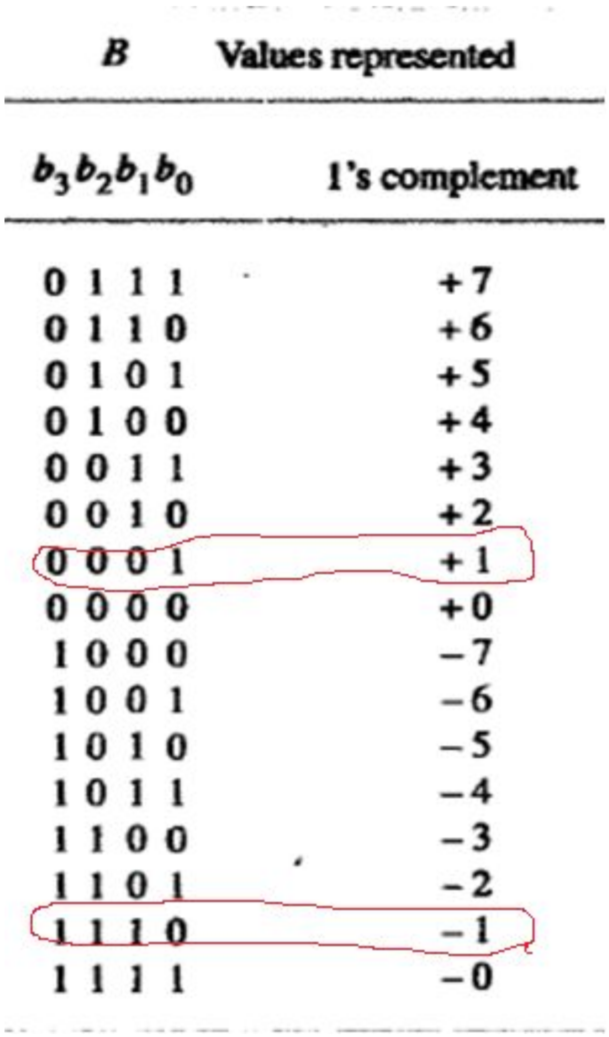
	b_3	b_2	b_1	b_0
+0	0	0	0	0
-0	**1**	**1**	**1**	**1**



B	Values represented
$b_3b_2b_1b_0$	1's complement
0 1 1 1	+7
0 1 1 0	+6
0 1 0 1	+5
0 1 0 0	+4
0 0 1 1	+3
0 0 1 0	+2
0 0 0 1	+1
0 0 0 0	+0
1 0 0 0	−7
1 0 0 1	−6
1 0 1 0	−5
1 0 1 1	−4
1 1 0 0	−3
1 1 0 1	−2
1 1 1 0	−1
1 1 1 1	−0

1's complement system

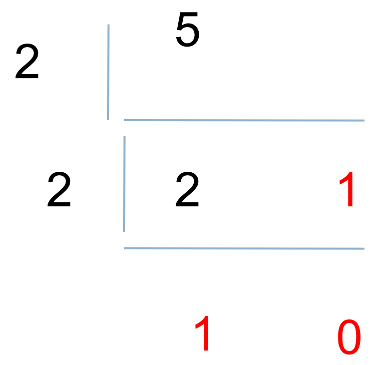
	b_3	b_2	b_1	b_0
+1	0	0	0	1
-1	1	1	1	0



B	Values represented
$b_3b_2b_1b_0$	1's complement
0 1 1 1	+7
0 1 1 0	+6
0 1 0 1	+5
0 1 0 0	+4
0 0 1 1	+3
0 0 1 0	+2
0 0 0 1	+1
0 0 0 0	+0
1 0 0 0	−7
1 0 0 1	−6
1 0 1 0	−5
1 0 1 1	−4
1 1 0 0	−3
1 1 0 1	−2
1 1 1 0	−1
1 1 1 1	−0

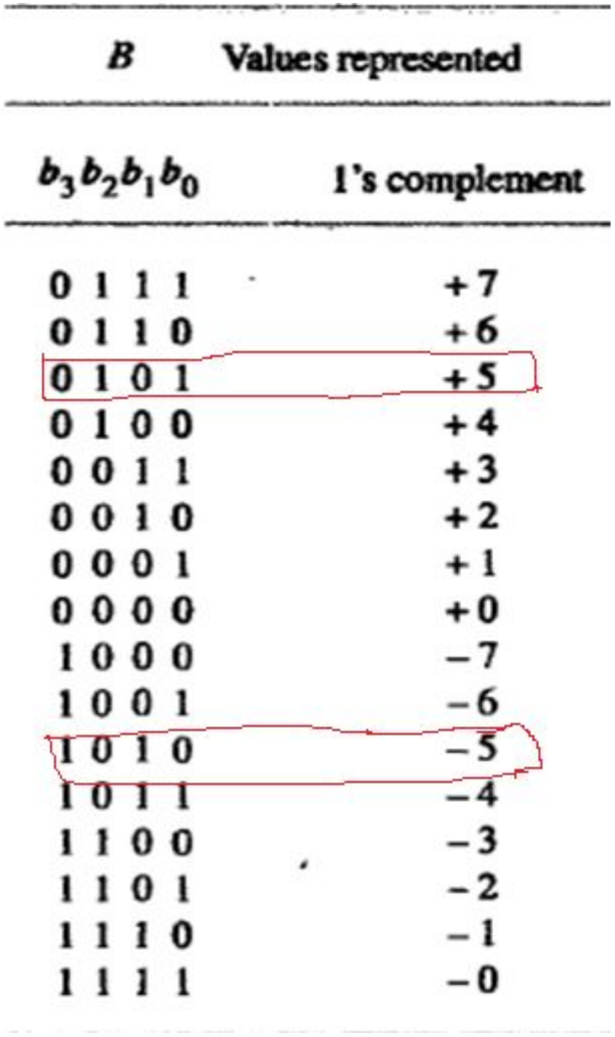
1's complement system

	b_3	b_2	b_1	b_0
+5	0	1	0	1
-5	**1**	**0**	**1**	**0**


2 | 5

2 | 2 1

1 0

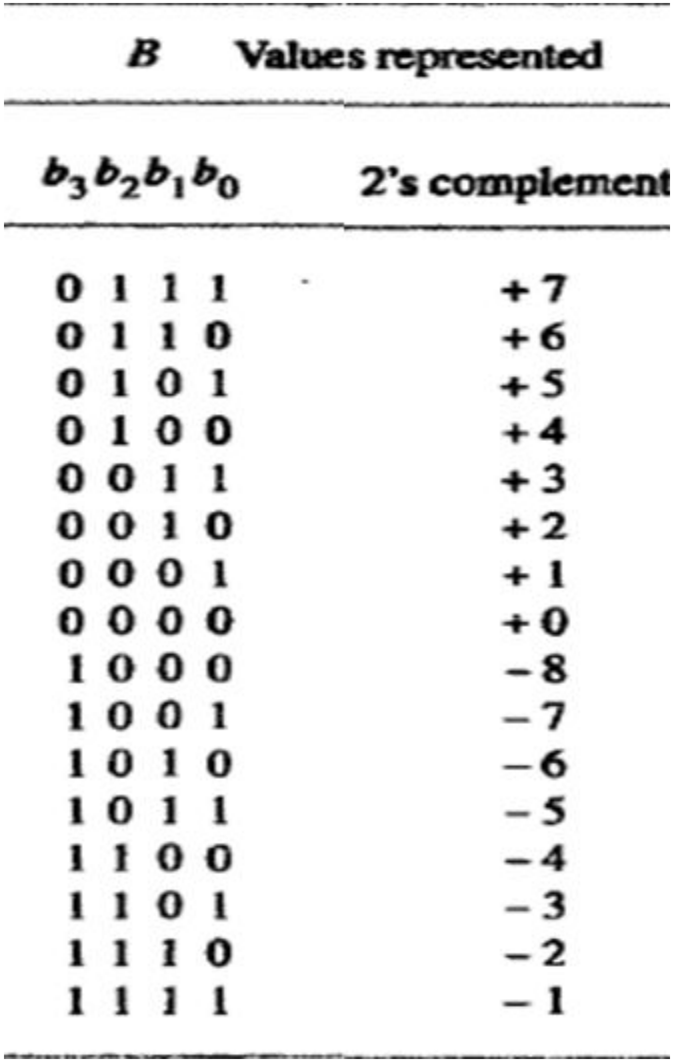


B	Values represented
$b_3b_2b_1b_0$	1's complement
0 1 1 1	+7
0 1 1 0	+6
0 1 0 1	+5
0 1 0 0	+4
0 0 1 1	+3
0 0 1 0	+2
0 0 0 1	+1
0 0 0 0	+0
1 0 0 0	−7
1 0 0 1	−6
1 0 1 0	−5
1 0 1 1	−4
1 1 0 0	−3
1 1 0 1	−2
1 1 1 0	−1
1 1 1 1	−0

2's complement system

- The 2’s complement is the binary number that results when we add 1 to the 1’s complement.

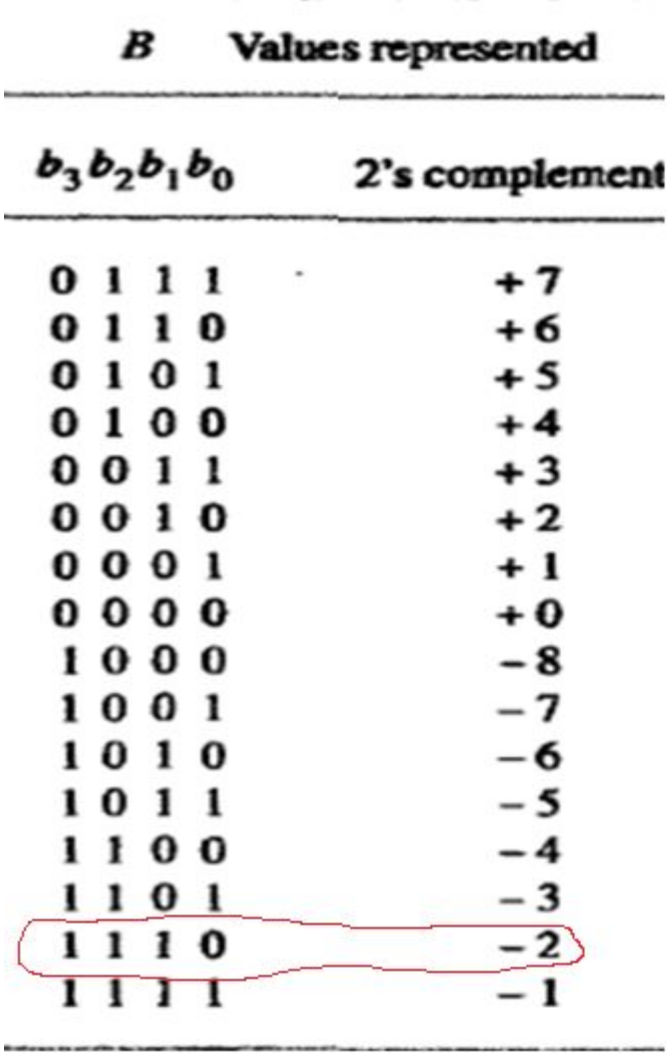
2’s complement = 1’complement +1



B	Values represented
$b_3b_2b_1b_0$	2’s complement
0 1 1 1	+7
0 1 1 0	+6
0 1 0 1	+5
0 1 0 0	+4
0 0 1 1	+3
0 0 1 0	+2
0 0 0 1	+1
0 0 0 0	+0
1 0 0 0	−8
1 0 0 1	−7
1 0 1 0	−6
1 0 1 1	−5
1 1 0 0	−4
1 1 0 1	−3
1 1 1 0	−2
1 1 1 1	−1

2's complement system

C				1	
	+2	0	0	1	0
	1's	1	1	0	1
					1
	-2	1	1	1	0

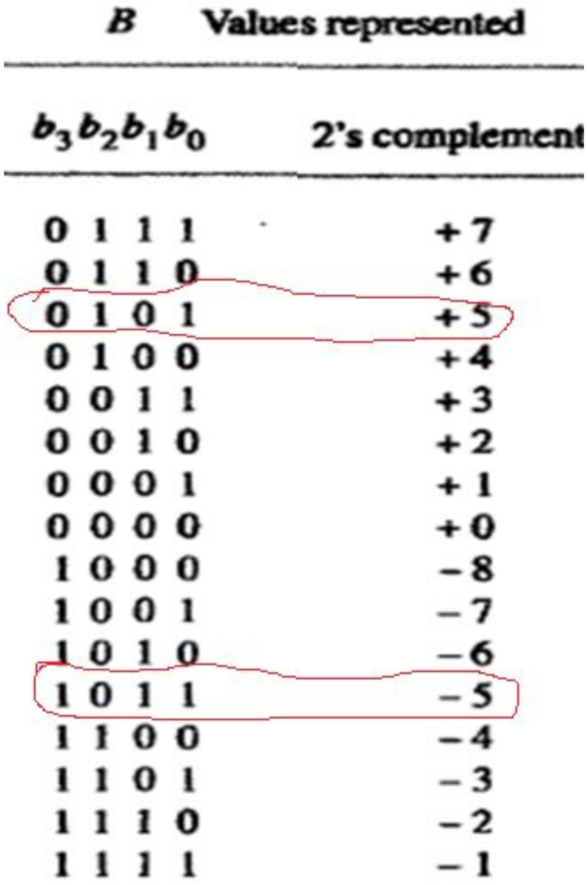


B	Values represented
$b_3b_2b_1b_0$	2's complement
0 1 1 1	+7
0 1 1 0	+6
0 1 0 1	+5
0 1 0 0	+4
0 0 1 1	+3
0 0 1 0	+2
0 0 0 1	+1
0 0 0 0	+0
1 0 0 0	−8
1 0 0 1	−7
1 0 1 0	−6
1 0 1 1	−5
1 1 0 0	−4
1 1 0 1	−3
1 1 1 0	−2
1 1 1 1	−1

2's complement system

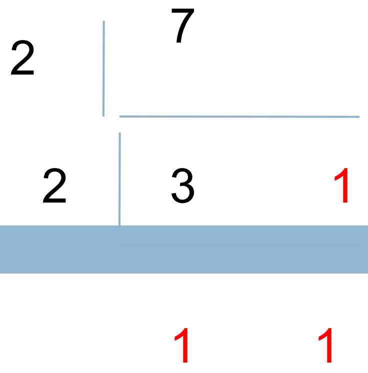
2 | 5
2 | 2 1
1 0

C					
	+5	0	1	0	1
	1's	1	0	1	0
					1
	-5	1	0	1	1

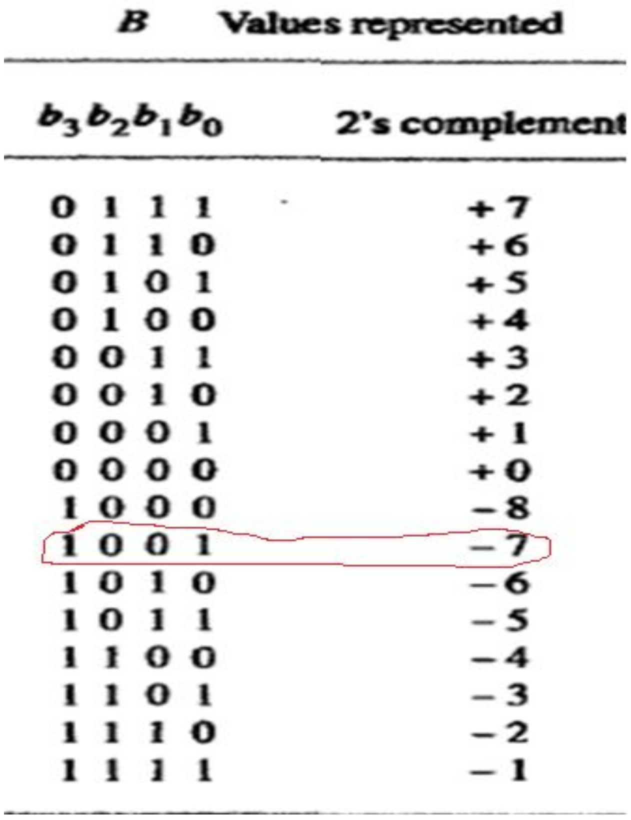


B	Values represented
$b_3b_2b_1b_0$	2's complement
0 1 1 1	+7
0 1 1 0	+6
0 1 0 1	+5
0 1 0 0	+4
0 0 1 1	+3
0 0 1 0	+2
0 0 0 1	+1
0 0 0 0	+0
1 0 0 0	−8
1 0 0 1	−7
1 0 1 0	−6
1 0 1 1	−5
1 1 0 0	−4
1 1 0 1	−3
1 1 1 0	−2
1 1 1 1	−1

2's complement system



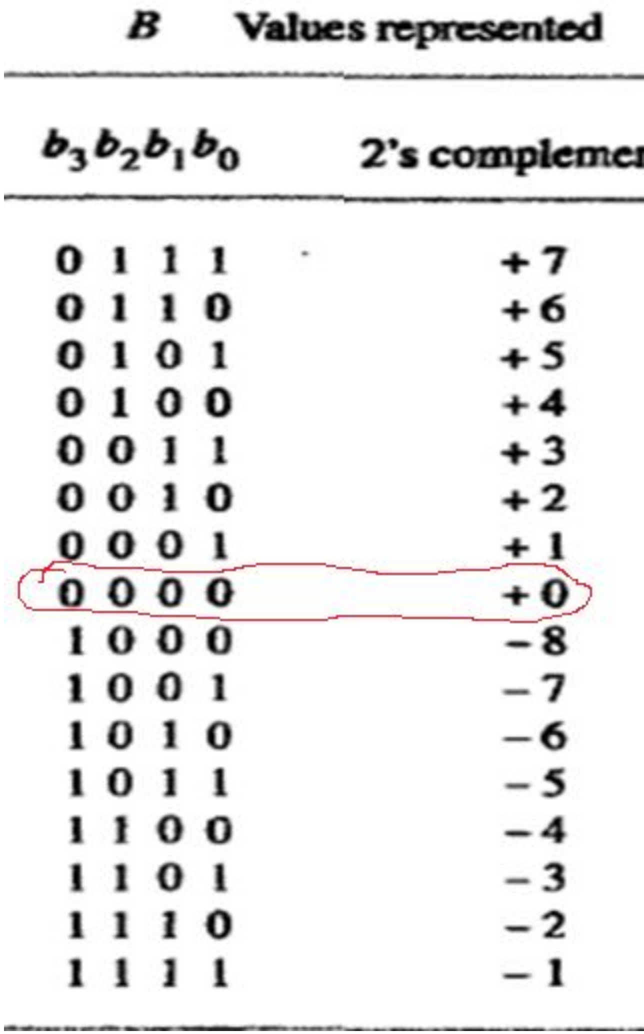
C					
	+7	0	1	1	1
	1's	1	0	0	0
					1
	-7	1	0	0	1



B	Values represented
$b_3b_2b_1b_0$	2's complement
0 1 1 1	+7
0 1 1 0	+6
0 1 0 1	+5
0 1 0 0	+4
0 0 1 1	+3
0 0 1 0	+2
0 0 0 1	+1
0 0 0 0	+0
1 0 0 0	−8
1 0 0 1	−7
1 0 1 0	−6
1 0 1 1	−5
1 1 0 0	−4
1 1 0 1	−3
1 1 1 0	−2
1 1 1 1	−1

2's complement system

C	1	1	1	1	
	+0	0	0	0	0
	1’s	1	1	1	1
					1
	-0	0	0	0	0

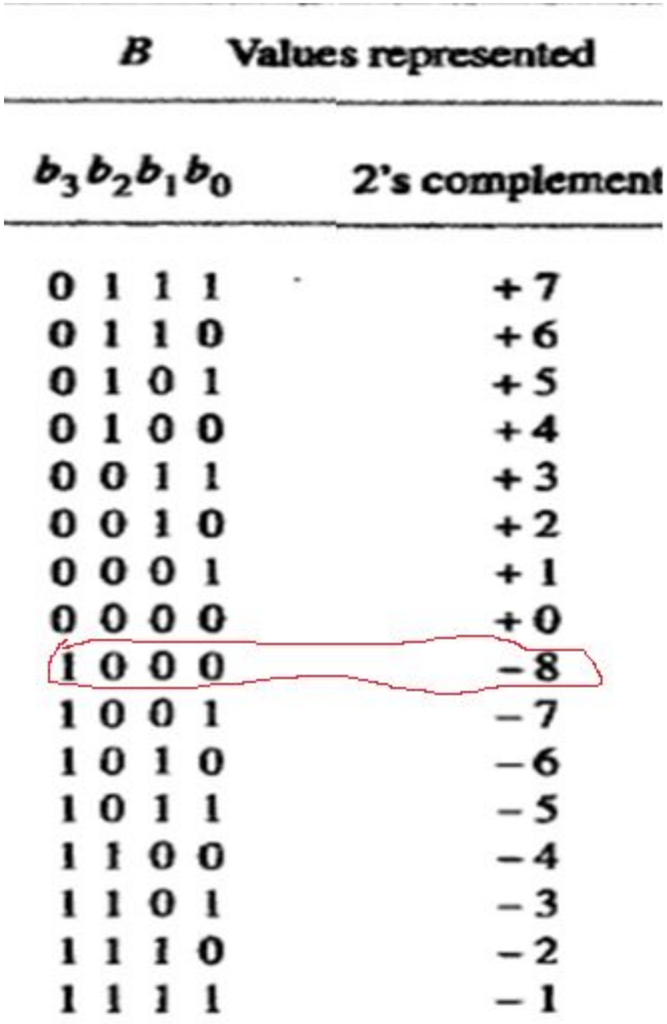


B	Values represented
$b_3b_2b_1b_0$	2’s complemen
0 1 1 1	+ 7
0 1 1 0	+ 6
0 1 0 1	+ 5
0 1 0 0	+ 4
0 0 1 1	+ 3
0 0 1 0	+ 2
0 0 0 1	+ 1
0 0 0 0	+ 0
1 0 0 0	− 8
1 0 0 1	− 7
1 0 1 0	− 6
1 0 1 1	− 5
1 1 0 0	− 4
1 1 0 1	− 3
1 1 1 0	− 2
1 1 1 1	− 1

2's complement system

Advantages

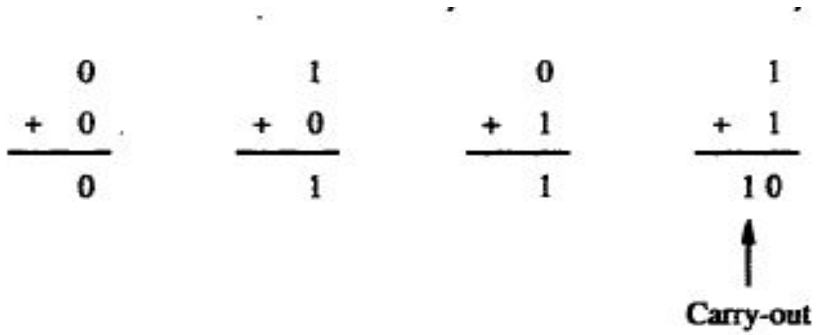
- □ Here One representation for +0 and -0
- □ Here -8 is representable
- □ Most efficient way to carry out addition and subtraction operations
- □ Most often used in computers



B	Values represented
$b_3 b_2 b_1 b_0$	2's complement
0 1 1 1	+7
0 1 1 0	+6
0 1 0 1	+5
0 1 0 0	+4
0 0 1 1	+3
0 0 1 0	+2
0 0 0 1	+1
0 0 0 0	+0
1 0 0 0	−8
1 0 0 1	−7
1 0 1 0	−6
1 0 1 1	−5
1 1 0 0	−4
1 1 0 1	−3
1 1 1 0	−2
1 1 1 1	−1

ADDITION OF POSITIVE NUMBERS

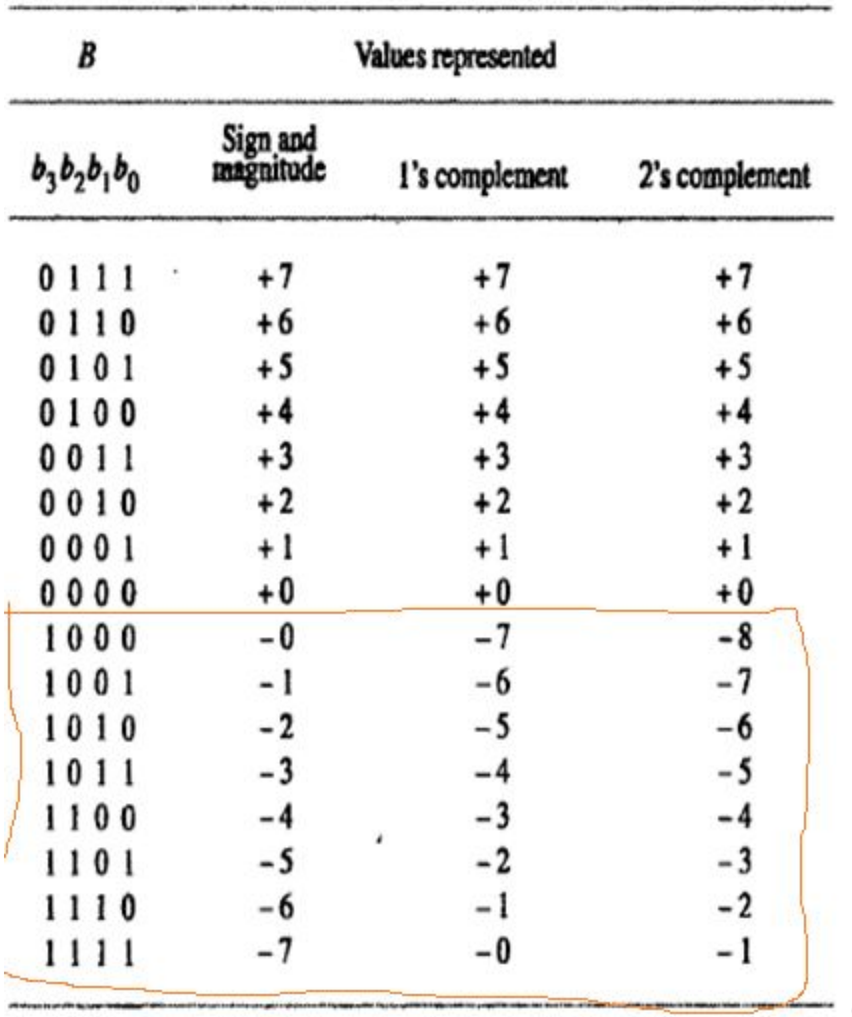
- Consider adding two 1-bit numbers.
- The sum of 1 & 1 requires the 2-bit vector 10 to represent the value 2.
- We say that sum is 0 and the carry-out is 1.



