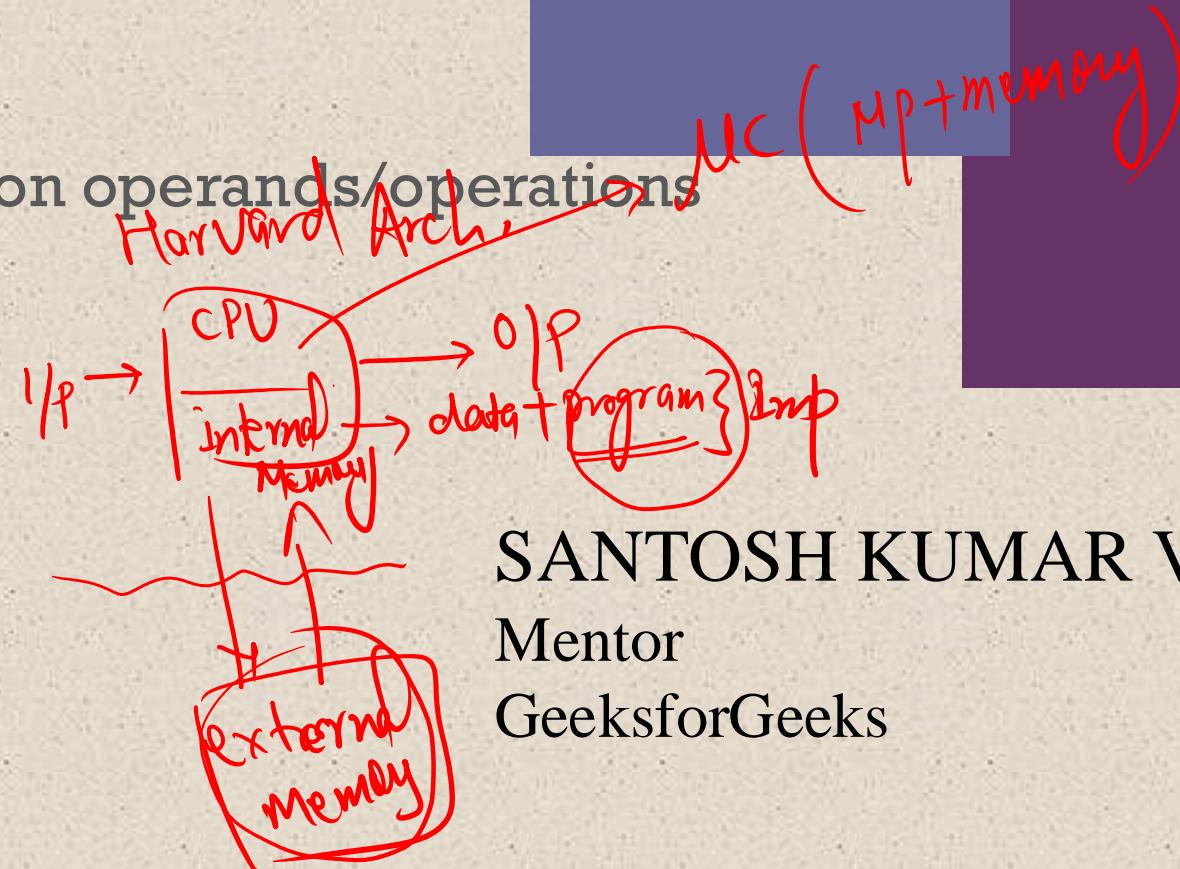
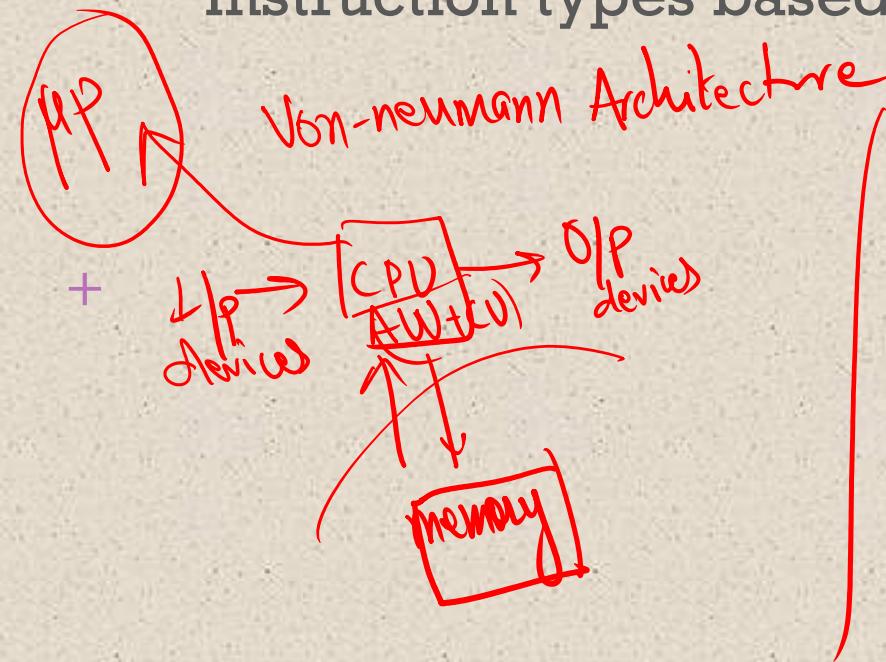