Carry out	1	
		1
		1
sum		0

Number, Arithmetic Operations, and Characters

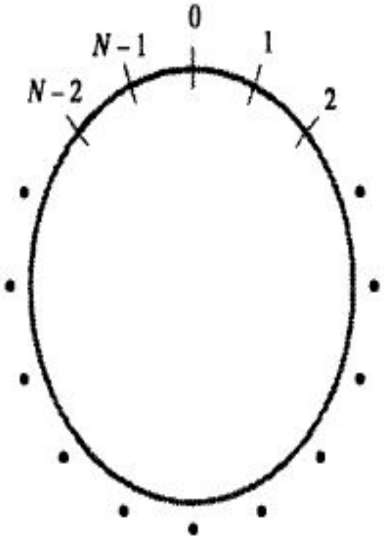
- Various techniques
  - Sign and magnitude
  - One’s complement
  - Two’s complement
- They differ in the way we represent negative numbers



B	Values represented		
$b_3 b_2 b_1 b_0$	Sign and magnitude	1's complement	2's complement
0 1 1 1	+7	+7	+7
0 1 1 0	+6	+6	+6
0 1 0 1	+5	+5	+5
0 1 0 0	+4	+4	+4
0 0 1 1	+3	+3	+3
0 0 1 0	+2	+2	+2
0 0 0 1	+1	+1	+1
0 0 0 0	+0	+0	+0
1 0 0 0	-0	-7	-8
1 0 0 1	-1	-6	-7
1 0 1 0	-2	-5	-6
1 0 1 1	-3	-4	-5
1 1 0 0	-4	-3	-4
1 1 0 1	-5	-2	-3
1 1 1 0	-6	-1	-2
1 1 1 1	-7	-0	-1

To understand 2's complement arithmetic

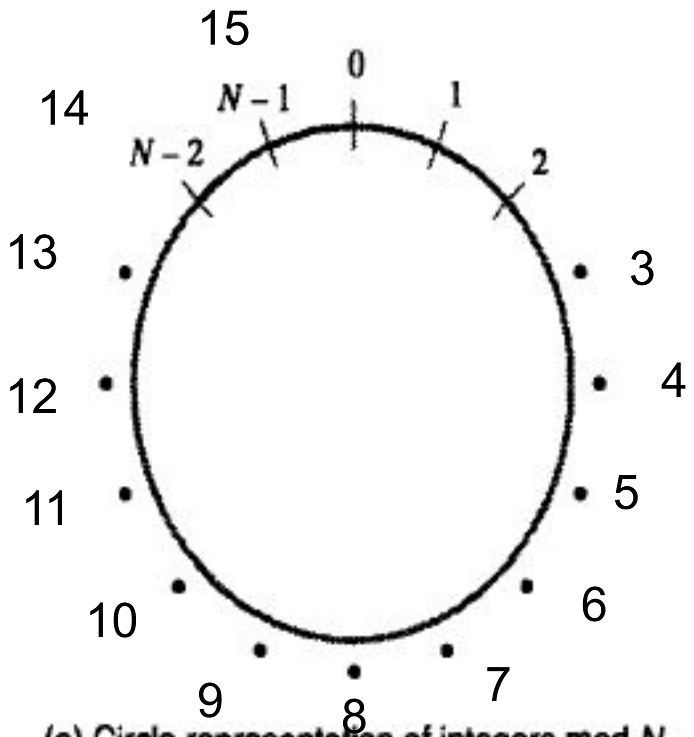
- Consider addition
  modulo N(mod N)
- A Graphical device of addition mod N of positive integers is a circle with N values (0 to N-1)along its perimeter
- Use this device to compute
  (a+b) mod N



(a) Circle representation of integers mod *N*

To understand 2’s complement arithmetic

- Consider the case N=16



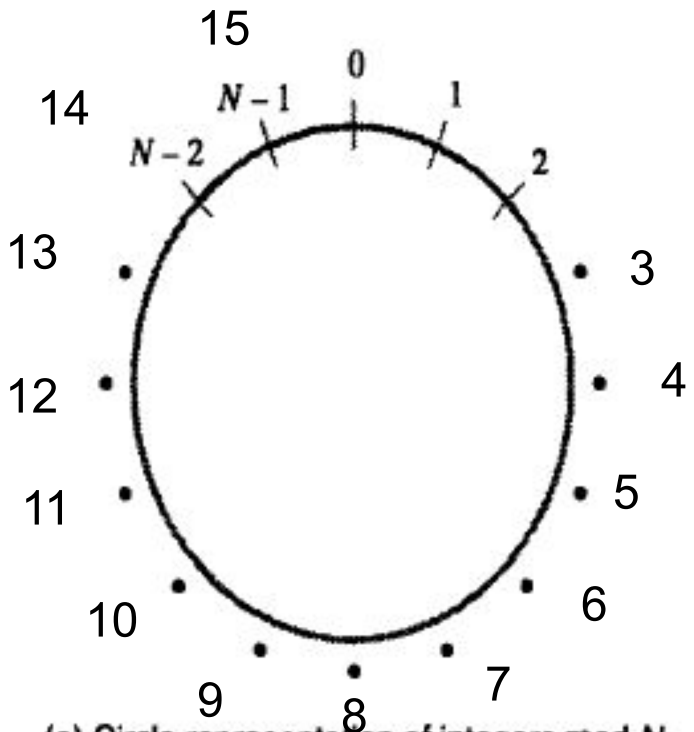
(a) Circle representation of integers mod N

(a + b) mod N
(7 +4) mod 16
= 11

Locate 7 on the circle and then move 4 units in the clockwise direction to arrive at the answer 11

To understand 2's complement arithmetic

- Consider the case N=16



(a) Circle representation of integers mod *N*

(a + b) mod N

(9 +14) mod 16

23 mod 16

= 7

Locate 9 on the circle and then move 14 units in the clockwise direction to arrive at the answer 7

ADDITION & SUBTRACTION OF SIGNED NUMBERS

Two rules for addition and subtraction of n-bit signed numbers using the 2's complement representation system

Rule 1:

To Add two numbers, add their n-bits and ignore the carry-out signal from the MSB position.

Result will be algebraically correct, if it lies in the range -2^{n-1} to $+2^{n-1}-1$.

Rule 2:

To Subtract two numbers X and Y (that is to perform X-Y), take the 2's complement of Y and then add it to X as in rule 1.

Result will be algebraically correct, if it lies in the range -2^{n-1} to $+2^{n-1}-1$.

ADDITION OF SIGNED NUMBERS

Rule 1:

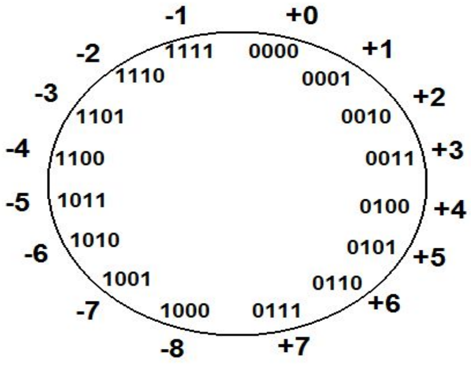
To Add two numbers, add their n-bits and ignore the carry-out signal from the MSB position.

Result will be algebraically correct, if it lies in the range -2^{n-1} to $+2^{n-1}-1$.

n = 4 bits □ $-\ 2^{4-1}$ to $+\ 2^{4-1} - 1$

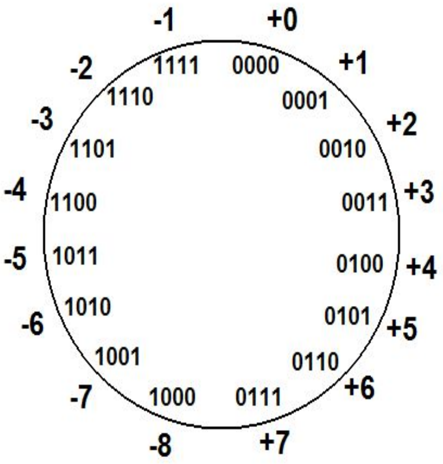
$-\,2^{3}$ to $+\ 2^{3} - 1$

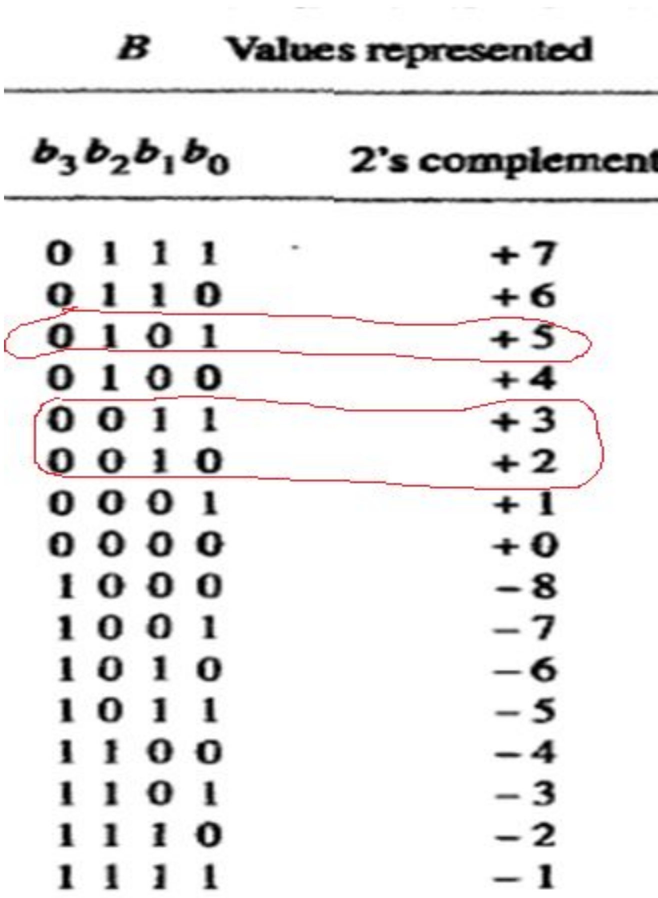
-8 to +7



Addition rule
2's complement system

C			1		
	+2	0	0	1	0
	+3	**0**	**0**	**1**	**1**
	+5	**0**	**1**	**0**	**1**

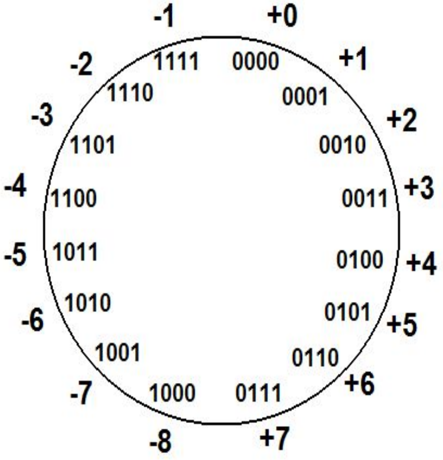


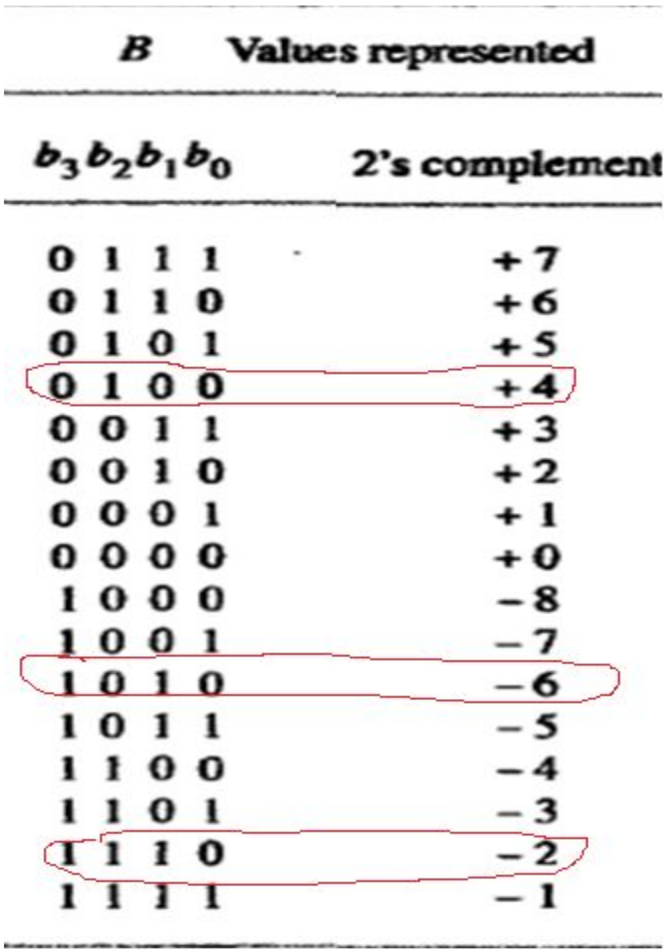


B	**Values represented**
$b_3b_2b_1b_0$	2's complement
0 1 1 1	+7
0 1 1 0	+6
0 1 0 1	+5
0 1 0 0	+4
0 0 1 1	+3
0 0 1 0	+2
0 0 0 1	+1
0 0 0 0	+0
1 0 0 0	−8
1 0 0 1	−7
1 0 1 0	−6
1 0 1 1	−5
1 1 0 0	−4
1 1 0 1	−3
1 1 1 0	−2
1 1 1 1	−1

Addition rule
2's complement system

C					
	+4	0	1	0	0
	-6	1	0	1	0
	-2	1	1	1	0

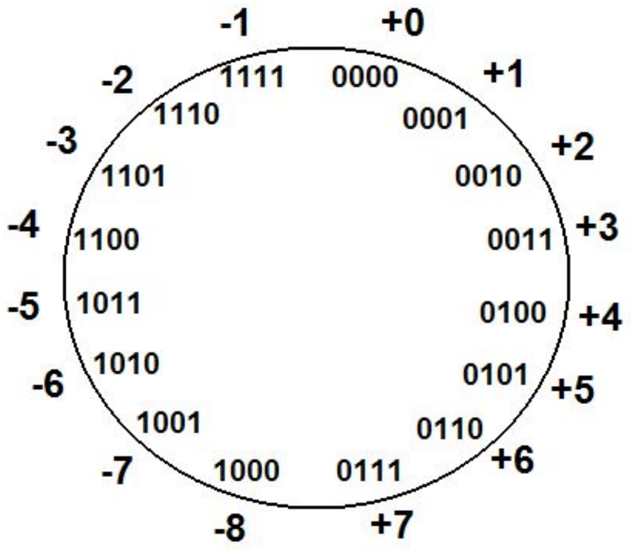




B	Values represented
$b_3 b_2 b_1 b_0$	2's complement
0 1 1 1	+ 7
0 1 1 0	+ 6
0 1 0 1	+ 5
0 1 0 0	+ 4
0 0 1 1	+ 3
0 0 1 0	+ 2
0 0 0 1	+ 1
0 0 0 0	+ 0
1 0 0 0	− 8
1 0 0 1	− 7
1 0 1 0	− 6
1 0 1 1	− 5
1 1 0 0	− 4
1 1 0 1	− 3
1 1 1 0	− 2
1 1 1 1	− 1

Addition rule
2's complement system

C	1	1	1		
	-5	1	0	1	1
	-2	1	1	1	0
s	-7	1	0	0	1



SUBTRACTION OF SIGNED NUMBERS

Rule 2:

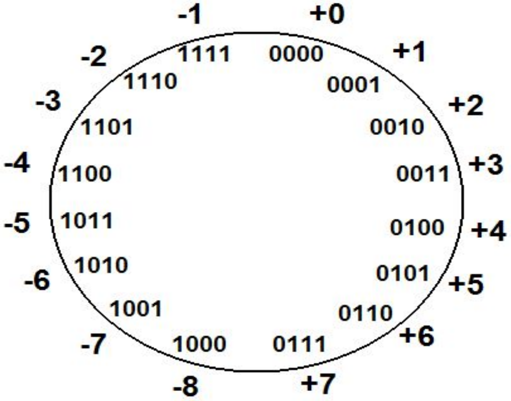
To Subtract two numbers X and Y (that is to perform X-Y), take the 2's complement of Y and then add it to X as in rule 1.

Result will be algebraically correct, if it lies in the range -2^{n-1} to $+2^{n-1}-1$.

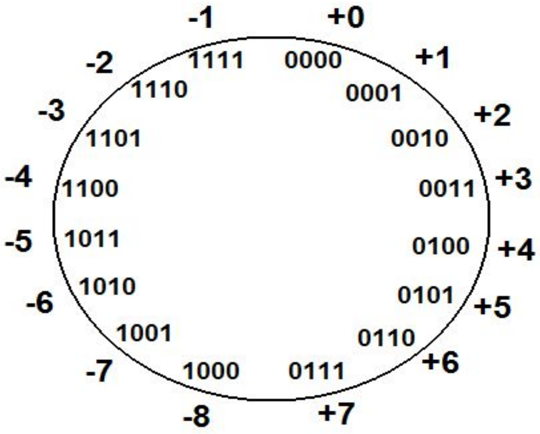
n = 4 bits □ -2^{4-1} to $+2^{4-1}-1$

$^{-}2^{3}$ to $+2^{3}-1$

-8 to +7



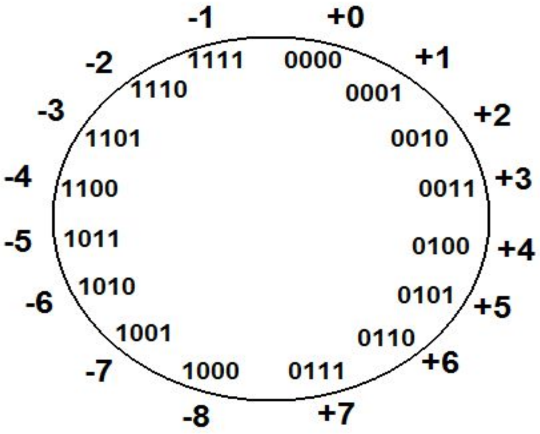
Subtraction rule
2's complement system



	1	1	1	1	
X	-3	1	1	0	1
Y	**-7**	**0**	**1**	**1**	**1**
X-Y	**+4**	**0**	**1**	**0**	**0**

C					
Y	-7	1	0	0	1
	1’s	**0**	**1**	**1**	**0**
					1
		0	**1**	**1**	**1**

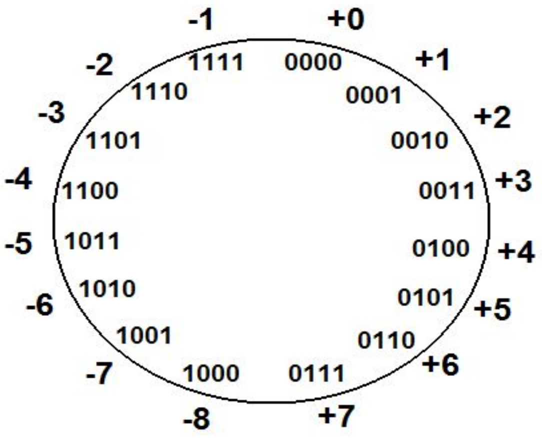
Subtraction rule
2's complement system



	1	1			
X	+6	0	1	1	0
Y	**+3**	**1**	**1**	**0**	**1**
X-Y	**+3**	**0**	**0**	**1**	**1**

C					
Y	+3	0	0	1	1
	1’s	**1**	**1**	**0**	**0**
					1
		1	**1**	**0**	**1**

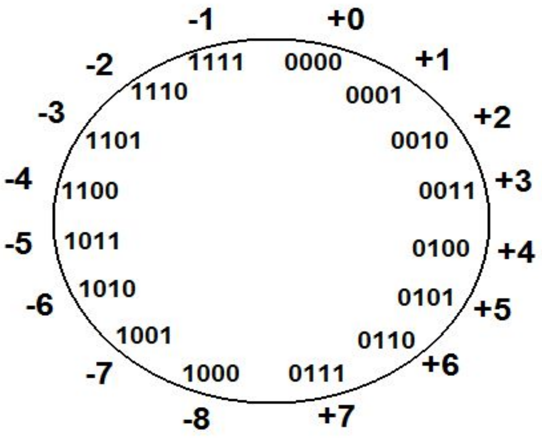
Subtraction rule
2's complement system



	1	1	1	1	
X	-7	1	0	0	1
Y	+1	1	1	1	1
X-Y	-8	1	0	0	0

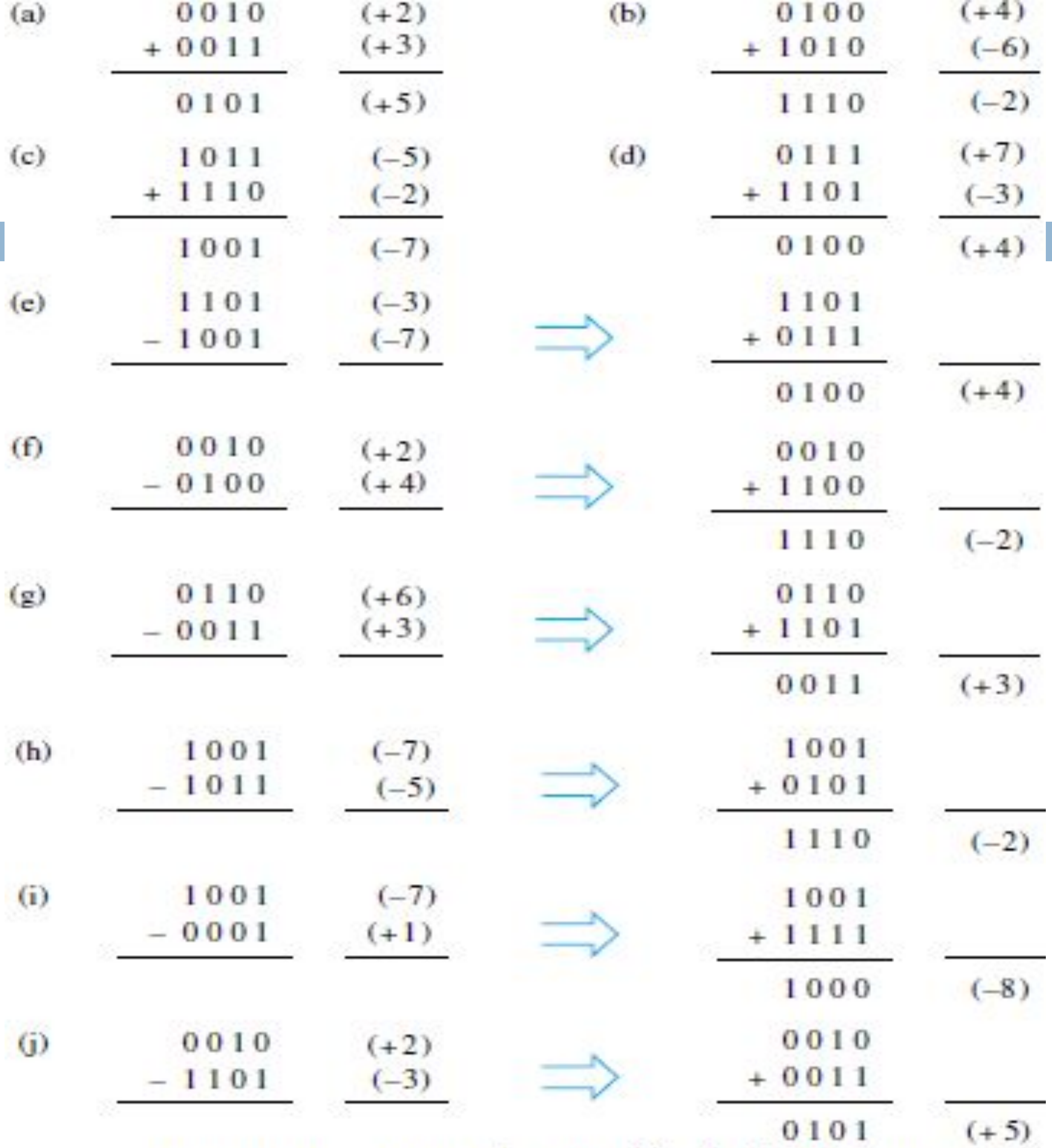
C					
Y	+1	0	0	0	1
	1’s	1	1	1	0
					1
		1	1	1	1

Subtraction rule
2's complement system



			1		
X	+2	0	0	1	0
Y	**-3**	0	**0**	**1**	**1**
X-Y	**+5**	**0**	**1**	0	1

C					
Y		1	1	0	1
	1's	**0**	**0**	**1**	**0**
					1
		0	**0**	**1**	**1**



(a) $$\begin{array}{r} 0010 \\ +\;0011 \\ \hline 0101 \end{array} \qquad \begin{array}{r} (+2) \\ (+3) \\ \hline (+5) \end{array}$$

(b) $$\begin{array}{r} 0100 \\ +\;1010 \\ \hline 1110 \end{array} \qquad \begin{array}{r} (+4) \\ (-6) \\ \hline (-2) \end{array}$$

(c) $$\begin{array}{r} 1011 \\ +\;1110 \\ \hline 1001 \end{array} \qquad \begin{array}{r} (-5) \\ (-2) \\ \hline (-7) \end{array}$$

(d) $$\begin{array}{r} 0111 \\ +\;1101 \\ \hline 0100 \end{array} \qquad \begin{array}{r} (+7) \\ (-3) \\ \hline (+4) \end{array}$$

(e) $$\begin{array}{r} 1101 \\ -\;1001 \\ \hline \end{array} \qquad \begin{array}{r} (-3) \\ (-7) \\ \hline \end{array} \qquad \Longrightarrow \qquad \begin{array}{r} 1101 \\ +\;0111 \\ \hline 0100 \end{array} \qquad \begin{array}{r} \\ \\ \hline (+4) \end{array}$$

(f) $$\begin{array}{r} 0010 \\ -\;0100 \\ \hline \end{array} \qquad \begin{array}{r} (+2) \\ (+4) \\ \hline \end{array} \qquad \Longrightarrow \qquad \begin{array}{r} 0010 \\ +\;1100 \\ \hline 1110 \end{array} \qquad \begin{array}{r} \\ \\ \hline (-2) \end{array}$$

(g) $$\begin{array}{r} 0110 \\ -\;0011 \\ \hline \end{array} \qquad \begin{array}{r} (+6) \\ (+3) \\ \hline \end{array} \qquad \Longrightarrow \qquad \begin{array}{r} 0110 \\ +\;1101 \\ \hline 0011 \end{array} \qquad \begin{array}{r} \\ \\ \hline (+3) \end{array}$$

(h) $$\begin{array}{r} 1001 \\ -\;1011 \\ \hline \end{array} \qquad \begin{array}{r} (-7) \\ (-5) \\ \hline \end{array} \qquad \Longrightarrow \qquad \begin{array}{r} 1001 \\ +\;0101 \\ \hline 1110 \end{array} \qquad \begin{array}{r} \\ \\ \hline (-2) \end{array}$$

(i) $$\begin{array}{r} 1001 \\ -\;0001 \\ \hline \end{array} \qquad \begin{array}{r} (-7) \\ (+1) \\ \hline \end{array} \qquad \Longrightarrow \qquad \begin{array}{r} 1001 \\ +\;1111 \\ \hline 1000 \end{array} \qquad \begin{array}{r} \\ \\ \hline (-8) \end{array}$$

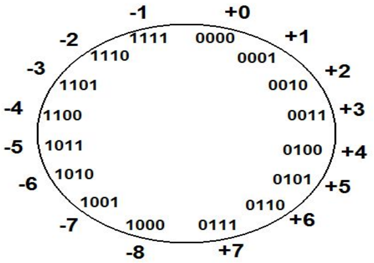
(j) $$\begin{array}{r} 0010 \\ -\;1101 \\ \hline \end{array} \qquad \begin{array}{r} (+2) \\ (-3) \\ \hline \end{array} \qquad \Longrightarrow \qquad \begin{array}{r} 0010 \\ +\;0011 \\ \hline 0101 \end{array} \qquad \begin{array}{r} \\ \\ \hline (+5) \end{array}$$

Figure 1.6 2's-complement Add and Subtract operations.