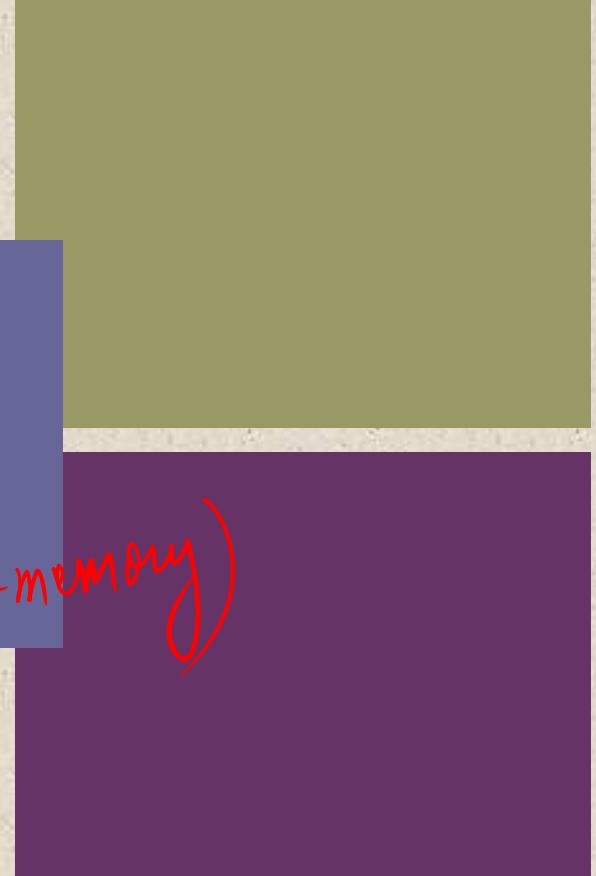
# Machine Instructions

Instruction format,

Instruction types based on operands/operations



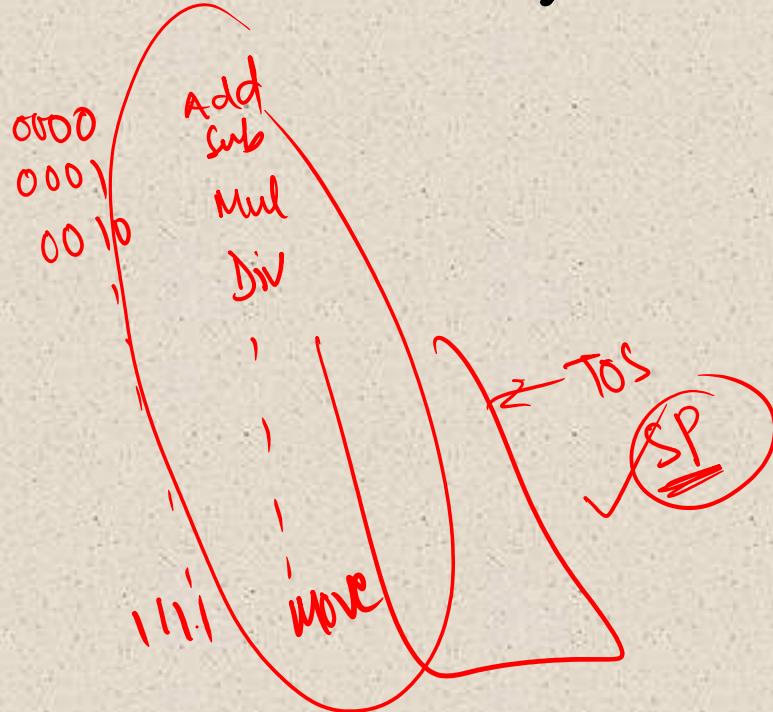
SANTOSH KUMAR VERMA  
Mentor  
GeeksforGeeks