OVERFLOW IN INTEGER ARITHMETIC

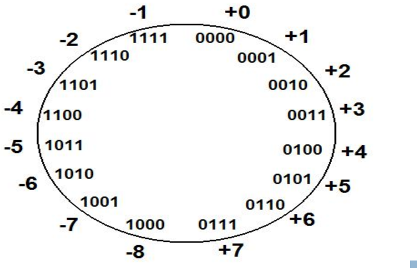
- Example of addition and subtraction in 4-bit example, answers fall in the range of -8 and +7.
- When answers do not fall within this range, we say that overflow has occurred

OVERFLOW IN INTEGER ARITHMETIC



- When result of an arithmetic operation is outside the representable range, an **arithmetic overflow** is said to occur.
- Examine the signs of the two summands X and Y and sign of the result. When both operands X and Y have the same sign, an overflow occurs when the sign of S is not the same as sign of X and Y.

OVERFLOW IN INTEGER ARITHMETIC

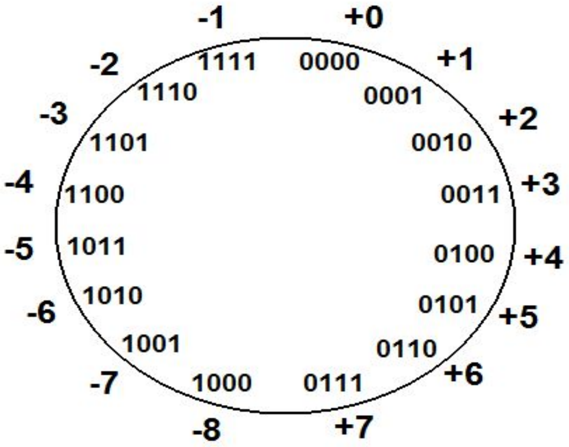


- **The rules for detecting overflow in a two's complement sum are simple:**
  1. If the sum of two positive numbers yields a negative result, the sum has overflowed.
  2. If the sum of two negative numbers yields a positive result, the sum has overflowed.
  3. Otherwise, the sum has not overflowed.

OVERFLOW IN INTEGER ARITHMETIC

- Perform following operation on the 4-bit signed numbers using 2's complement representation system. Also indicate whether overflow has occurred. (+7) + (+4)

C	0	1			
X	+7	0	1	1	1
Y	+4	0	1	0	0
S	-5	1	0	1	1

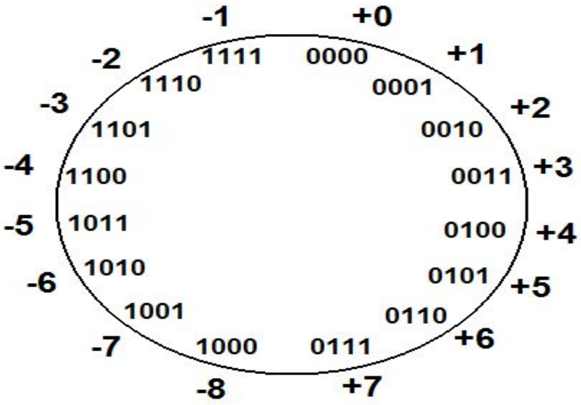


Overflow has occurred

OVERFLOW IN INTEGER ARITHMETIC

- Perform following operation on the 4-bit signed numbers using 2's complement representation system. Also indicate whether overflow has occurred. (-5) + (-6)

C	1		1		
X	-5	1	0	1	1
Y	-6	1	0	1	0
S	+5	0	1	0	1

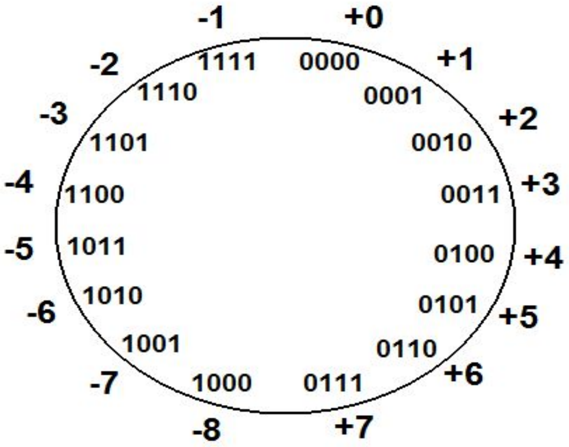


Overflow has occurred

OVERFLOW IN INTEGER ARITHMETIC

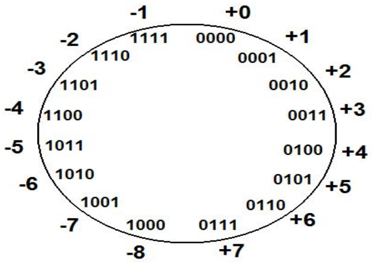
- Perform following operation on the 4-bit signed numbers using 2's complement representation system. Also indicate whether overflow has occurred. (+7) + (-4)

C	1	1			
X	+7	0	1	1	1
Y	-4	1	1	0	0
S	+3	0	0	1	1

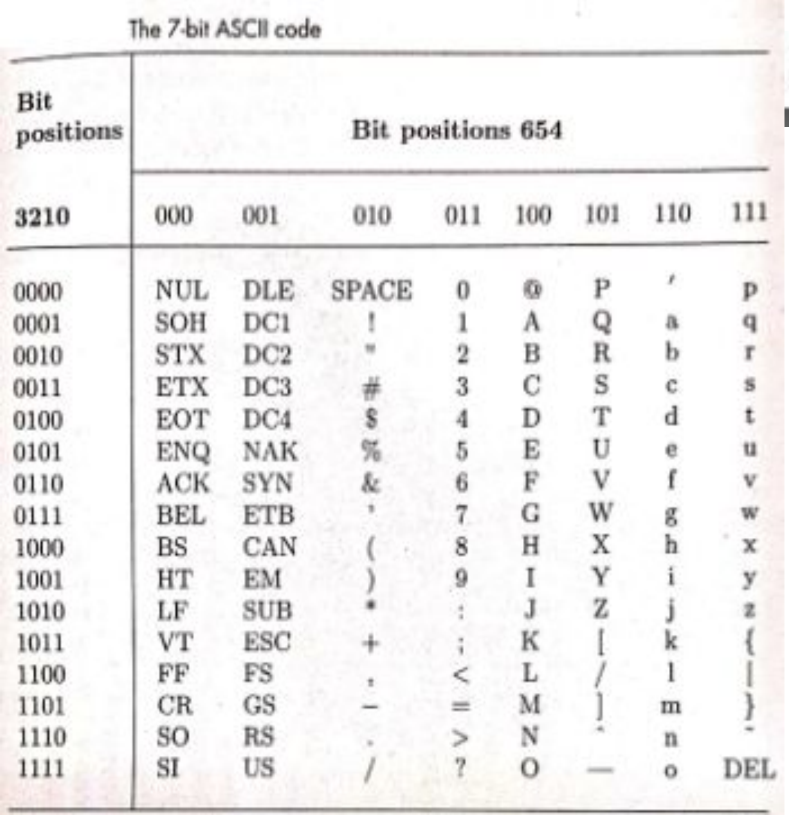


No Overflow

CHARACTERS

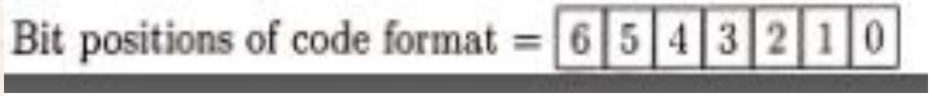


- Character can be letters of the alphabet , decimal digits, punctuation marks and so on.
- Are represented by codes(ASCII -7bits and EBDIC – 8bits)



The 7-bit ASCII code

Bit positions	Bit positions 654							
3210	000	001	010	011	100	101	110	111
0000	NUL	DLE	SPACE	0	@	P	'	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	/	l	|
1101	CR	GS	–	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	—	o	DEL



Bit positions of code format = 6 5 4 3 2 1 0

ASCII American standard committee on Information Interchange

EBDIC Extended Binary coded Decimal Interchange code

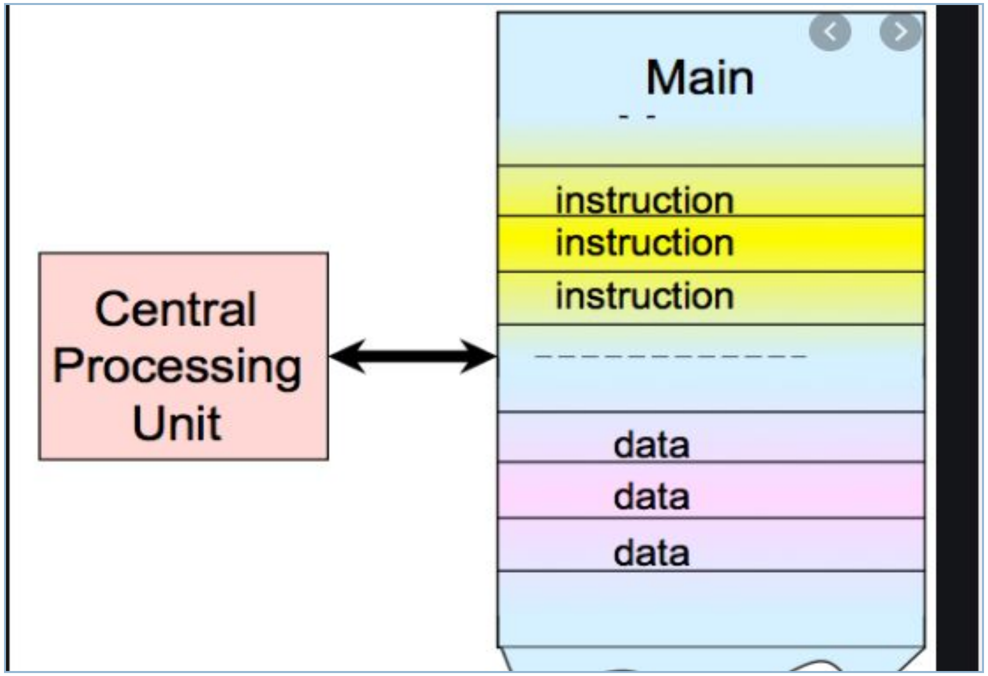
Unit -1

- Functional units, Bus structures, performance,
- Overflow in integer arithmetic: Numbers, Arithmetic operations and characters
- Memory locations and addresses,
- Memory operations,
- Instructions and instruction sequencing,
- Addressing modes,
- Subroutines and use of stack frames,
- Encoding of machine instructions.

Memory Location and Addresses

- What is Memory

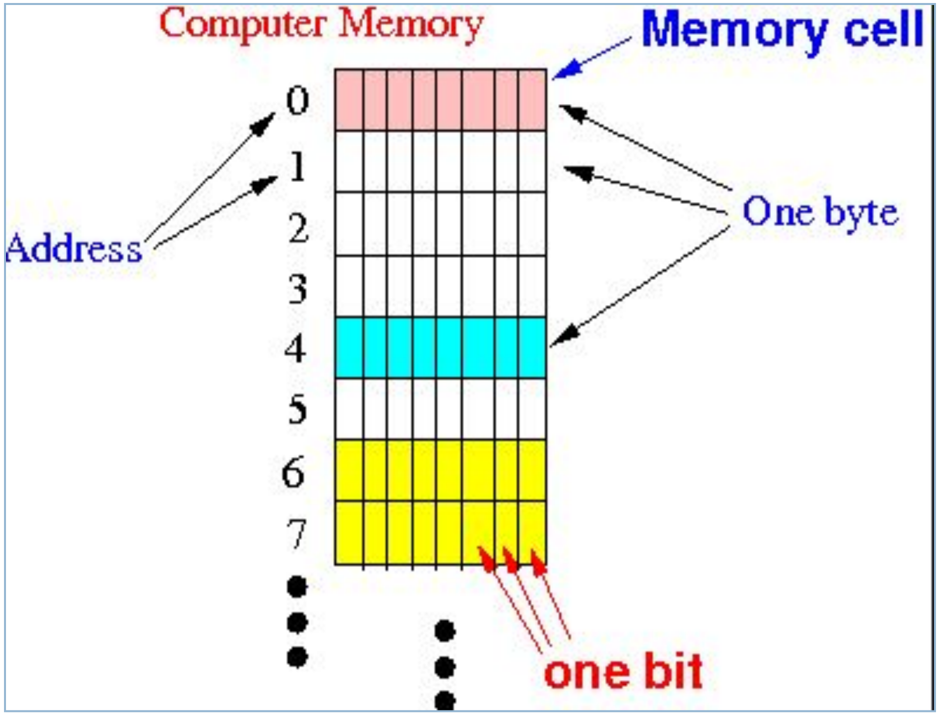
It is a storage unit, a place to hold data and instruction.



Memory Location and Addresses

Memory Organisation

- Memory consist of many millions of storage cells.
- Each cell can store 1-bit information having value (0 or 1)
- Memory organized as group of n-bits



Memory Location and Addresses

- Group of n-bits is referred as word
- n is called word length
- The word length of 8 bit is known as byte
- Word length can range from 16 to 64bits, simple called word.

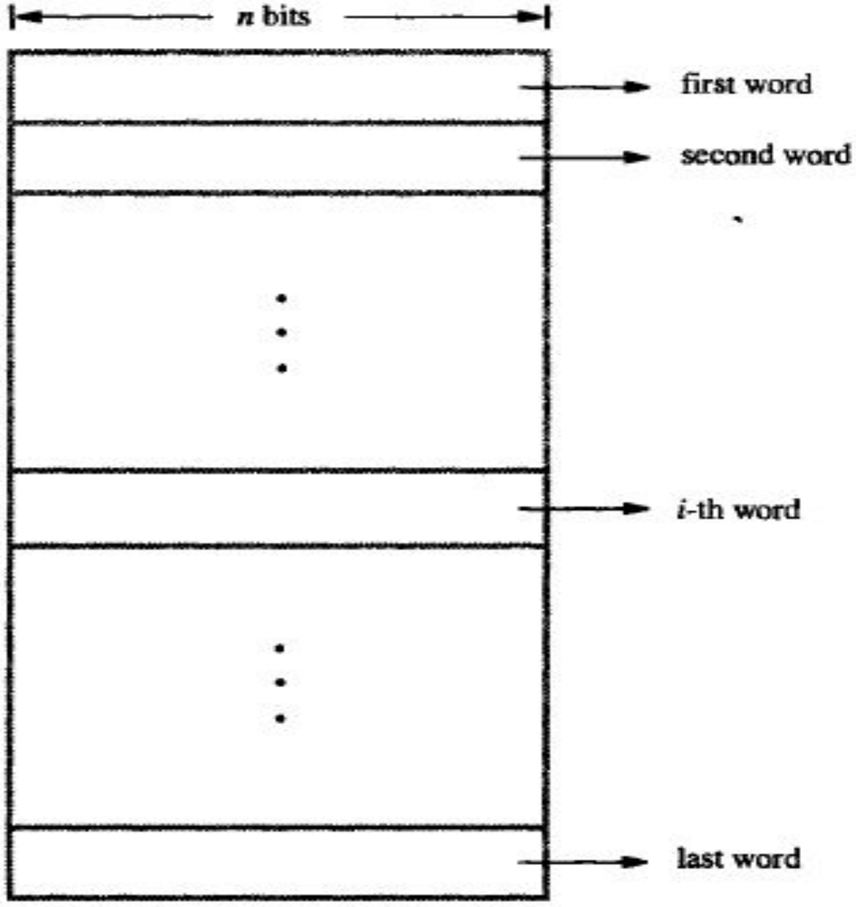


Figure 2.5 Memory words.

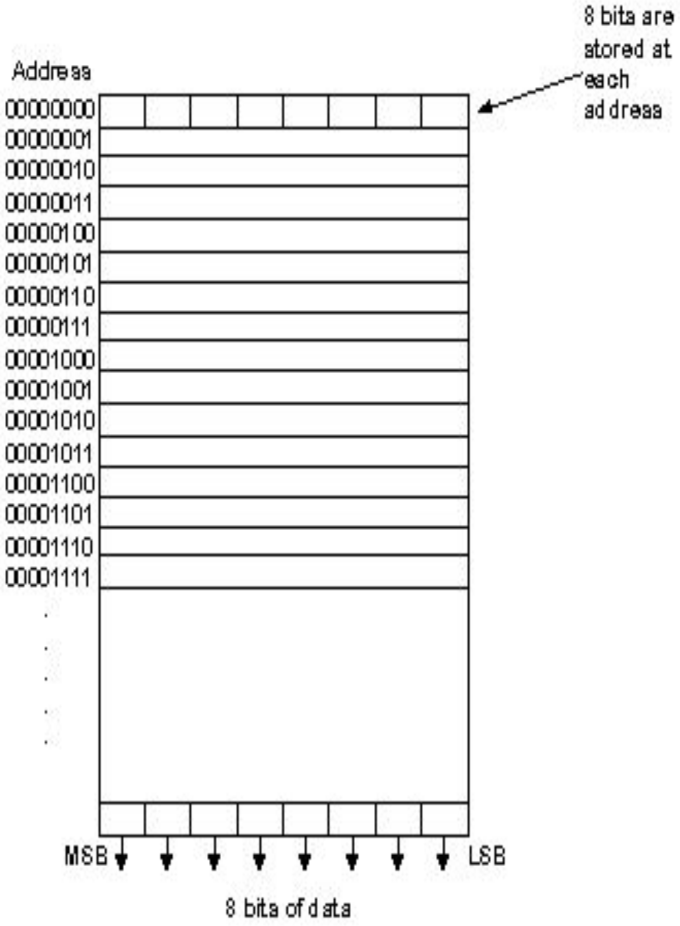
Memory Location and Addresses

- □ Memory Location

  Memory is divided into multiple small parts, of fixed size and capable of holding information.

- □ Memory Address

  - ? Each memory location is uniquely identified by a BINARY ADDRESS(HEXA VALUE).



Memory Location and Addresses

- □ Memory Address

  If 2^k addresses constitute the address space of computer, then memory can have up to 2^k addressable locations.

  Example

  k= 10 , 2^{10} = 1,024(1K kilo) locations

  k= 20 , 2^{20} = 1,048,576(1M mega) locations

  k= 24 , 2^{24} = 16,777,216(16M mega) locations

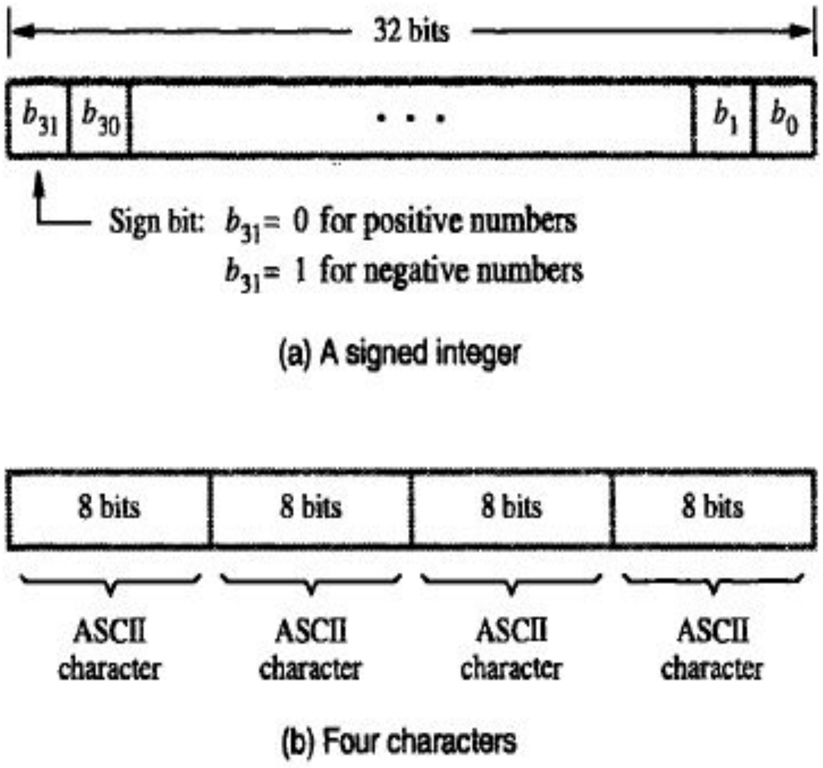
  k= 30 , 2^{30} = 1G (giga) locations

  k= 32 , 2^{32} = 4G (giga) locations

  k= 40 , 2^{40} = 1T(tera) locations

Memory Location and Addresses

- □ If the word length is 32 bits, single word can store 32 bits or 4 ASCII characters as shown in the Figure .



(a) A signed integer

(b) Four characters

Examples of encoded information in a 32-bit word.

Memory Location and Addresses

- It is impractical for bit address locations
- But prefer byte address location – byte addressable memory

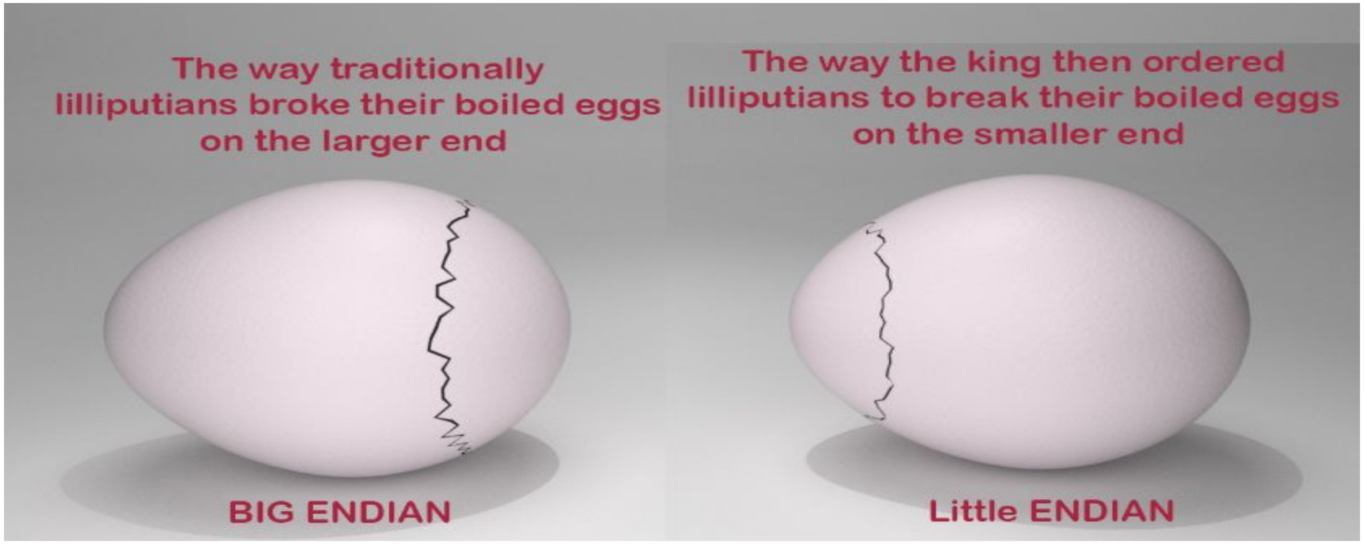
Byte Addressable memory

Each Memory address refers to a single byte of storage. To store data of large size, multiple consecutive locations are used

Memory Location and Addresses

Two ways Byte address can be assigned across words

- Big-Endian Assignments
- Little-Endian Assignments


The way traditionally lilliputians broke their boiled eggs on the larger end
BIG ENDIAN
The way the king then ordered lilliputians to break their boiled eggs on the smaller end
Little ENDIAN

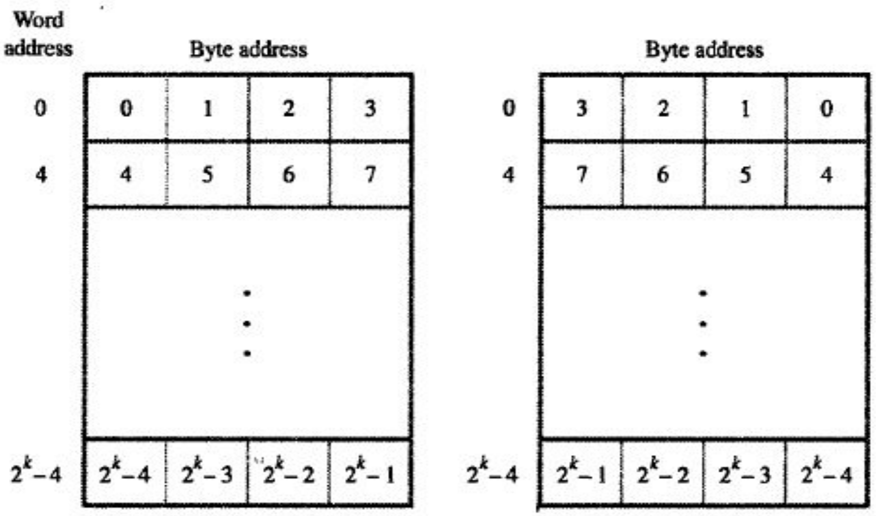
Memory Location and Addresses

Big Endian

The MSB of the data is placed at the byte with the lowest address.