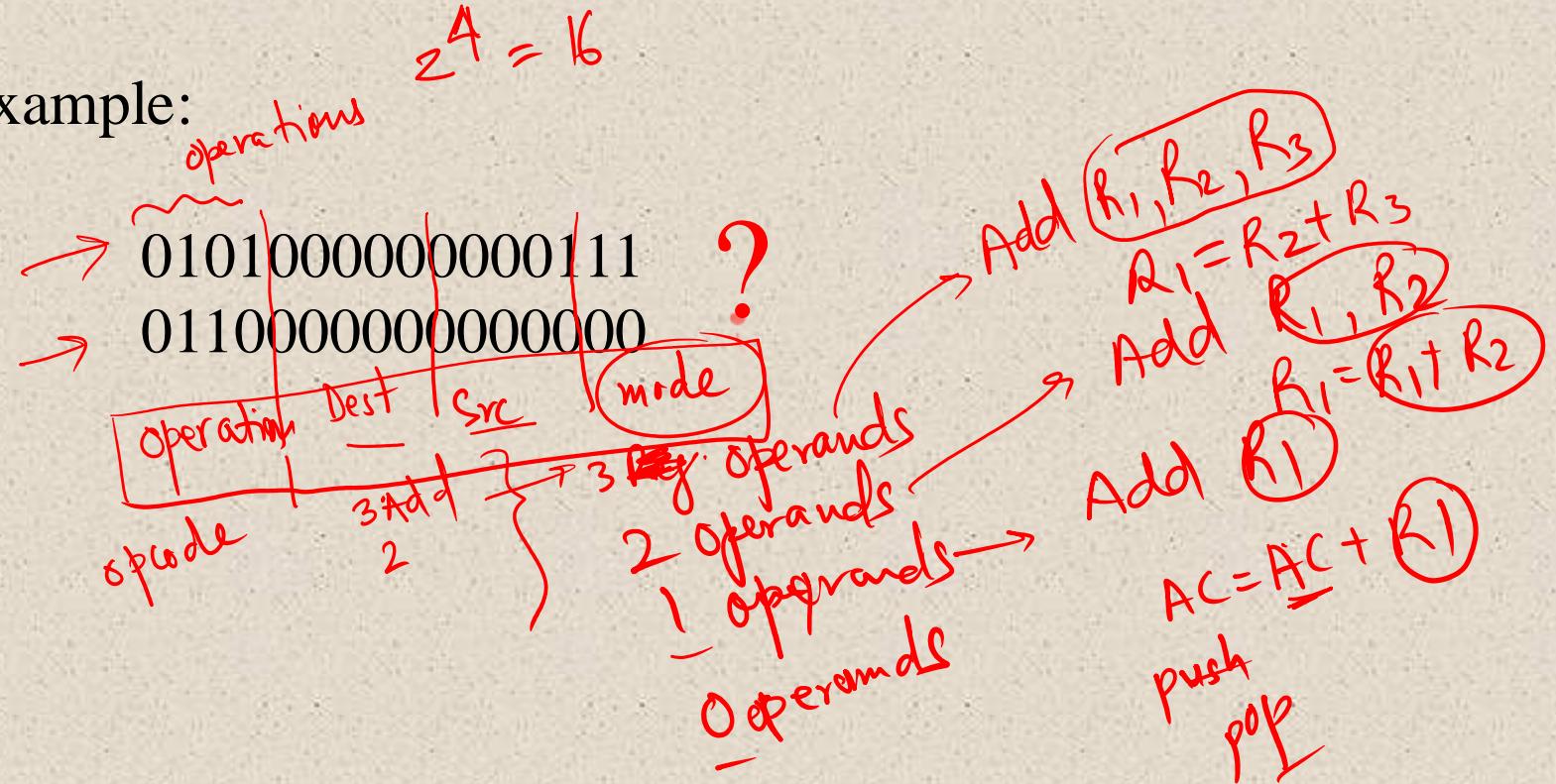
+

# Computer/Machine Instruction

A binary code used for specifying micro operations for computer.



Example:



+

# Machine Instruction Characteristics

- The operation of the processor is determined by the instructions it executes, referred to as *machine instructions* or *computer instructions*
- The collection of different instructions that the processor can execute is referred to as the processor's instruction set (ISA architecture)
- Each **instruction must contain the information required by the processor for execution**



# Elements of a Machine Instruction

## Operation code (opcode)

- Specifies the operation to be performed. The operation is specified by a binary code, known as the operation code, or *opcode*