Little Endian

The LSB of the data is placed at the byte with the lowest address.

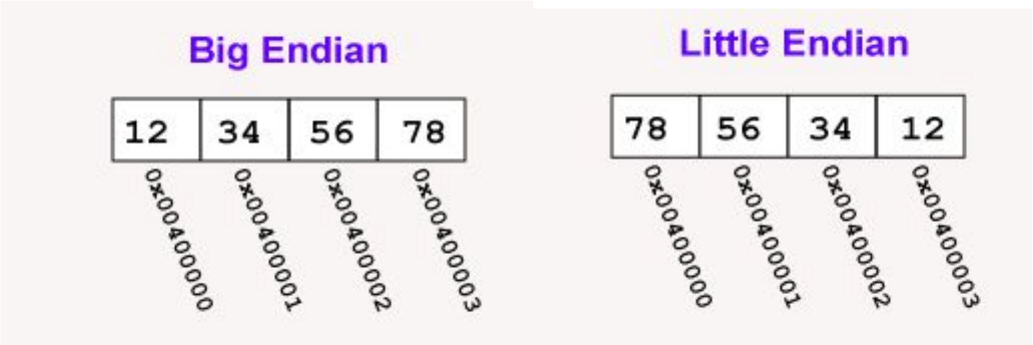


Word address	Byte address			
0	0	1	2	3
4	4	5	6	7
		⋮		
2^k-4	2^k-4	2^k-3	2^k-2	2^k-1

(a) Big-endian assignment

Word address	Byte address			
0	3	2	1	0
4	7	6	5	4
		⋮		
2^k-4	2^k-1	2^k-2	2^k-3	2^k-4

(b) Little-endian assignment



Big Endian

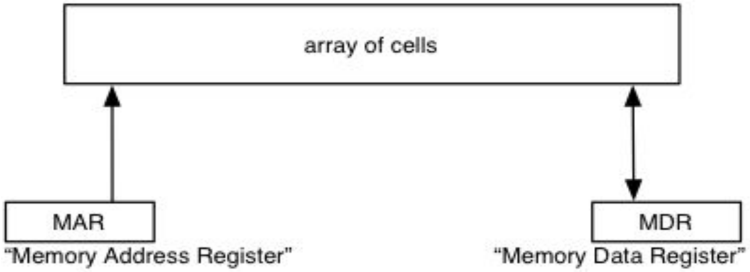
12	34	56	78
0x00400000	0x00400001	0x00400002	0x00400003

Little Endian

78	56	34	12
0x00400000	0x00400001	0x00400002	0x00400003

Memory Operations

- Load (or Read or Fetch)
  - Copy the content. The memory content doesn’t change.
  - Address – Load
  - Registers can be used
- Store (or Write)
  - Overwrite the content in memory
  - Address and Data – Store
  - Registers can be used



fetch (addr):
1. Put *addr* into MAR
2. Tell memory unit to “load”
3. Memory copies data into MDR

store (addr, new-value):
1. Put *addr* into MAR
2. Put *new-value* into MDR
3. Tell memory unit to “store”
4. Memory stores data from MDR into memory cell.

Instruction and Instruction sequencing

Computer performs 4 types of operations:

- Data transfer between memory and processor
- Arithmetic and logic operations on data
- Program sequencing and control
- I/O transfers

Data Transfer

- Transfer data from one location to another location
  - Memory locations
  - Processor registers
  - Registers in I/O system
- Location is given symbolic name
  - Name for memory address location A,LOC,VAR,PLACE
  - Name for Processor register names R1,R2,R3…..
  - Name for I/O register names DATAIN,DATAOUT

Data Transfer

- Register Transfer Notation
- Assembly Language Notation

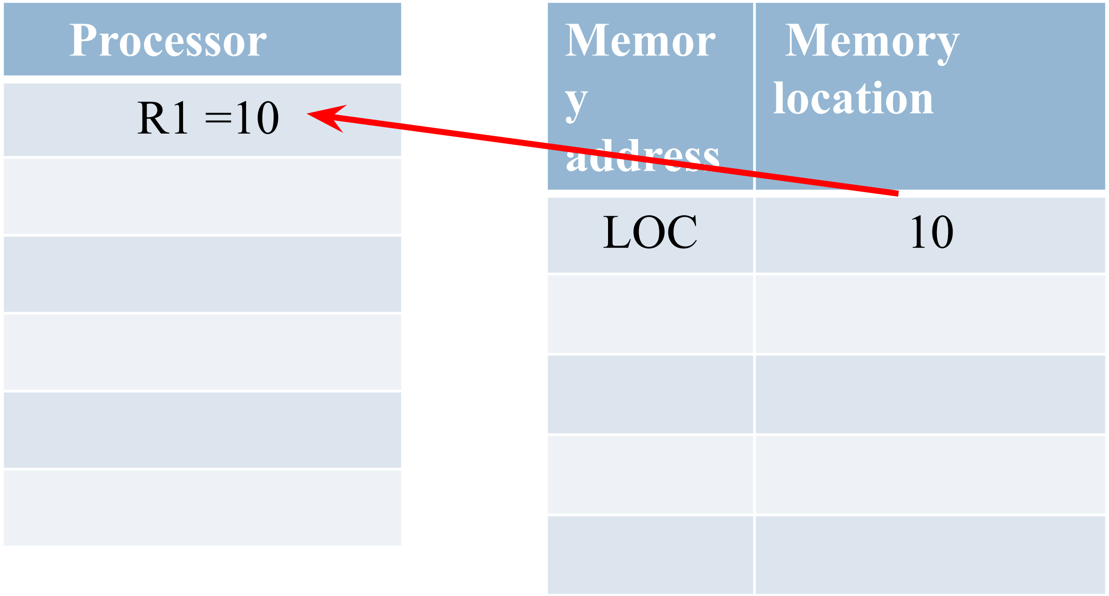
Data Transfer

- R3 □ [R1] + [R2]
- This type of notation is called RTN (Register Transfer Notation)

Processor
R1 =10
R2 = 20
R3 = 30

Data Transfer

- Contents of location are denoted by placing square brackets around the name of the location
- R1 □ [LOC]



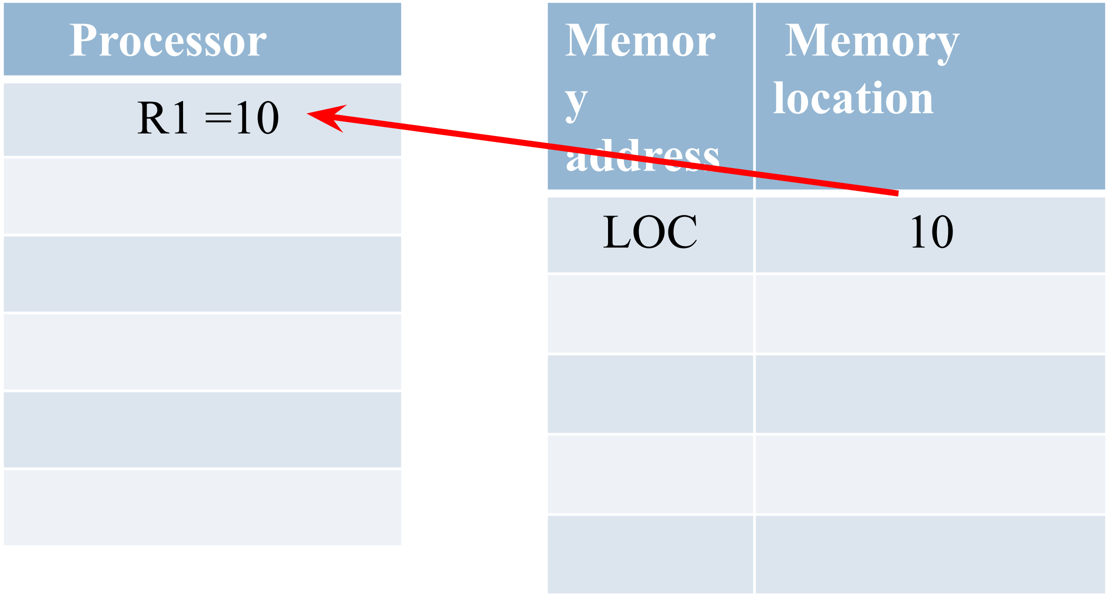
Processor
R1 =10

Memory address	Memory location
LOC	10

Data Transfer

- Assembly Language Notation

Move LOC, R1



Processor
R1 =10

Memory address	Memory location
LOC	10

Data Transfer

Assembly Language Notation

Add R1,R2,R3

Processor
R1 =10
R2 = 20
R3 = 30

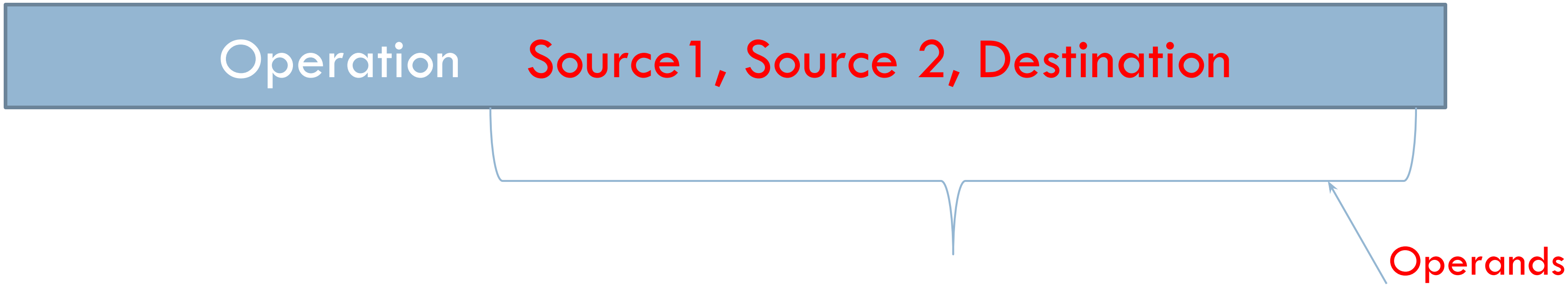
Instruction and Instruction sequencing

- Basic instruction types
  - Three address instruction format
  - Two address instruction format
  - One address instruction format
  - Zero address instruction format

Basic Instruction Types

Instruction
Opcode
Operand(s) or Address(es)

- To represent machine instruction and program, we use assembly language format.
- General 3 address instruction format



  - Source 1, Source 2 are source operands
  - Destination are called destination operands

Basic Instruction Types

- Example C= A+B
- Add the value of variable A and B and store in C
- During compilation three variables A,B,C are assigned to distinct locations in memory
- Action
  - C □ [A]+[B]

Memory address	Memory location
A	10
B	20
C	30

Basic Instruction Types

- Three address instruction format

Add A, B ,C

- This instruction contain memory address of three operands
- Operands A and B – Source operand
- Operand C – Destination operand

Basic Instruction Types

- Three address instruction format

Add A, B ,C

- Operands A , B,C are memory address - k bits
  - 3 * 1000 = 3kbits for addressing purpose
- Operation Add – n bits
- Modern Processor is 32bit address space, then 3 address instruction format is too large to fit in one word
- So multiple word is required for single instruction

Basic Instruction Types

- Two address instruction format

Operation Source, Destination

Add A, B

- B □ [A]+[B]
- But here we are replacing the

  Content of original location B

Memory address	Memory location
A	10
B	20 30

Basic Instruction Types

- Two address instruction format

Operation Source, Destination

Move B, C
Add A, C

Memory address	Memory location
A	10
B	20
C	30

Basic Instruction Types

□ One address instruction format

Operation	Operand

□ Operand Specified in instruction can be source or destination, depending on the instruction

Basic Instruction Types

- One address instruction format

Operation	Operand

- When second operand is required, processor register like Accumulator can be used

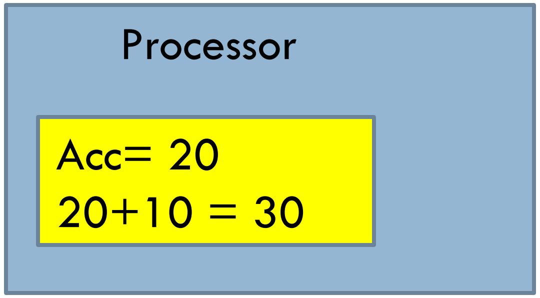
Add	A

Basic Instruction Types

- One address instruction format

Add A

- Add the content of memory location to the content of Accumulator and store the result in accumulator



Processor

Acc= 20
20+10 = 30

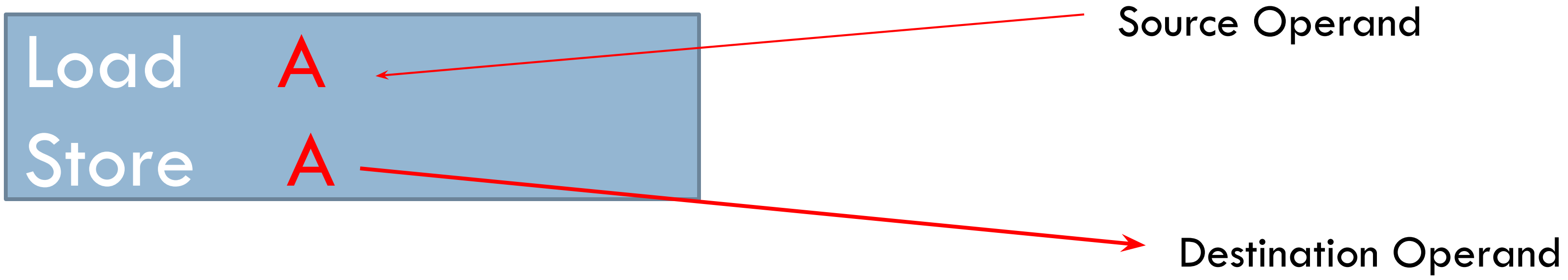
Memory address	Memory location
A	10

Basic Instruction Types

- One address instruction format

Operation Operand

- Operand Specified in instruction can be source or destination, depending on the instruction



Load A

Store A

Source Operand

Destination Operand

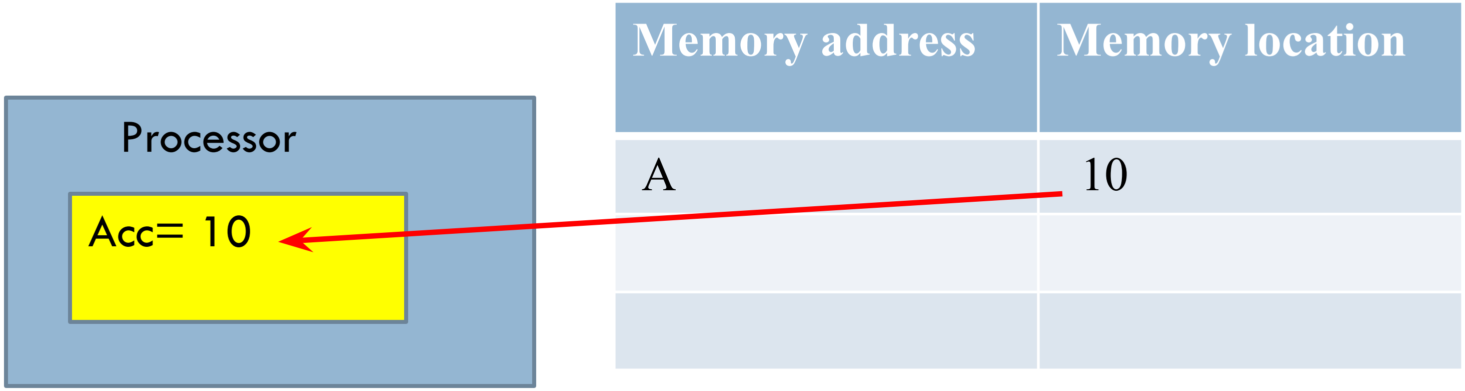
Basic Instruction Types

Add A

Load A

Store A

- Load A - copies the contents of memory location A into accumulator



Memory address	Memory location
A	10

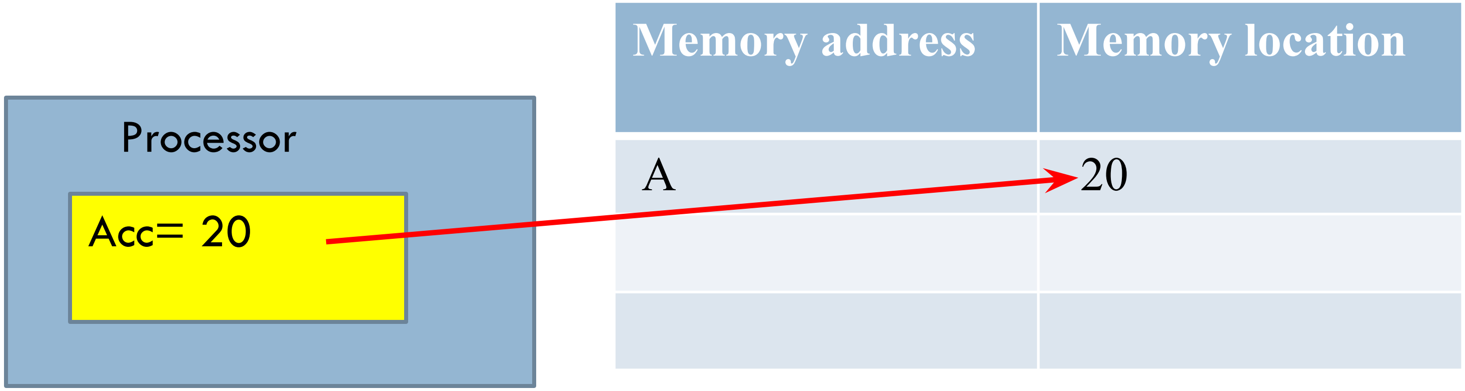
Basic Instruction Types

Add A

Load A

Store A

- Store A - copies the contents of accumulator into memory location A



Processor

Acc= 20

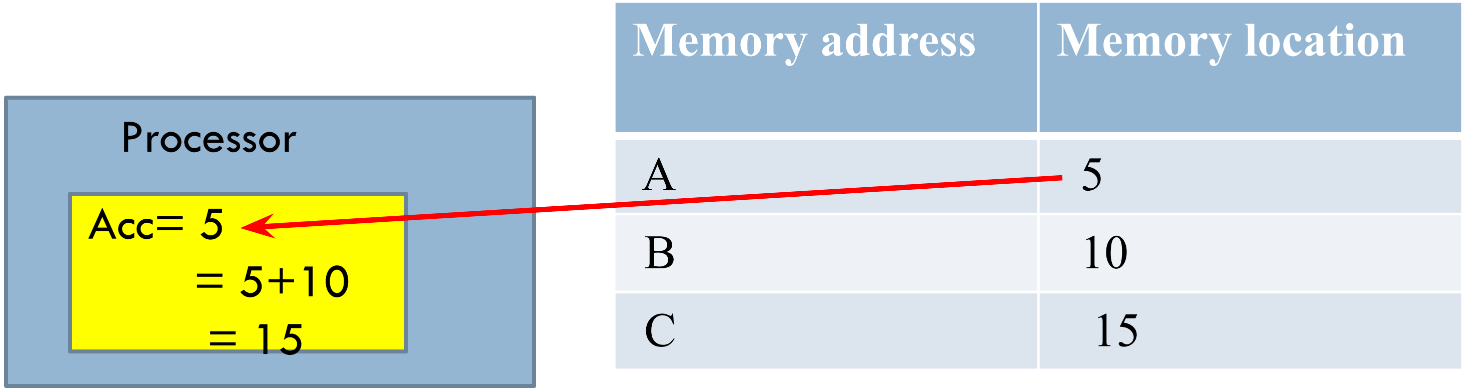
Memory address	Memory location
A	20

Basic Instruction Types

Add A,B,C

Load A
Add B
Store C

□ C □ [A]+[B]



Processor

Acc= 5
= 5+10
= 15

Memory address	Memory location
A	5
B	10
C	15

Basic Instruction Types

- Processor has several general purpose register
- Many instruction involve operands that are in the register
- Processor computations are performed directly on data held in processor register

Add R_i, R_j or

Add R_i, R_j , R_k

Processor
R1 =10
R2 =20
R3 =30

Basic Instruction Types

- □ Transfer data between different location

Move Source Destination

- □ Places a copy of the content of source to destination

Move R1 , R2

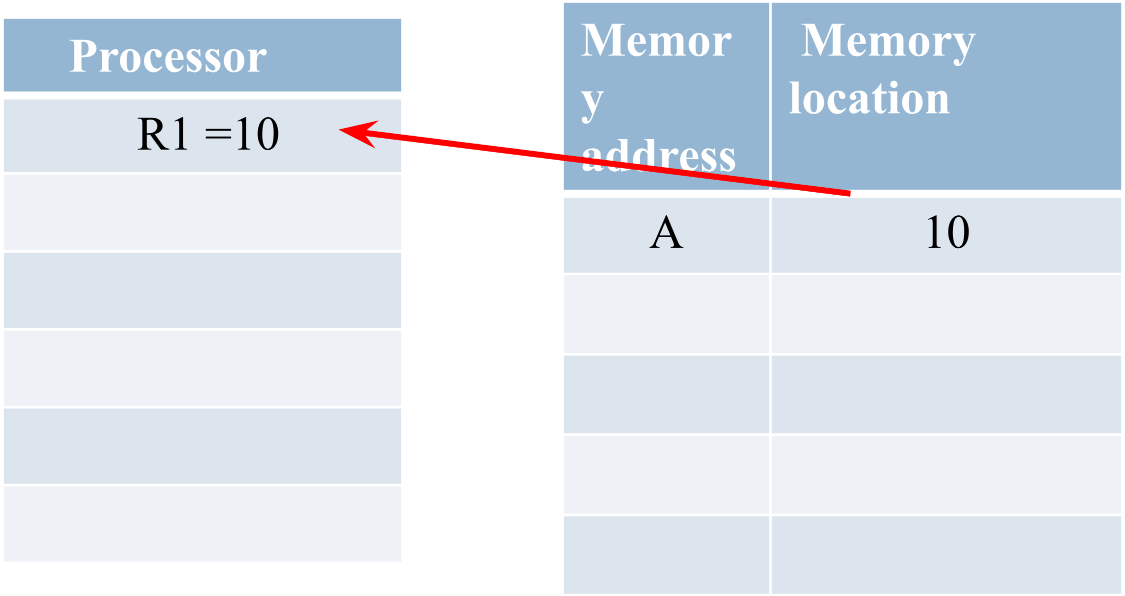
Processor
R1 =10
R2 =10

Basic Instruction Types

Move A , R1

Load A , R1

□ Both are same

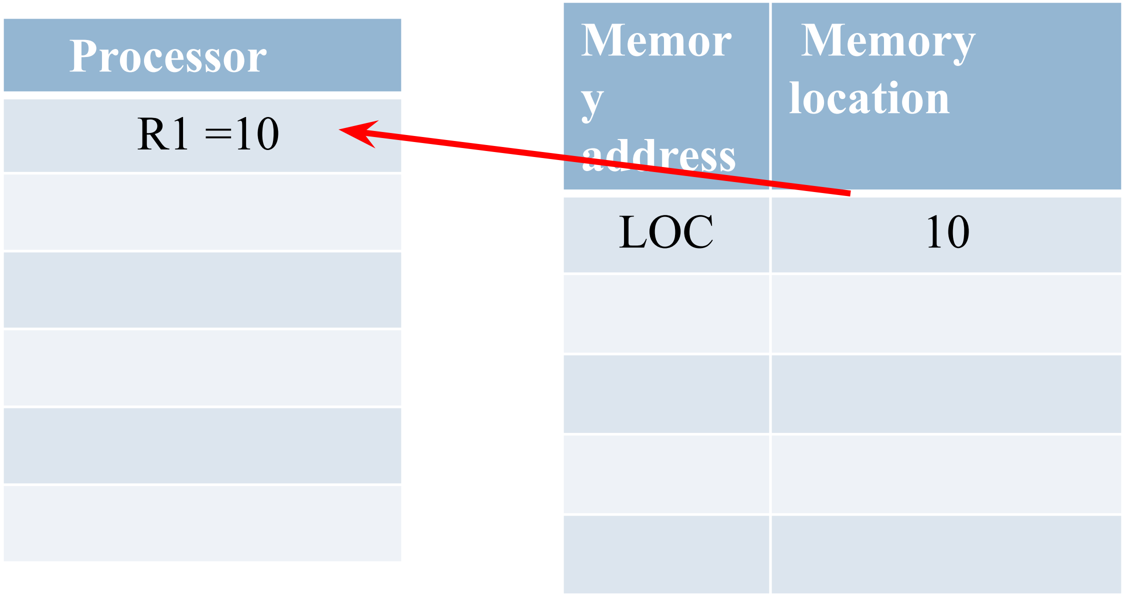


Processor
R1 =10

Memory address	Memory location
A	10

Basic Instruction Types

Move LOC , R1



Processor
R1 =10

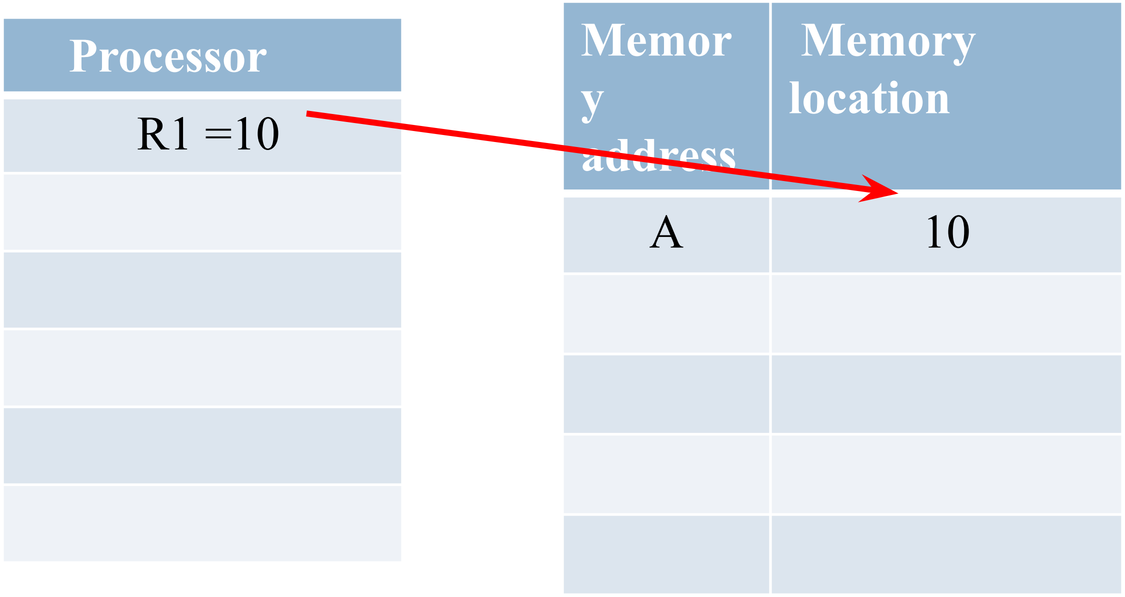
Memory address	Memory location
LOC	10

Basic Instruction Types

Move R1, A

Store R1, A

- □ Both are same



Processor
R1 =10

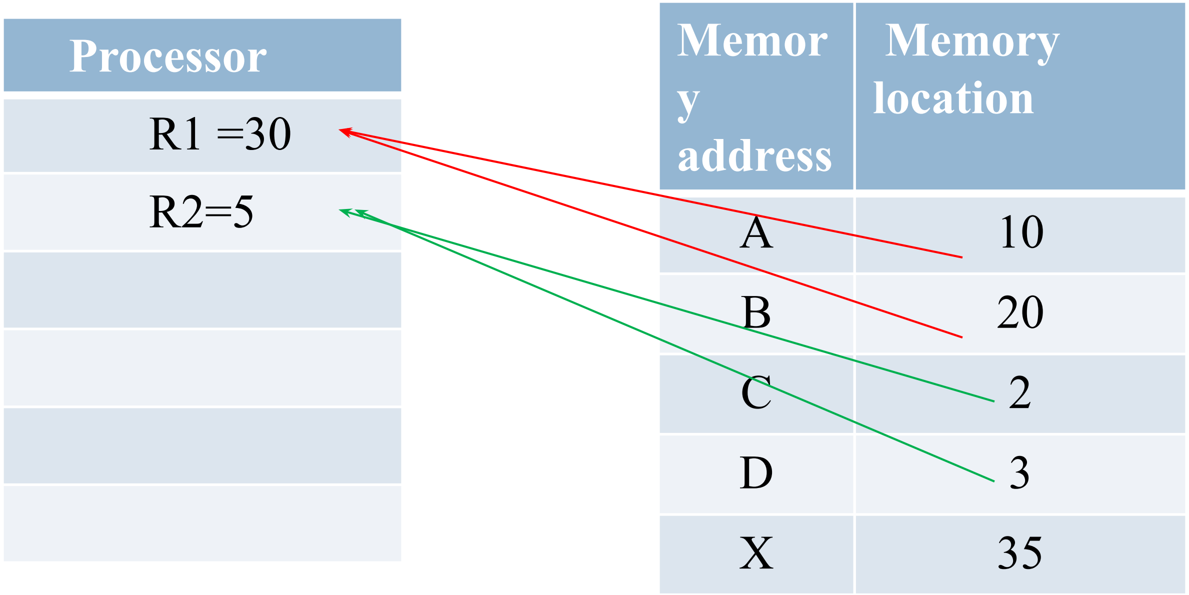
Memory address	Memory location
A	10

Basic Instruction Types

- Zero address instruction format
  - It is also possible to use instruction in which locations of all the operands are defined implicitly.
  - Such instruction are found in machines that stores operands in a structure called pushdown stack

Example: Evaluate (A+B) * (C+D)

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



Processor
R1 =30
R2=5

Memory address	Memory location
A	10
B	20
C	2
D	3
X	35

Example: Evaluate (A+B) * (C+D)

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

Processor
R1 =30
R2=150

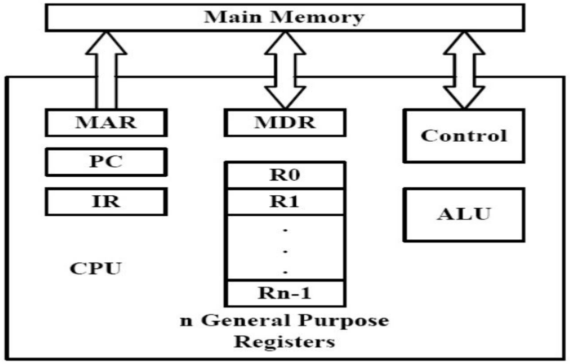
Memory address	Memory location
A	10
B	20
C	2
D	3
X	150

Unit -1

- Functional units, Bus structures, performance,
- **Overflow in integer arithmetic:** Numbers, Arithmetic operations and characters
- Memory locations and addresses,
- Memory operations,
- Instructions and instruction sequencing,
- Addressing modes,
- Subroutines and use of stack frames,
- Encoding of machine instructions.

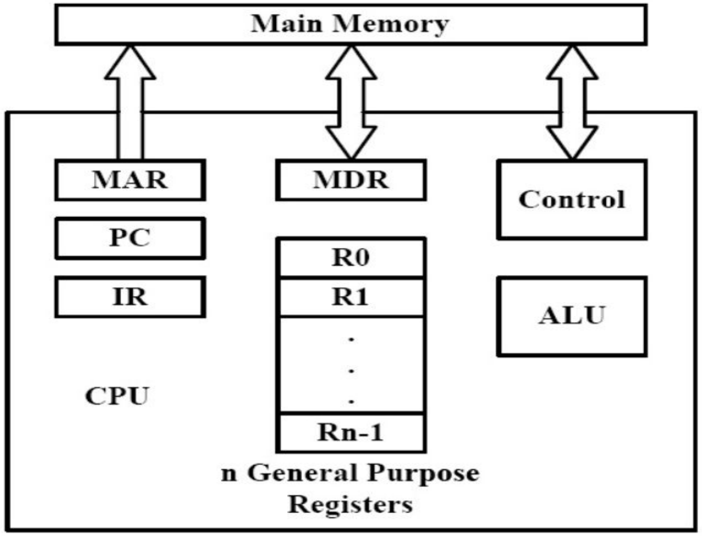
Instruction Execution and Straight-Line Sequencing

- Executing a given instruction is a two phase procedure
  - Instruction Fetch
  - Instruction Execute
- Instruction Fetch – instruction is fetched from memory location whose address is in PC, and placed in IR.
- Instruction Execute – instruction placed in IR is examined and the required operation is done. PC is incremented to point to next instruction.



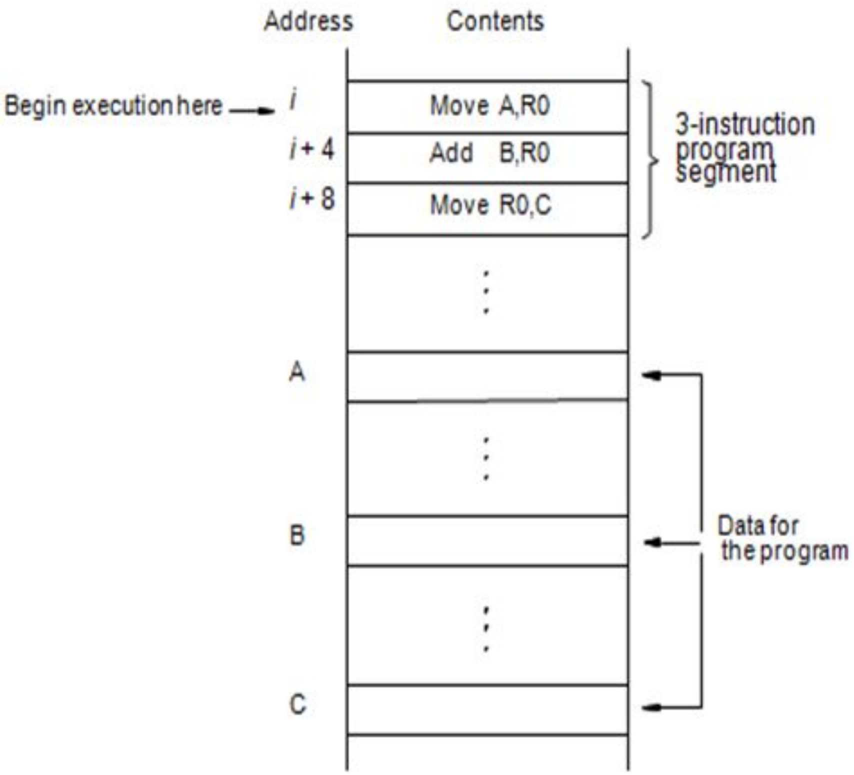
Instruction Execution and Straight-Line Sequencing

- To begin executing a program, the address of first instruction will be placed in program counter, then processor control circuit will use the information in PC to fetch and execute instruction one at a time, in the order of increasing addresses. This is called Straight-Line Sequencing

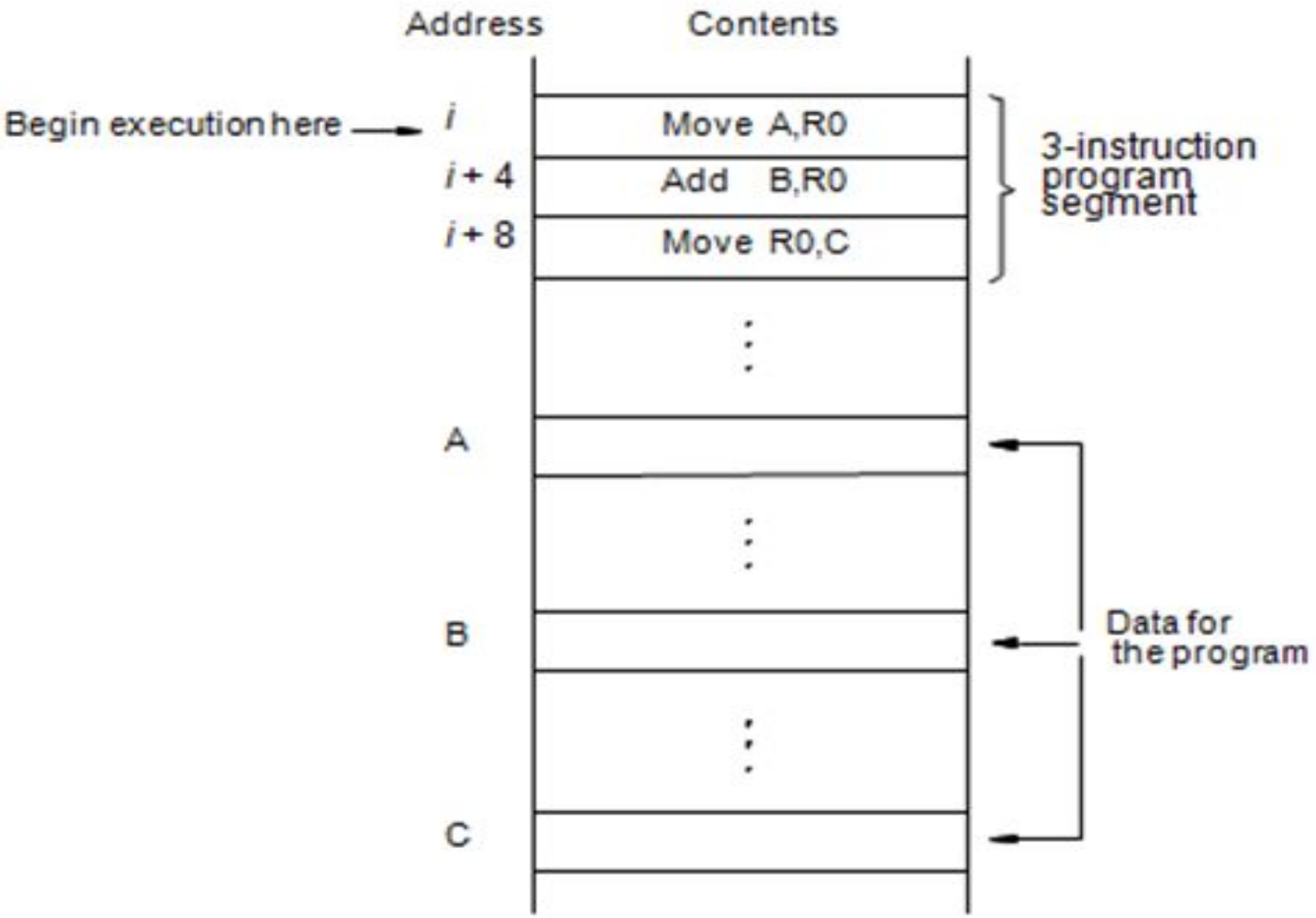


Instruction Execution and Straight-Line Sequencing

- Write a program for C □ [A] +[B]
  1. MOV A,R0
  2. ADD B,R0
  3. MOV R0,C
- Assume
  - Memory Word length is 32 bits
  - Memory is byte addressable
  - Three instructions are stored in successive word locations, starting at location i.
  - Each instruction is 4 bytes long, i+4, i+8



Instruction Execution and Straight-Line Sequencing

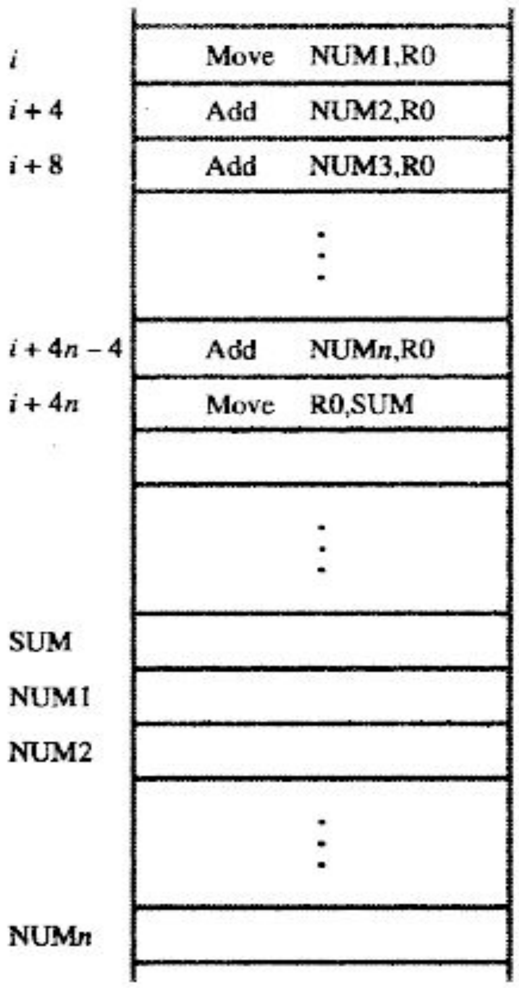


Instruction Execution and Straight-Line Sequencing

Consider the task to add a list of n numbers

□ The addresses of memory locations of n numbers are NUM1,NUM2,NUM3..

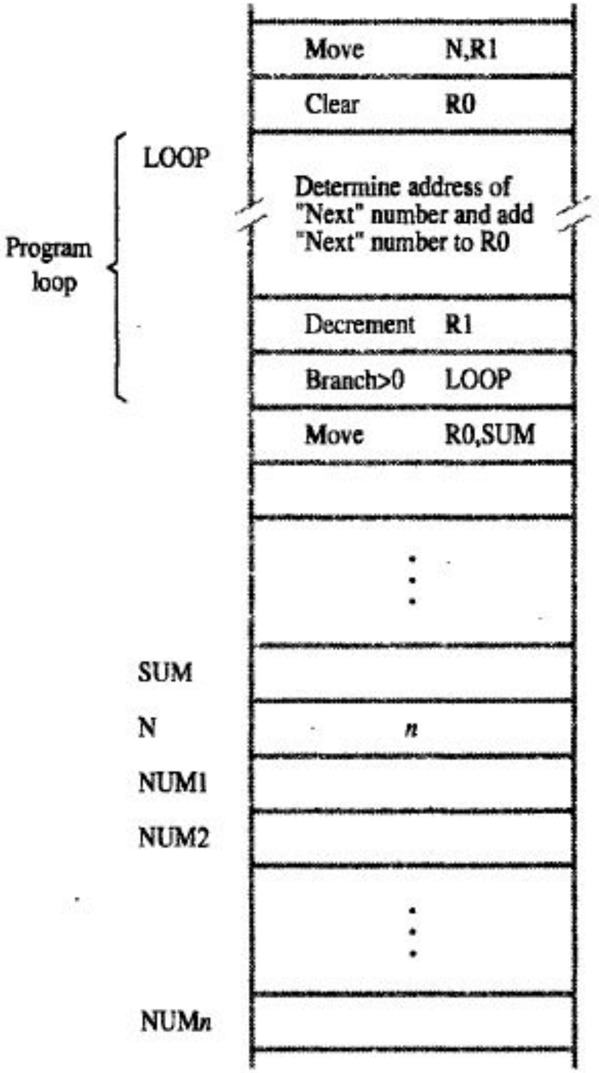
□ After adding the result is placed in SUM



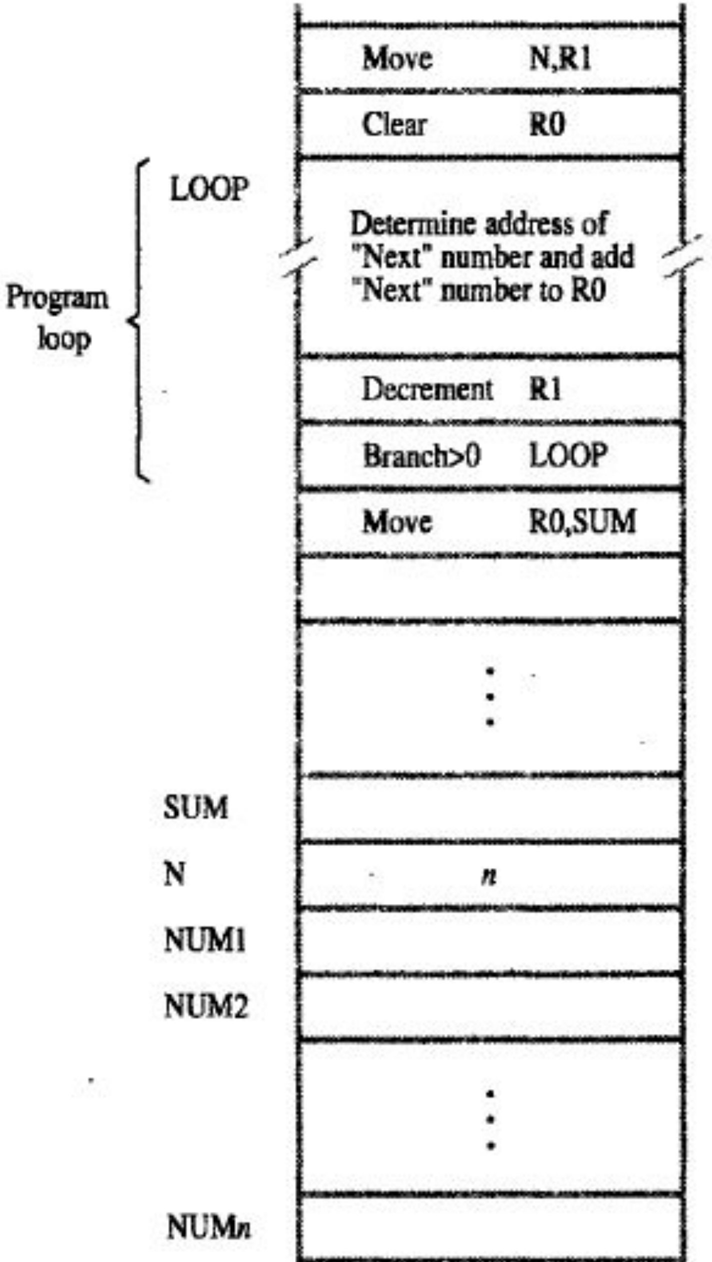
i	Move NUM1,R0
$i+4$	Add NUM2,R0
$i+8$	Add NUM3,R0
	⋮
$i+4n-4$	Add NUMn,R0
$i+4n$	Move R0,SUM
	⋮
SUM	
NUM1	
NUM2	
	⋮
NUMn	

Instruction Execution and Straight-Line Sequencing

- Instead of using long list of Add instructions, it is possible to place a single Add instructions in a program loop.
- This loop is straight line sequence of instruction executed as many times as needed
- It starts at location LOOP and ends at the Branch instruction (Branch >0)



d Straight-Line Sequencing



Processor
R1 =3/2/1
R0 =2+3+5

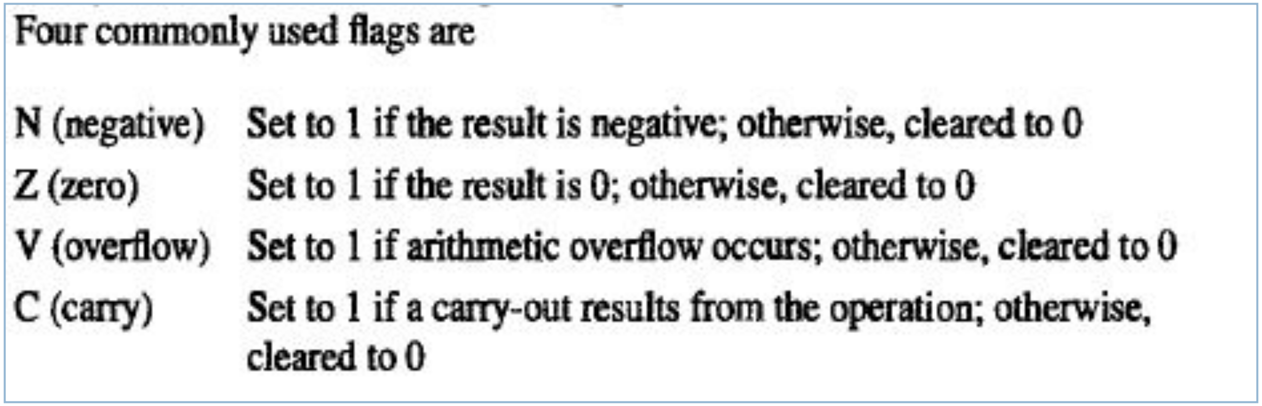
Memory address	Memory location
SUM	10
N	n=3
NUM1	2
NUM2	3
NUM3	5

Condition Codes

- Processor status is described as Condition codes or Status codes.
- Condition codes refers to the information about most recently executed information.
- This is accomplished by recording the required information in individual bits called condition code flags(0 or 1)

Condition Codes

- Condition code flags
  - N (negative)
  - Z (zero)
  - V (overflow)
  - C (carry)



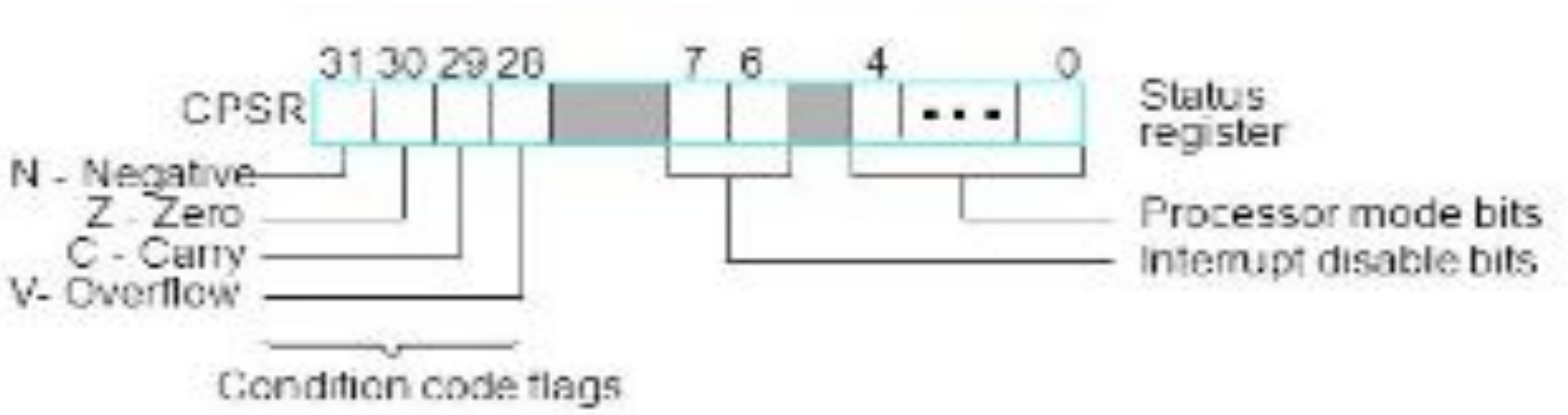
Four commonly used flags are

N (negative) Set to 1 if the result is negative; otherwise, cleared to 0

Z (zero) Set to 1 if the result is 0; otherwise, cleared to 0

V (overflow) Set to 1 if arithmetic overflow occurs; otherwise, cleared to 0

C (carry) Set to 1 if a carry-out results from the operation; otherwise, cleared to 0



Unit -1

- Functional units, Bus structures, performance
- Overflow in integer arithmetic: Numbers, Arithmetic operations and characters
- Memory locations and addresses
- Memory operations
- Instructions and instruction sequencing
- Addressing modes
- Subroutines and use of stack frames
- Encoding of machine instructions

Generating memory address

- How to specify the address of an operand
  - The instruction set of a computer provides a method called Addressing modes
  - Addressing modes : Different ways in which the location of an operand is specified is called addressing modes.
  - Important addressing modes found in modern processor

Addressing modes

□ Different Addressing modes are :

1. Register mode
2. Absolute mode
3. Immediate mode
4. Indirect mode
5. Index mode
6. Base with Index mode
7. Base with Index and Offset mode
8. Autoincrement mode
9. Autodecrement mode

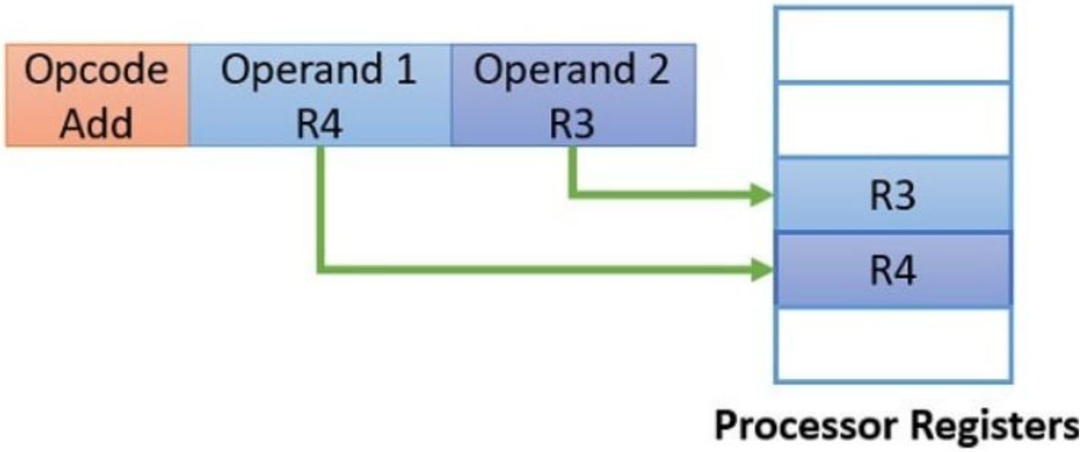
Addressing modes

- Two Addressing modes to access variables
  1. Register mode
  2. Absolute mode

Addressing modes

- Register mode
  - Every instruction includes operands; the operands can be a memory location, a processor register or an I/O device.
  - The instruction which uses processor **registers** to represent operands is the instruction in **register addressing mode**.

Add R4, R3,   Load R3, R2



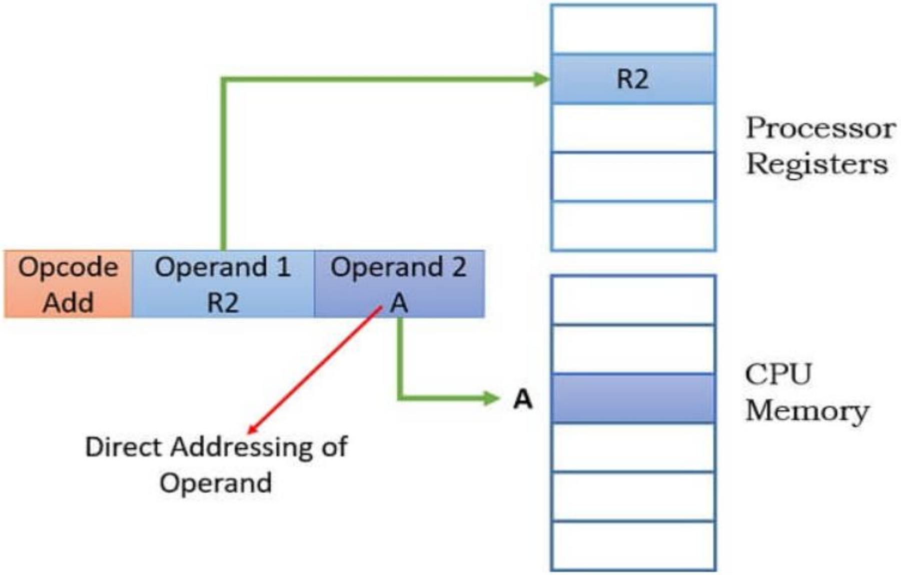
Addressing modes

- Register mode
  - Effective address is the location of an operand of the instruction. Operand is the data to be accessed.
  - EA=R

Add R4, R3,   Load R3, R2

Addressing modes

- Absolute mode(Direct mode)
  - The direct addressing mode is also known as **Absolute Addressing mode**.
  - Here, the instruction contains the address of the **location in memory** where the value of the operand is stored.
  - EA = A
  - Add R2, A
  - Store R2, B



Addressing modes

- Addressing modes to represent constants

  3. Immediate mode

Addressing modes

- Immediate mode
  - In immediate addressing mode, the value of the operand is **explicitly mentioned** in the instruction.
  - Here, effective address is not required as the operand is explicitly defined in instruction.

Addressing modes

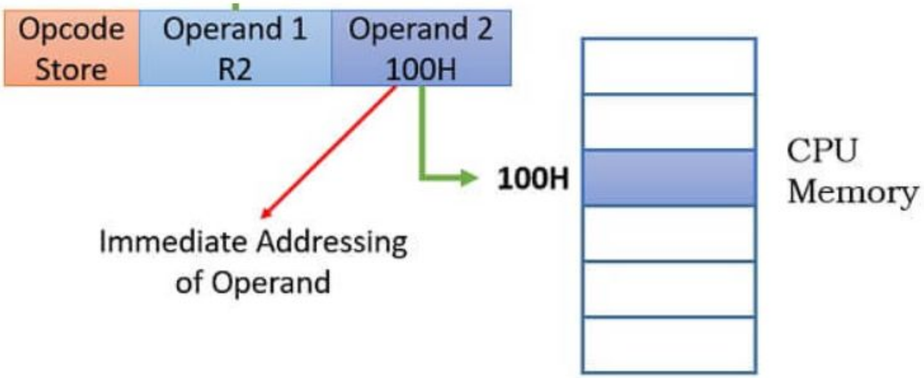
- Immediate mode
  - The Add instruction, adds 100 to R2’s content .
  - The # sign in front of the value indicates the immediate value to be operated.
  - If a value does not have # sign in front of it then it is the address of a memory location.

```
Add #100, R2
```

Addressing modes

- Immediate mode
  - Store considers the immediate value 100H as address as it does not have # sign in front of it.
  - The Store instruction stores the content of R2 at memory location 100H.

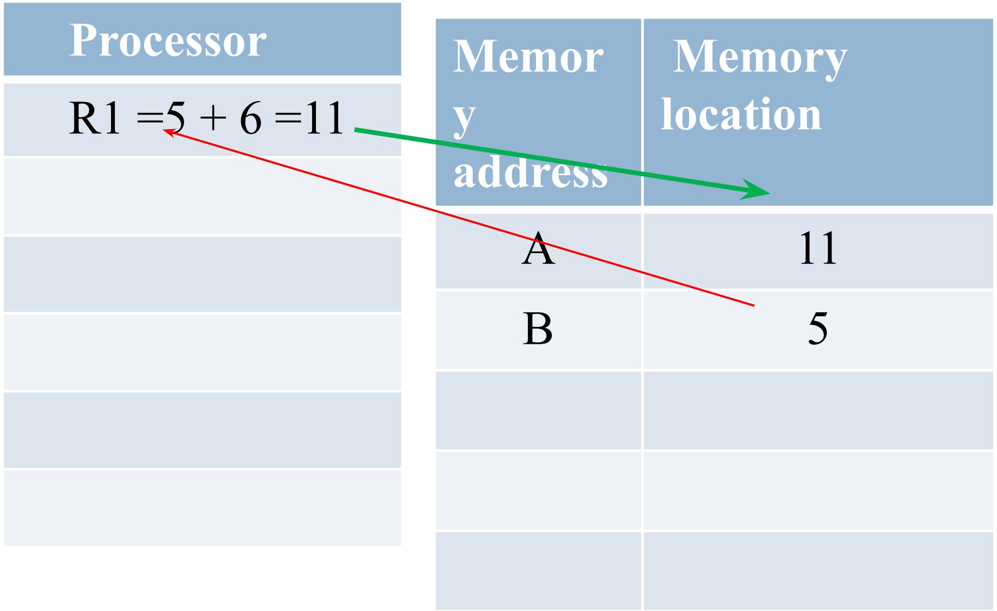
Store R2,100H



Addressing modes

- Example : A= B+6

```
Move      B , R1
Add       #6 , R1
Move      R1, A
```



Processor
R1 =5 + 6 =11

Memory address	Memory location
A	11
B	5

Addressing modes

- Addressing modes to Indirection and Pointers

  4. Indirect mode

Addressing modes

Indirect mode

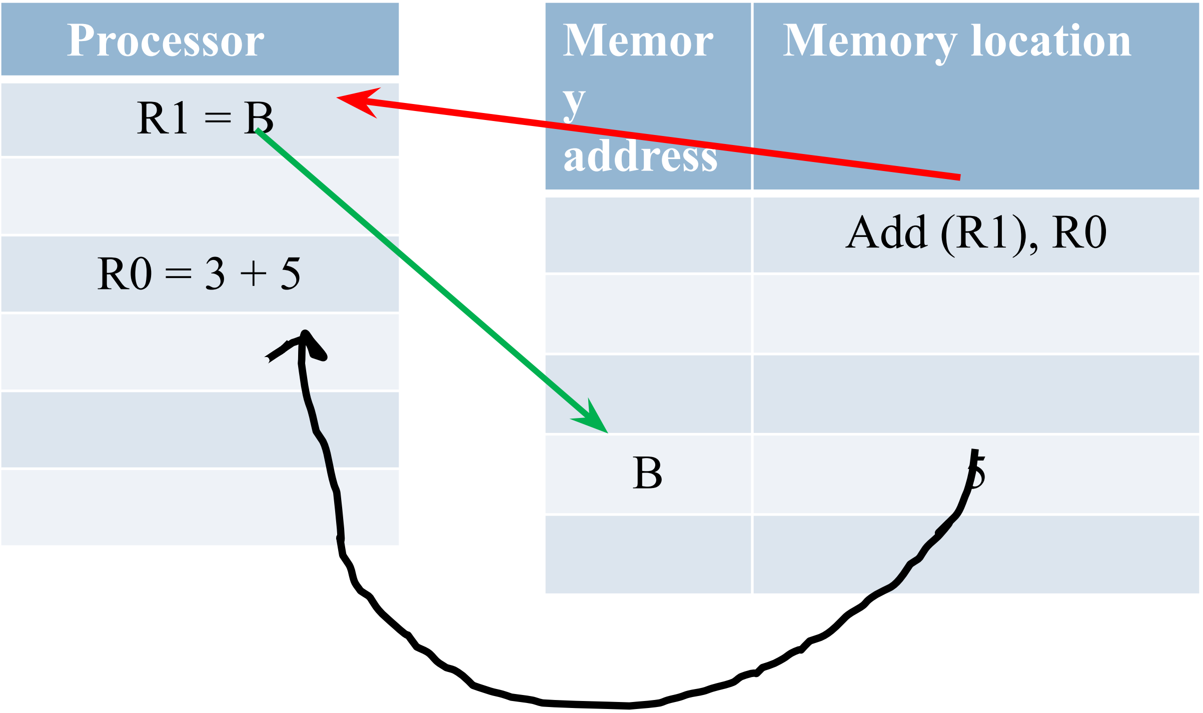
- A processor register is used to hold the address of a memory location where the operand is placed.
- In higher-level language, it is referred to as pointers.
- The indirect mode is denoted by placing the register inside the parenthesis.
- Here the effective address is the content of memory location present in the register.

EA=(R)

Addressing modes

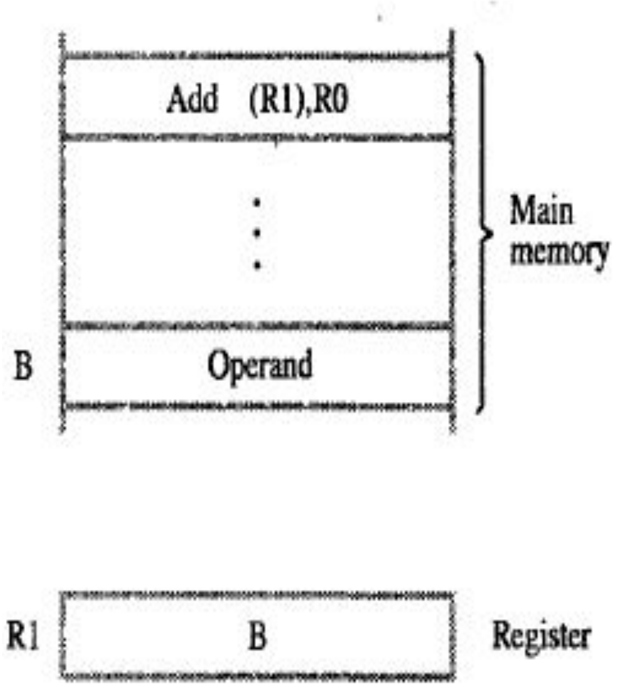
- Example to execute Add instruction Sum =5+3;

Add (R1) , R0



Processor
R1 = B
R0 = 3 + 5

Memory address	Memory location
	Add (R1), R0
B	5

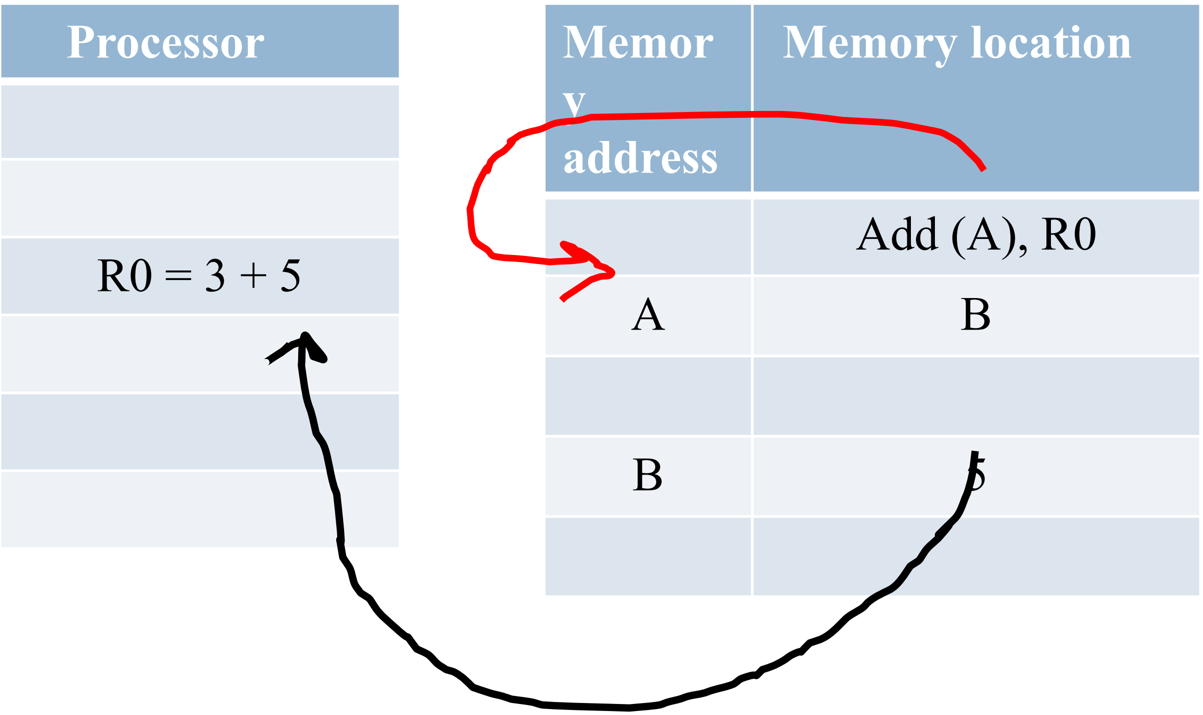


(a) Through a general-purpose register

Addressing modes

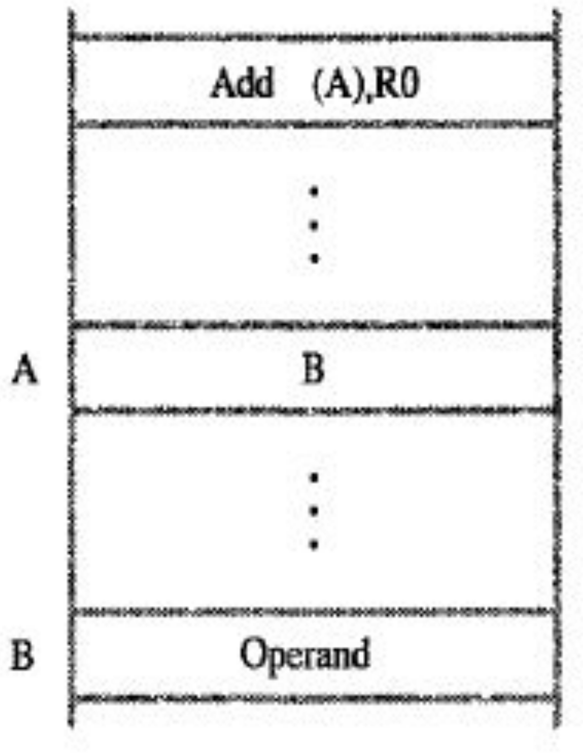
- Example to execute Add instruction Sum =5+3;

Add (A) , R0



Processor
R0 = 3 + 5

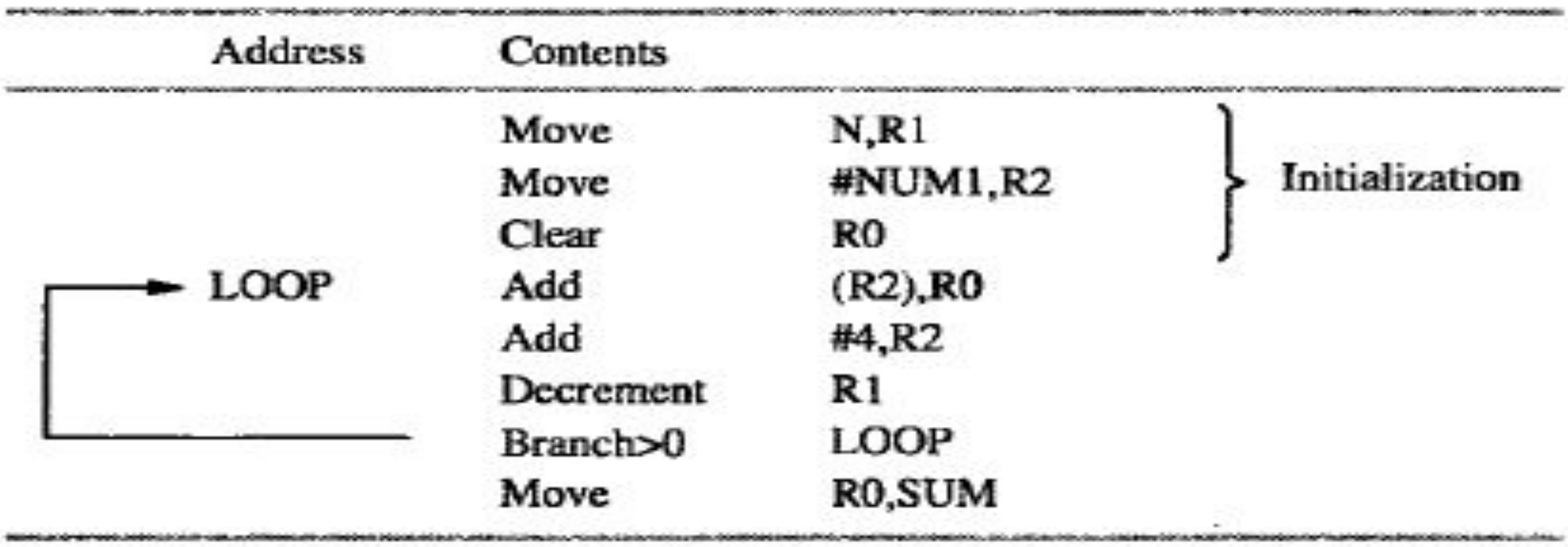
Memory address	Memory location
	Add (A), R0
A	B
B	5



(b) Through a memory location

Addressing modes

- Adding a list of numbers . Indirect addressing can be used used to access successive number in the list.



Address	Contents		
	Move	N,R1	Initialization
	Move	#NUM1,R2	
	Clear	R0	
LOOP	Add	(R2),R0	
	Add	#4,R2	
	Decrement	R1	
	Branch>0	LOOP	
	Move	R0,SUM	

Figure Use of indirect addressing in the program

Addressing modes

- Addressing modes to deal with lists and array

  5. Index mode

  6. Base with Index mode

  7. Base with Index and Offset mode

Addressing modes

- Index mode
  - The effective address of the operand is generated by adding a constant value to the contents of a register
  - Register □ General purpose registers (Index register)
  - Index mode □symbolically as $X(R_i)$
  - X denotes constant value contained in instruction, It can be an explicit number or symbolic name representing a numerical value
  - (R_i) denotes name of the register
  - $EA = X + [R_i]$

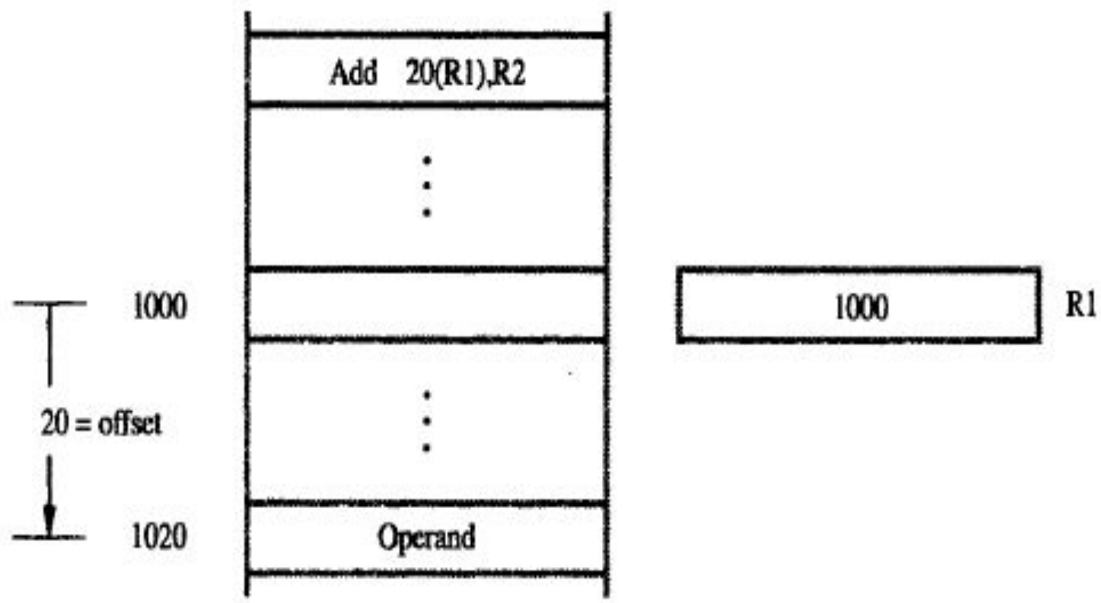
Addressing modes

- 2 ways of using Index mode –Offset given as constant

Add 20(R1) , R2

$EA = X + [R_i]$

X defines the Offset(displacement)



(a) Offset is given as a constant

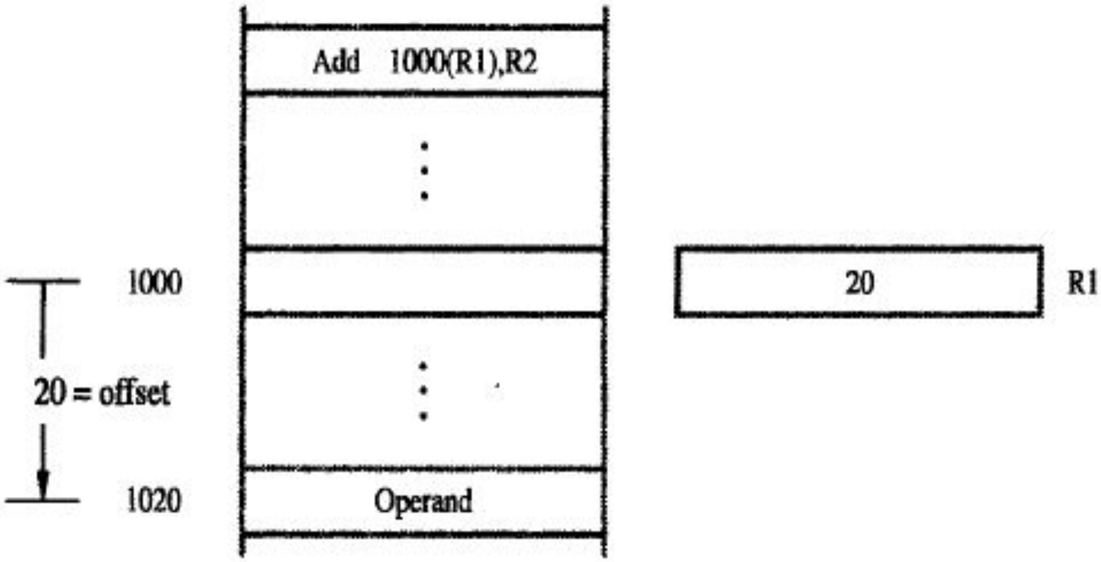
Addressing modes

- 2 ways of using Index mode –Offset is in index register

Add 1000(R1) , R2

$EA = X + [R_i]$

X corresponds to memory location



(b) Offset is in the index register

Addressing modes

- **Base with Index Mode**

  we know that Index mode EA $= X + [R_i]$

  - Sometimes second register can be used to contain the offset X, we can write index mode as (R_i, R_j)
  - ○ Effective address is the sum of the contents of register R_i and R_j.
  - EA = $[R_i] + [R_j]$

Addressing modes

- **Base with Index and offset Mode**

we know that Index mode EA = $X + [R_i]$

- ○ Sometimes index mode uses two registers plus a constant , we can write index mode as $X(R_i ,R_j)$
- ○ Effective address is the sum of the contents of register R_i , R_j and a constant.
- ○ We can write the Index mode as $X(R_i ,R_j.)$
- ○ $EA = X + [R_i\ ]+ [R_j\ ]$

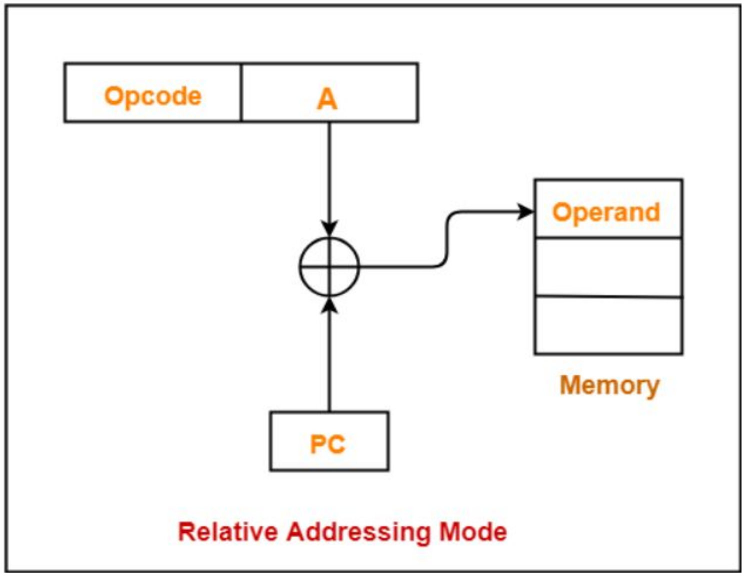
Addressing modes

- Addressing modes use PC instead of General purpose register

  8. Relative mode

Addressing modes

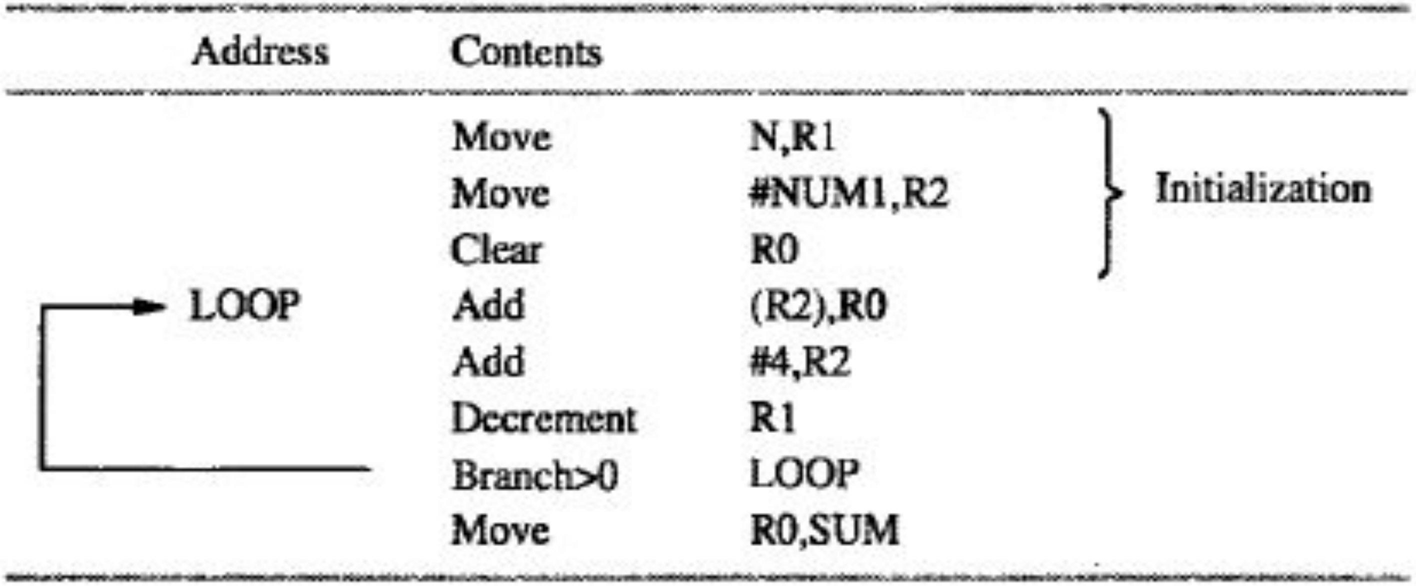
- Relative mode
  - The effective address is determined by the index mode using the PC in place of general purpose register R_i
  - Common use to specify target address in Branch instructions.



Addressing modes

- Example of Relative mode

Branch>0 LOOP



Address	Contents		
	Move	N,R1	Initialization
	Move	#NUM1,R2	
	Clear	R0	
LOOP	Add	(R2),R0	
	Add	#4,R2	
	Decrement	R1	
	Branch>0	LOOP	
	Move	R0,SUM	

Addressing modes

- Addressing modes for accessing data items in successive locations in memory

  9. Autoincrement mode

  10. Autodecrement mode

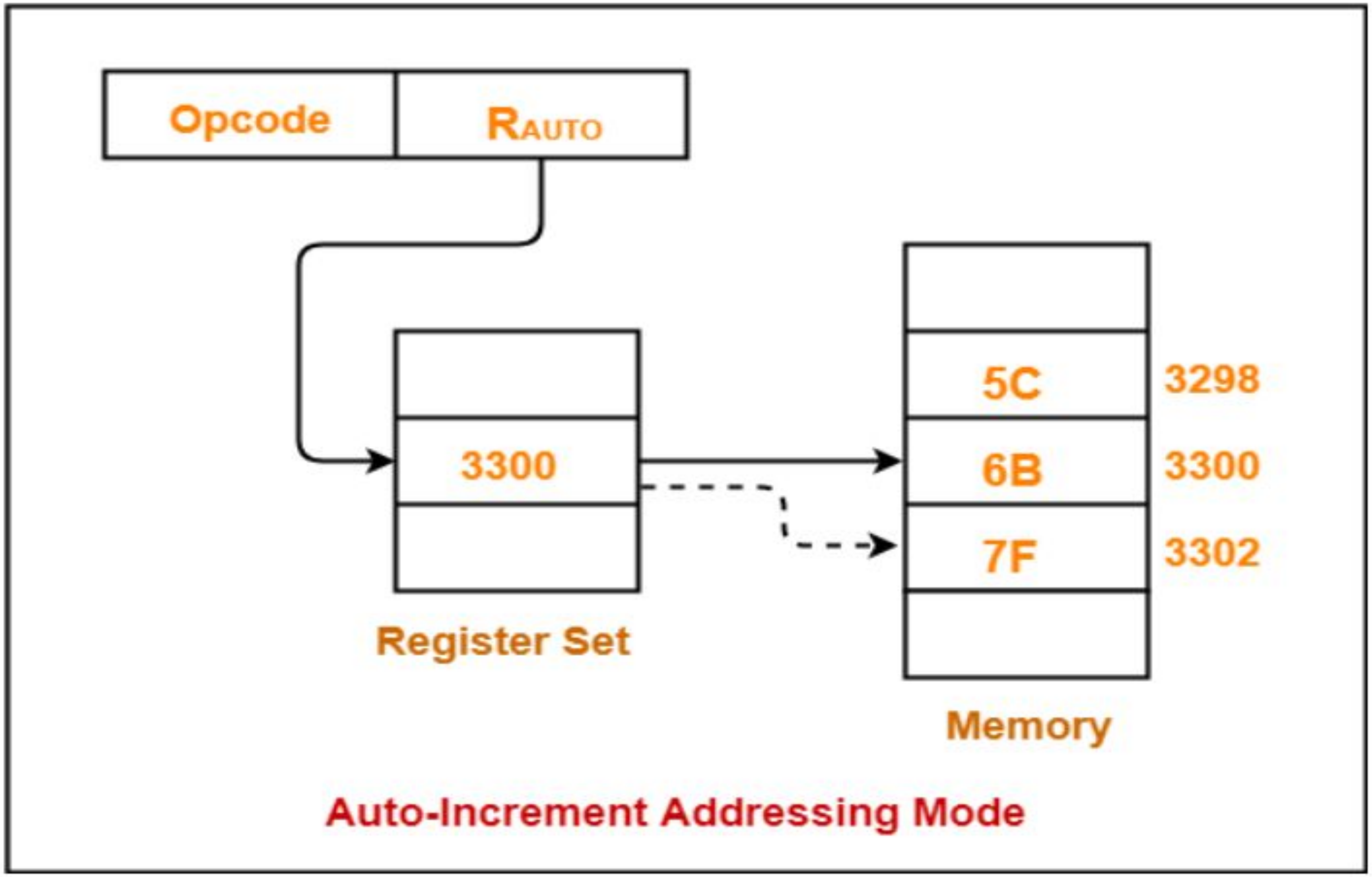
Addressing modes

- Autoincrement mode
  - The EA of the operand is the contents of register specified in instruction.
  - After accessing the operand, the contents of a register are automatically incremented to point to the next item in a list.
  - Written as $(R_i)+$

```
MOVE     R0,  (R2)+
```

Addressing modes

- Autoincrement mode



Auto-Increment Addressing Mode

Addressing modes

- Example

MOVE R0, (R2)+

Processor
R0 = 5

R2 = LOC(1000)
LOC1(1001)

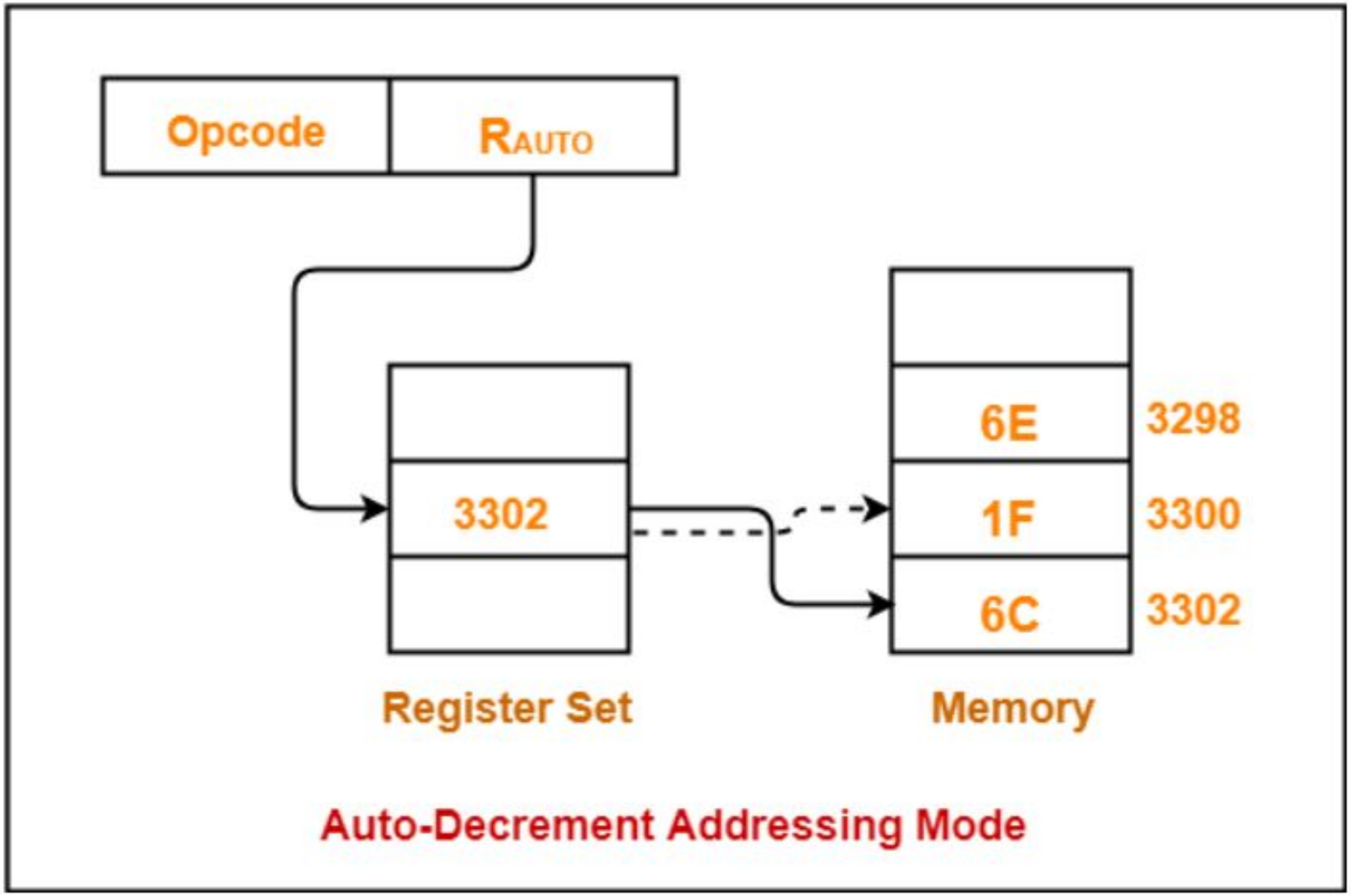
Memory address	Memory location
LOC(1000)	5
LOC1(1001)	

Addressing modes

- Autodecrement mode
  - The contents of a register are automatically decremented and are then used as EA of operand
  - Written as -(R_i)

Addressing modes

- Autodecrement mode

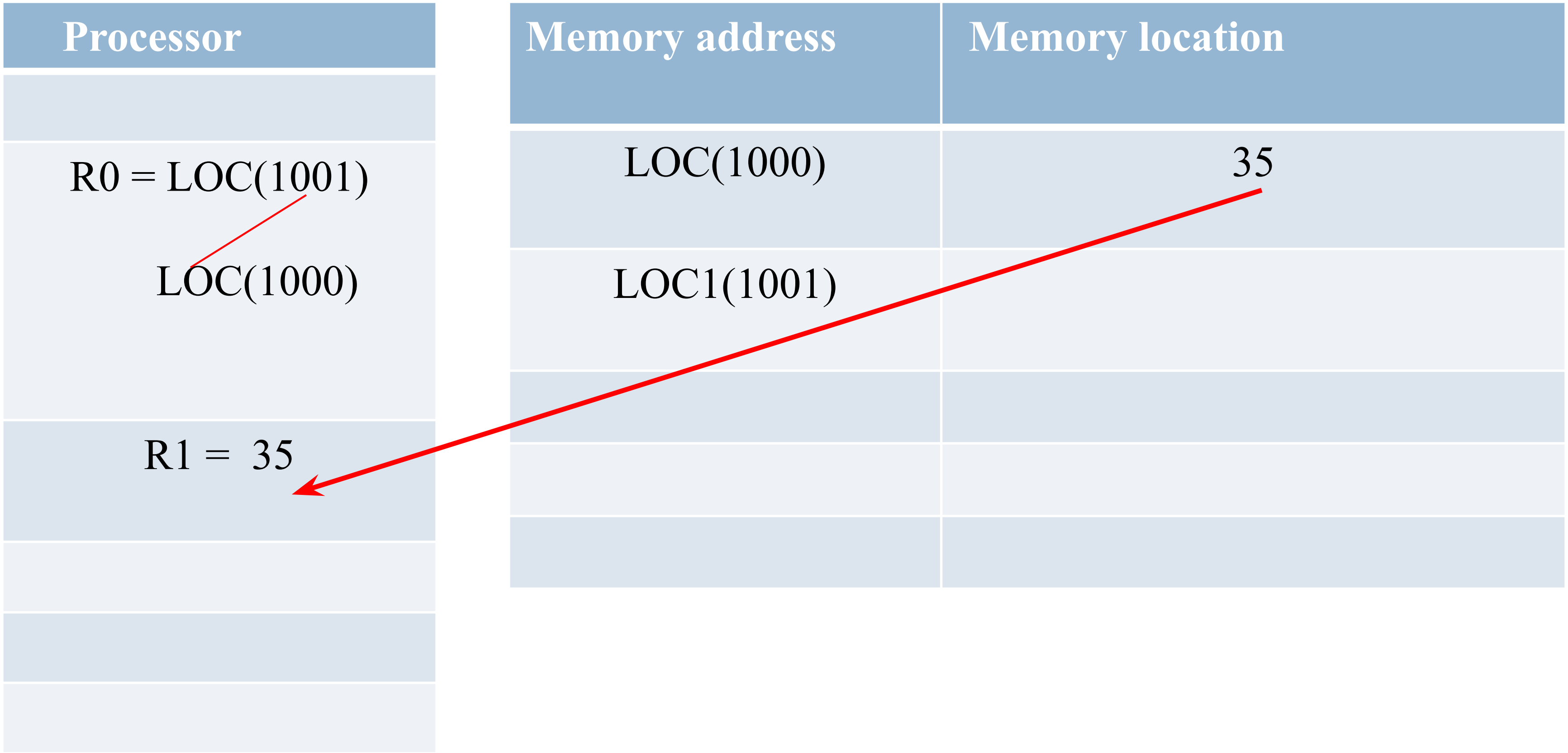


Auto-Decrement Addressing Mode

Addressing modes

- Example

MOVE -(R0), R1



Processor

R0 = LOC(1001)
LOC(1000)

R1 = 35

Memory address	Memory location
LOC(1000)	35
LOC1(1001)	

Generic addressing modes

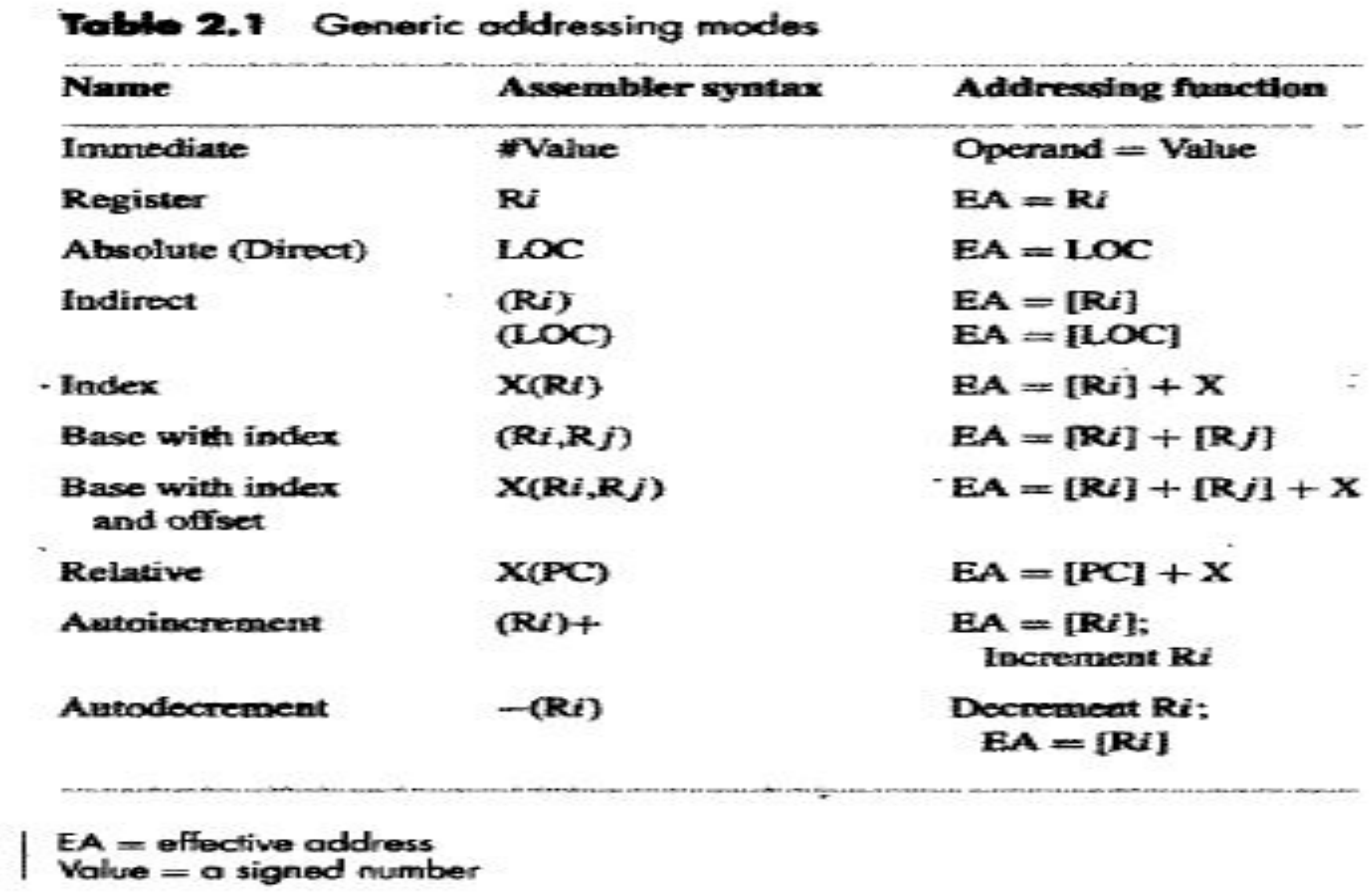


Table 2.1 Generic addressing modes

Name	Assembler syntax	Addressing function
Immediate	#Value	Operand = Value
Register	R*i*	EA = R*i*
Absolute (Direct)	LOC	EA = LOC

Indirect	(R*i*)
	(LOC)
		EA = [R*i*]
		EA = [LOC]

Index	X(R*i*)	EA = [R*i*] + X
Base with index	(R*i*,R*j*)	EA = [R*i*] + [R*j*]
Base with index and offset	X(R*i*,R*j*)	EA = [R*i*] + [R*j*] + X
Relative	X(PC)	EA = [PC] + X
Autoincrement	(R*i*)+	EA = [R*i*]; Increment R*i*
Autodecrement	−(R*i*)	Decrement R*i*; EA = [R*i*]

EA = effective address
Value = a signed number

Unit -1

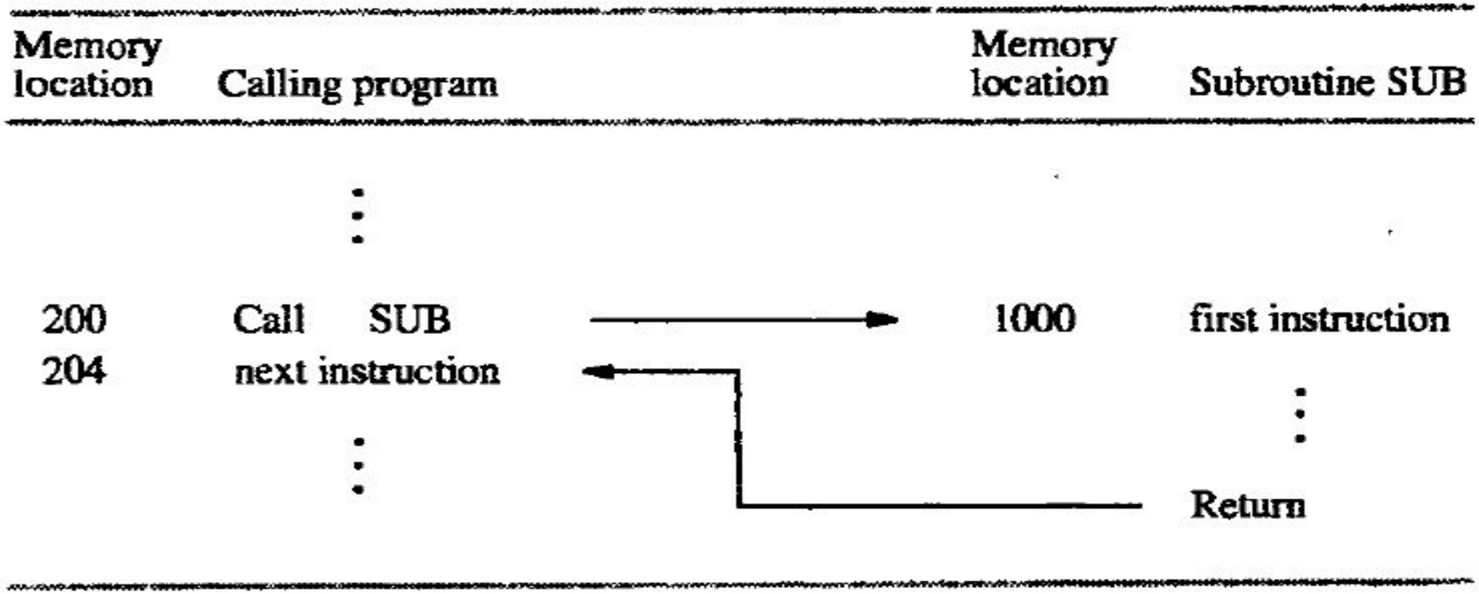
- Functional units, Bus structures, performance
- **Overflow in integer arithmetic:** Numbers, Arithmetic operations and characters
- Memory locations and addresses
- Memory operations
- Instructions and instruction sequencing
- Addressing mode
- Subroutines and use of stack frames
- Encoding of machine instructions

SUBROUTINES

- Subroutines
- Subroutine Nesting and Processor Stack
- Parameter Passing
- Stack Frame

SUBROUTINES

- A subtask consisting of a set of instructions which is executed many times is called a **subroutine**.
- The program branches to a subroutine with a Call instruction



Memory location | Calling program | Memory location | Subroutine SUB

200 Call SUB

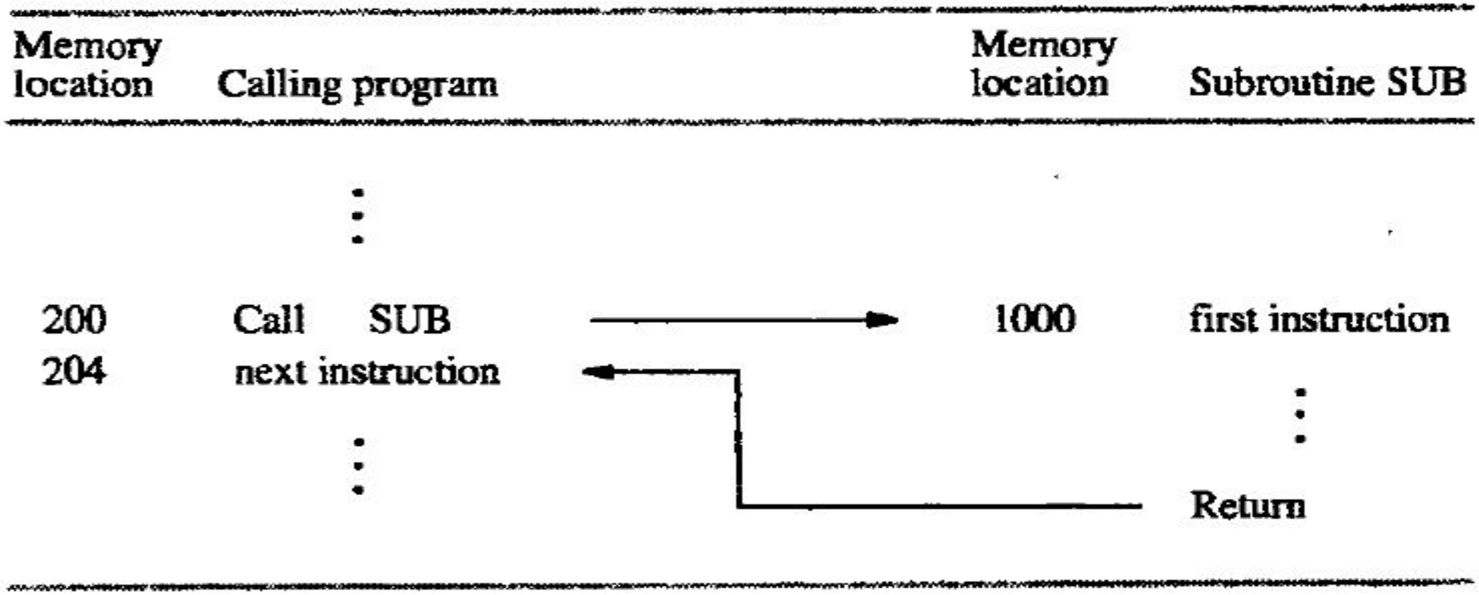
204 next instruction

1000 first instruction

Return

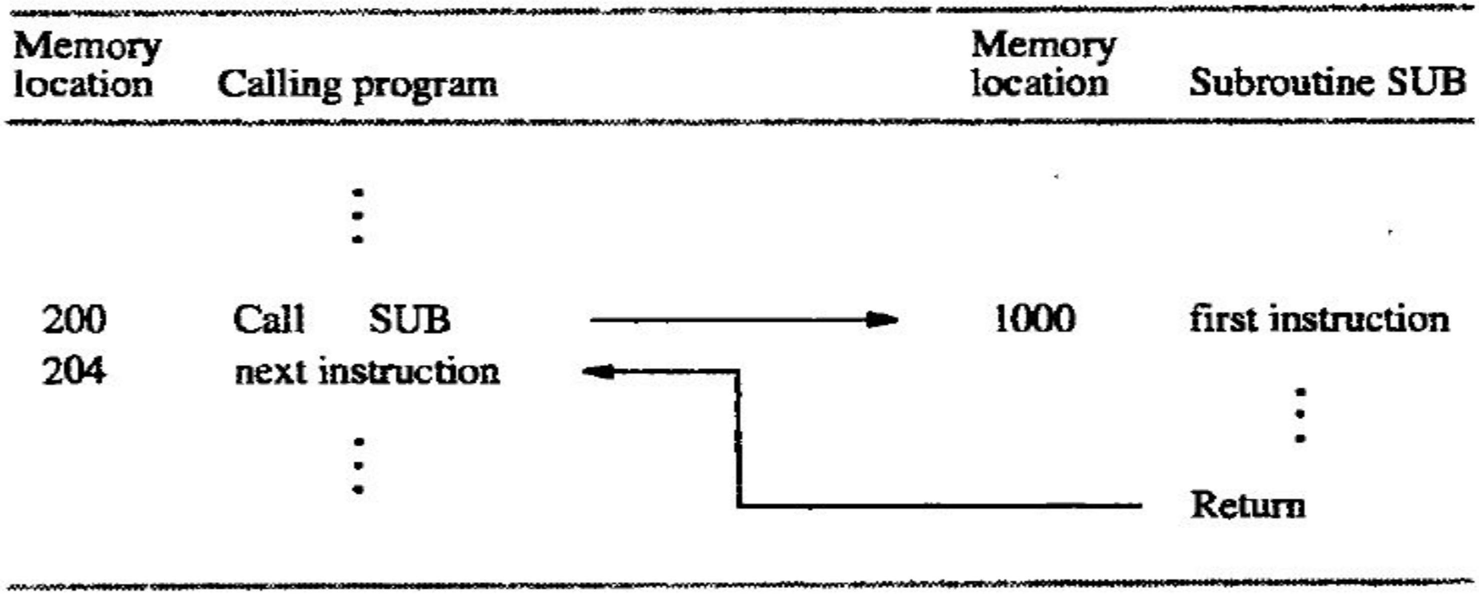
SUBROUTINES

- Once the subroutine is executed, the calling-program must resume execution starting from the instruction immediately following the Call instructions i.e. control is to be transferred back to the calling-program.
- This is done by executing a Return instruction at the end of the subroutine.



SUBROUTINES

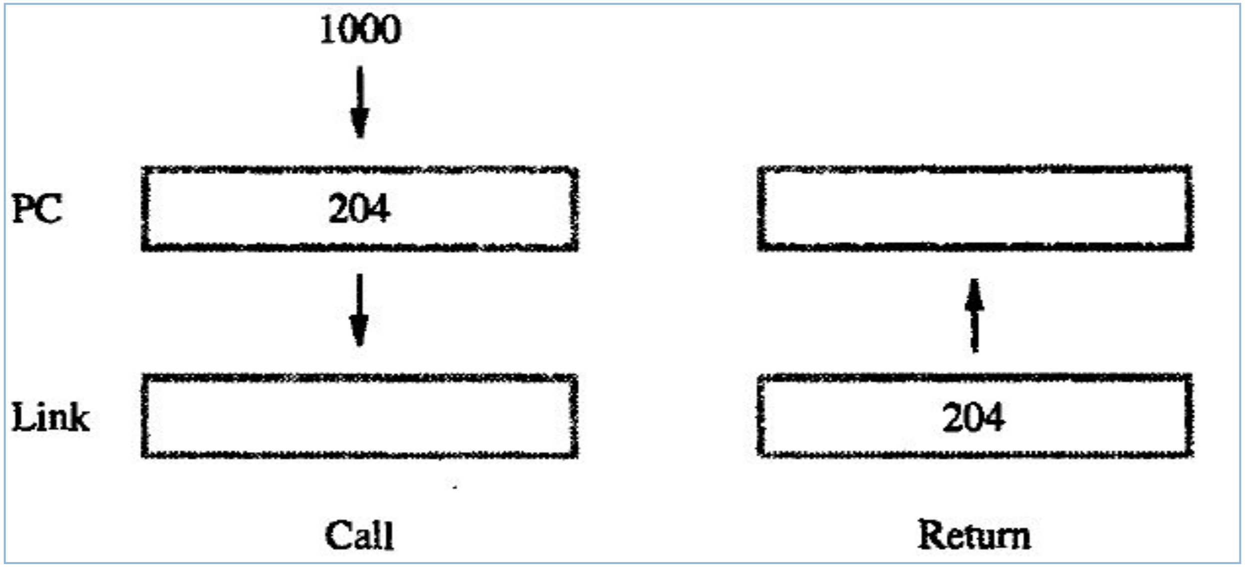
□ The way in which a computer makes it possible to call and return from subroutines is referred to as its subroutine linkage method.



Memory location	Calling program	Memory location	Subroutine SUB
	⋮		
200	Call SUB	1000	first instruction
204	next instruction		⋮
	⋮		Return

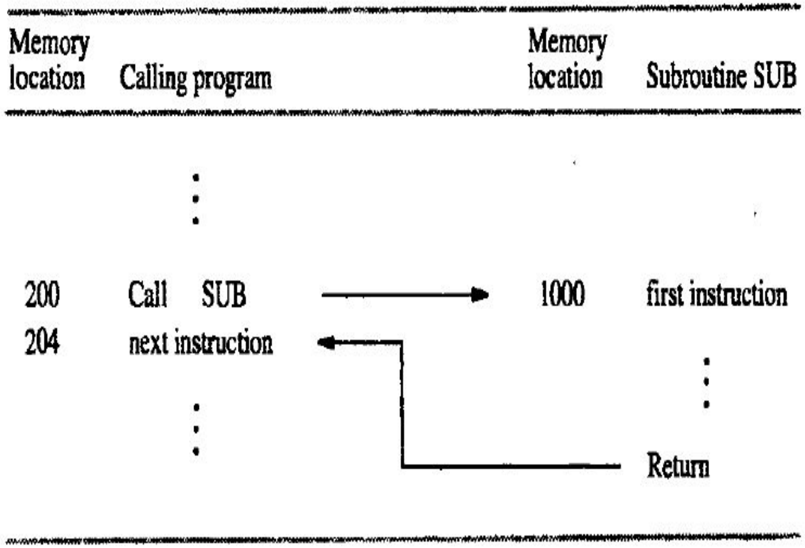
SUBROUTINES

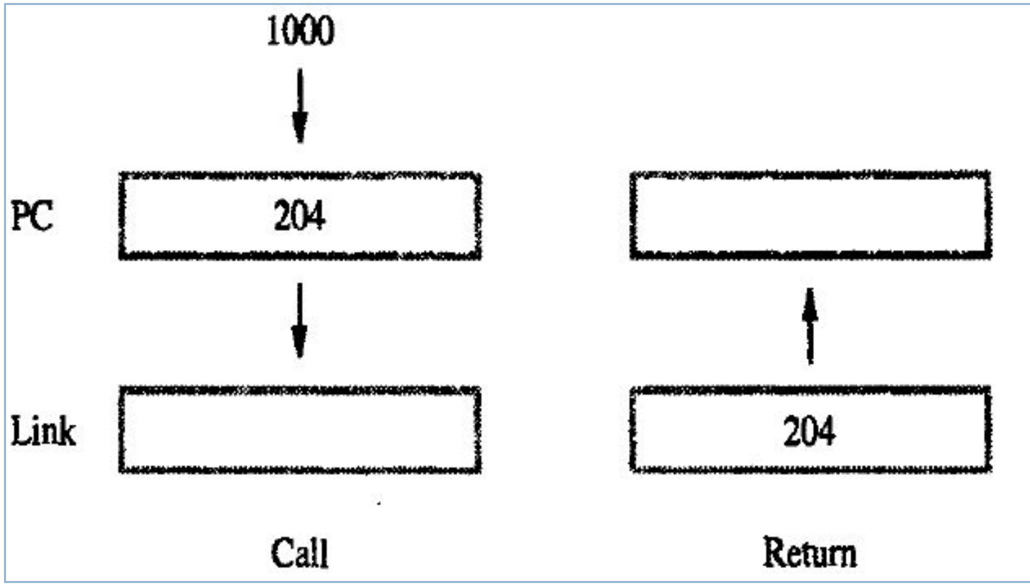
- The simplest subroutine linkage method is to save the return-address in a specific location, which may be a register dedicated to this function. Such a register is called the link register.



SUBROUTINES

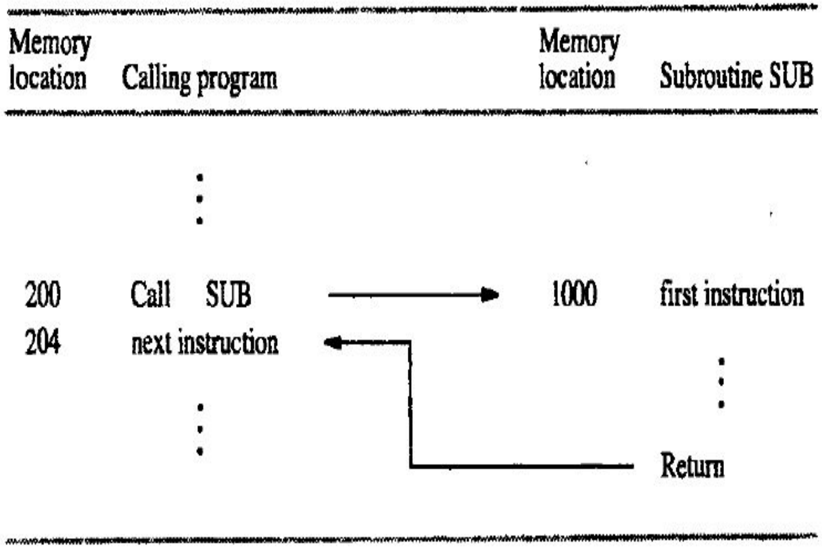
- The Call instruction is a special branch instruction that performs the following operations:
  - Store the contents of PC into link-register.
  - Branch to the target-address specified by the instruction.





SUBROUTINES

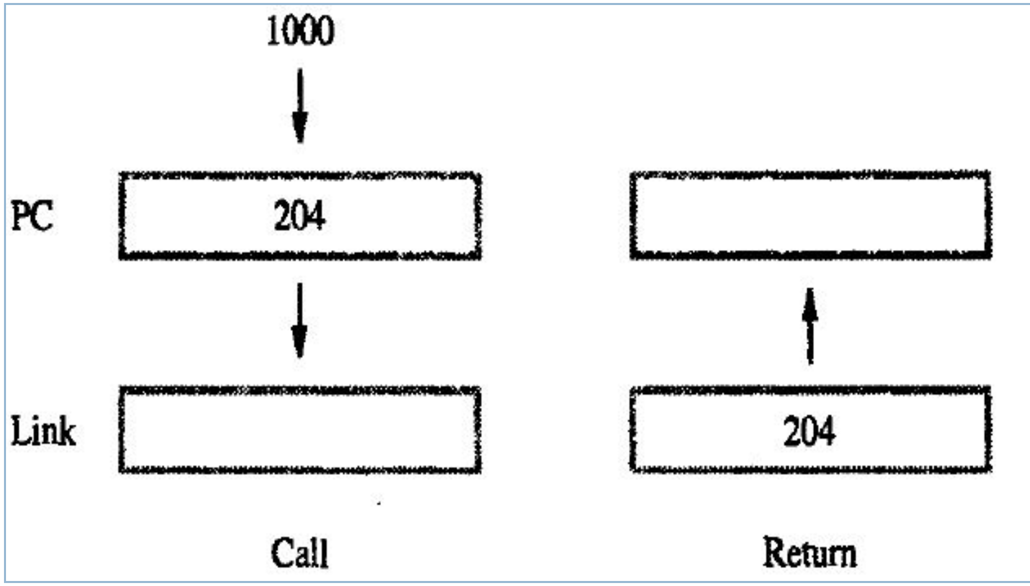
- The Return instruction is a special branch instruction that performs the operation:
  - Branch to the address contained in the link-register



Memory location | Calling program | Memory location | Subroutine SUB

200 Call SUB → 1000 first instruction

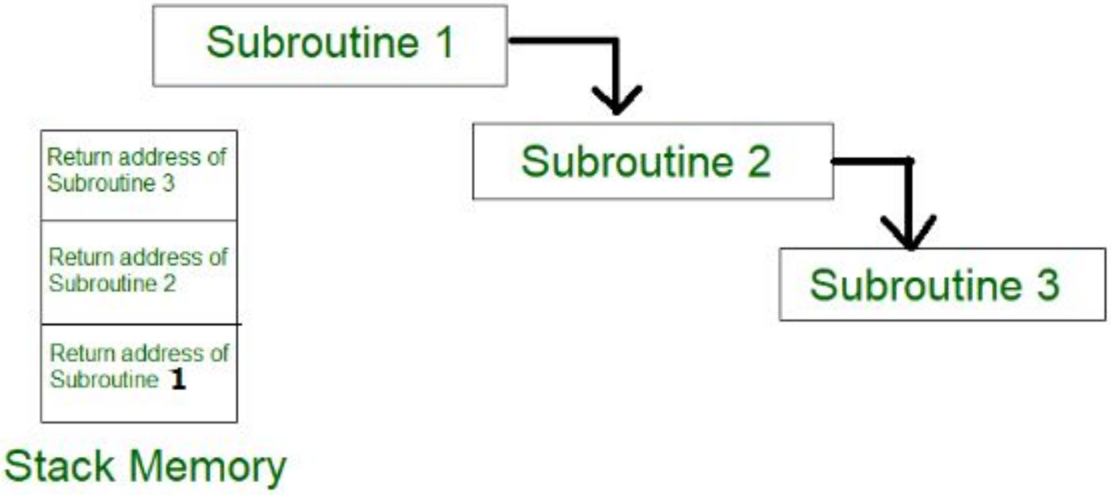
204 next instruction ← Return



SUBROUTINES

Subroutine Nesting

- This suggests that the return addresses associated with subroutine calls should be pushed onto a stack.
- Used as LIFO


Subroutine 1
Subroutine 2
Subroutine 3
Return address of Subroutine 3
Return address of Subroutine 2
Return address of Subroutine 1
Stack Memory

SUBROUTINES

Subroutine Nesting

- Processor do this operation automatically using Call instruction.
- Call instruction pushes the contents of the PC onto the processor stack and loads the subroutine address into the PC
- The Return instruction pops the return address from the processor stack into the PC

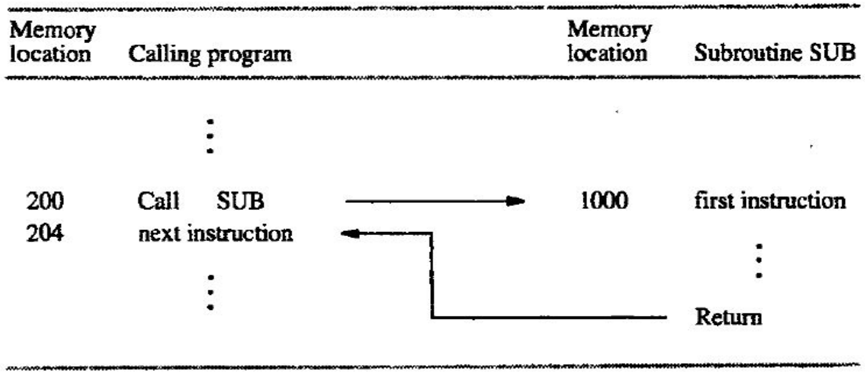
SUBROUTINES

Parameter Passing

The exchange of information between a calling program and a subroutine is referred to as parameter passing.

Parameter passing may be accomplished in several ways.

- The parameters may be placed in registers or in memory locations where they can be accessed by the subroutine.
- The parameters may be placed on the processor stack used for saving the return address.



SUBROUTINES

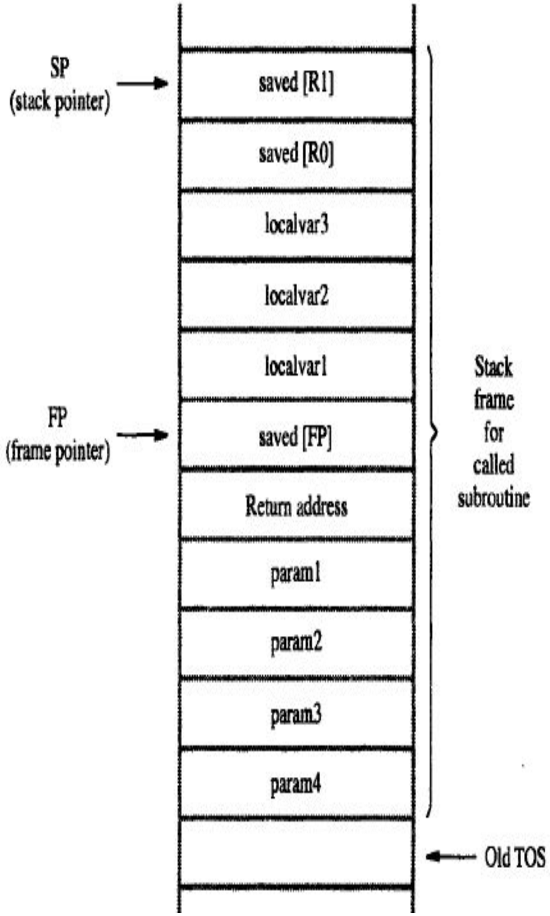
- **Pass by Value**: Values of actual parameters are copied to function's formal parameters and the two types of parameters are stored in different memory locations. So any changes made inside functions are not reflected in actual parameters of the caller.

- **Pass by reference**: Both the actual and formal parameters refer to the same locations, so any changes made inside the function are actually reflected in actual parameters of the caller.

SUBROUTINES

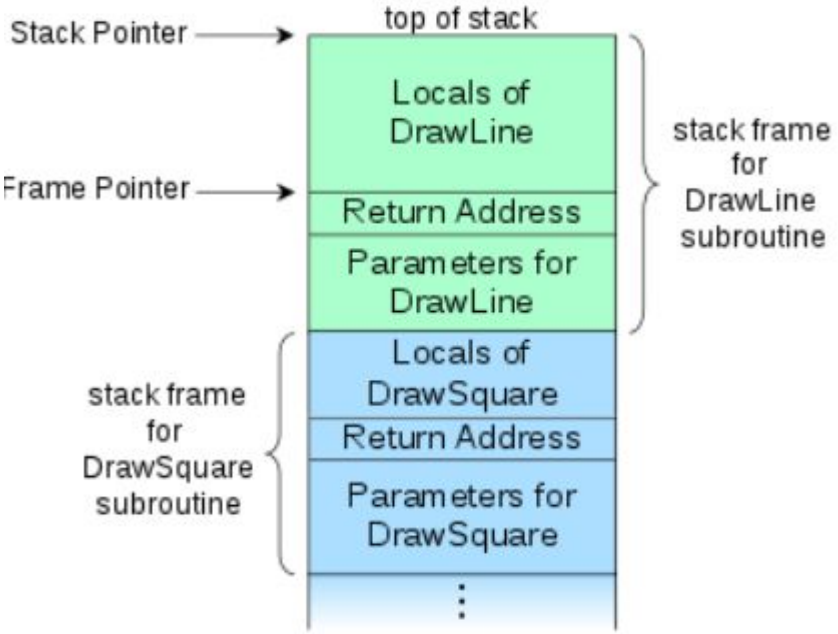
Stack Frame

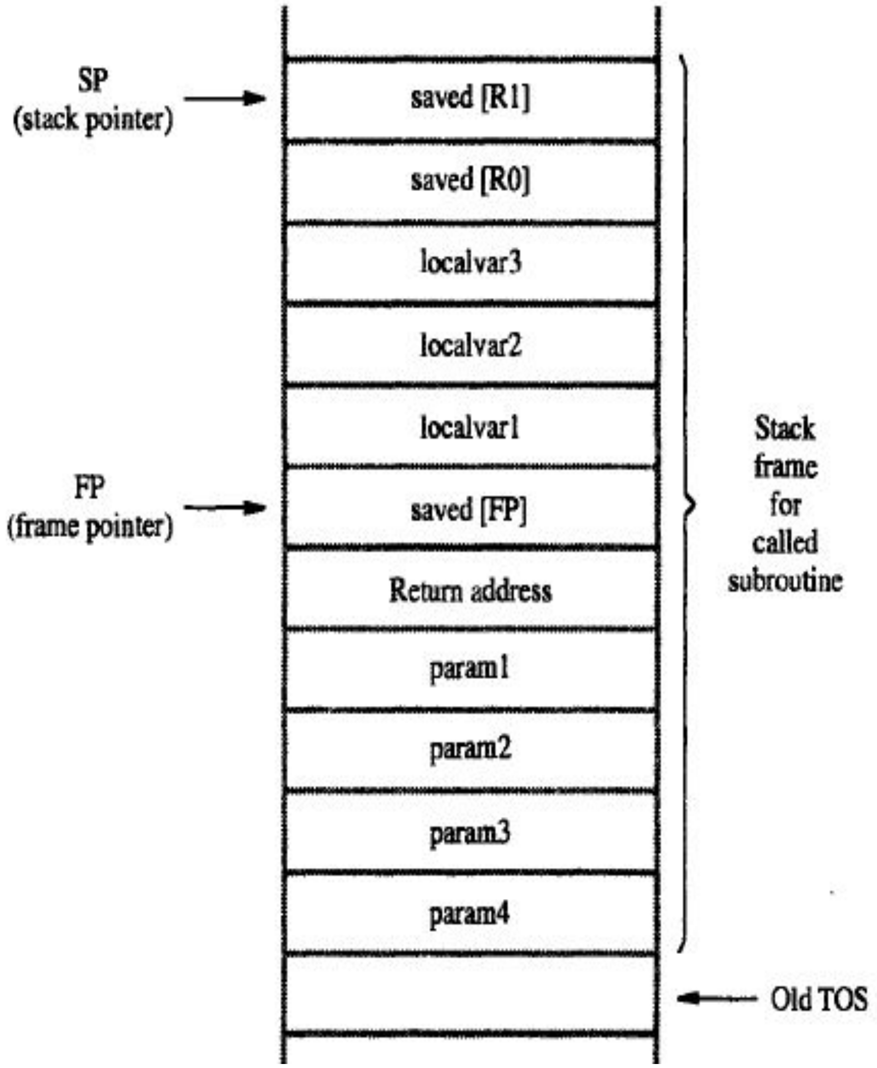
- □ During execution of the subroutine fixed locations at the top of the stack contain entries that are needed by the subroutine
- □ These locations constitute a private workspace for the time the subroutine, created at the time the subroutine is entered and freed up when the subroutine returns control to the calling program . Such space is called a stack frame



SUBROUTINES

- **Stack Pointer (SP)** points to the top of the stack
- **Frame Pointer (FP)** points to the current active frame.

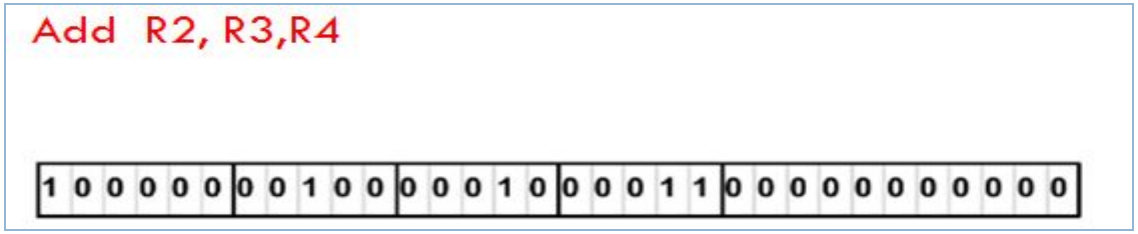




Encoding of Machine instruction

Encoding – representing instructions as binary bits

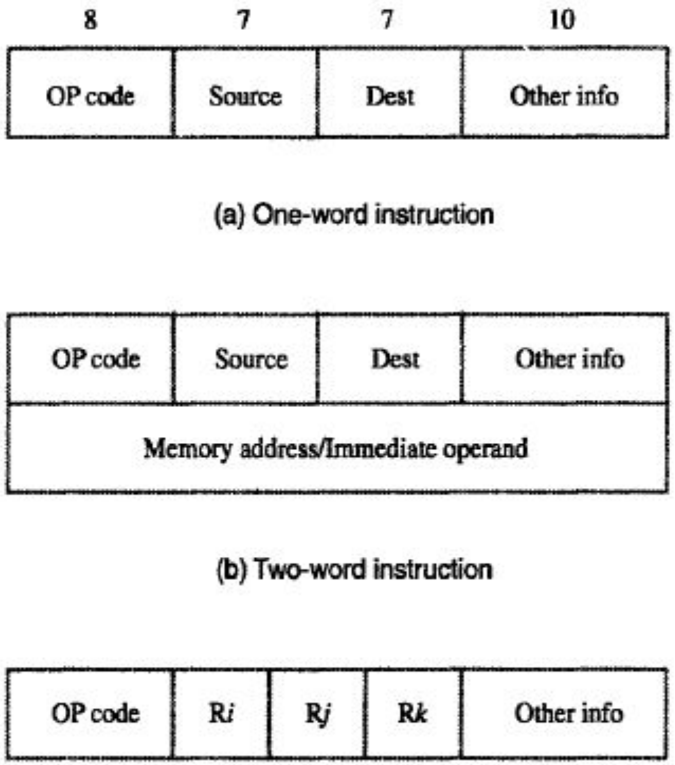
- If instruction is to be executed in a processor, an **instruction** must be **encoded** in a compact binary pattern.
- Such **encoded instructions** are properly referred to as **machine instructions stored in memory.**
- Instruction is one word (32bit) length



Add R2, R3,R4

1 0 0 0 0 0	0 0 1 0 0	0 0 0 1 0	0 0 0 1 1	0 0 0 0 0 0 0 0 0 0 0

Encoding of Machine instruction



8	7	7	10
OP code	Source	Dest	Other info

(a) One-word instruction

OP code	Source	Dest	Other info
Memory address/Immediate operand			

(b) Two-word instruction

OP code	R*i*	R*j*	R*k*	Other info

(c) Three-operand instruction

Encoding instructions into 32-bit words.

Encoding of Machine instruction

- But using multiple words, we can implement quite complex instructions, closely resembling operations in high level programming languages.

CISC(Complex instruction set computer) is used to refer to processors that use instructions set of this type.

- The restriction that an instruction must occupy only one word has led to a style of computers known as Reduced instruction set computers(RISC).