## Source operand reference

- The operation may involve one or more source operands, that is, operands that are inputs for the operation

immediate  
Reg  
Memory | Direct  
Indirect  
Base + Index  
Relative

Instruction →

## Result operand reference

- The operation may produce a result

## Next instruction reference

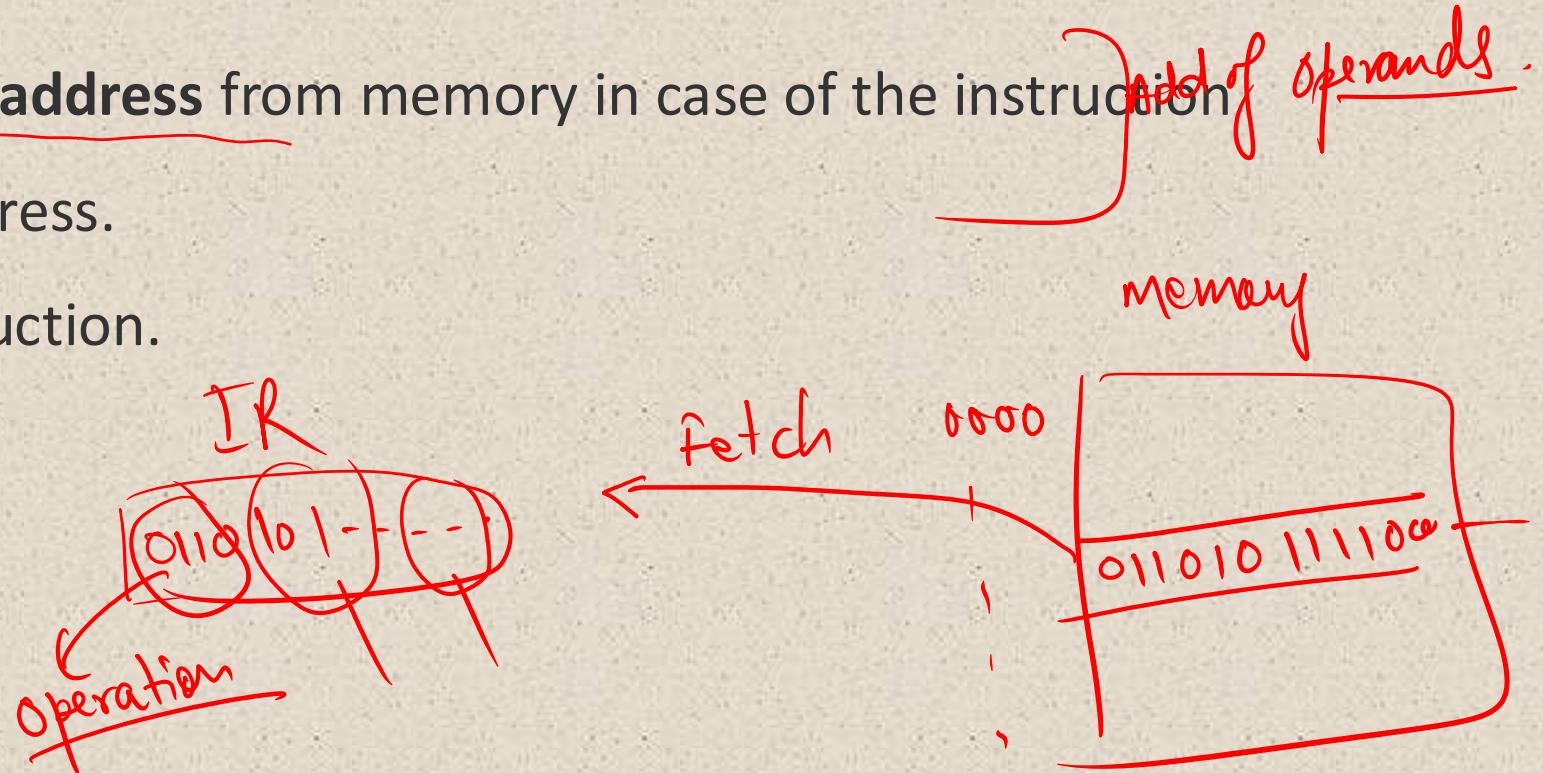
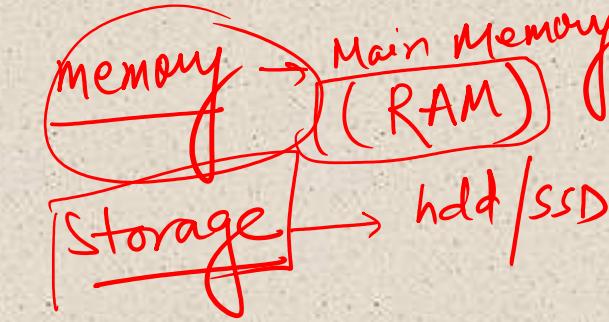
- This tells the processor where to fetch the next instruction after the execution of this instruction is complete

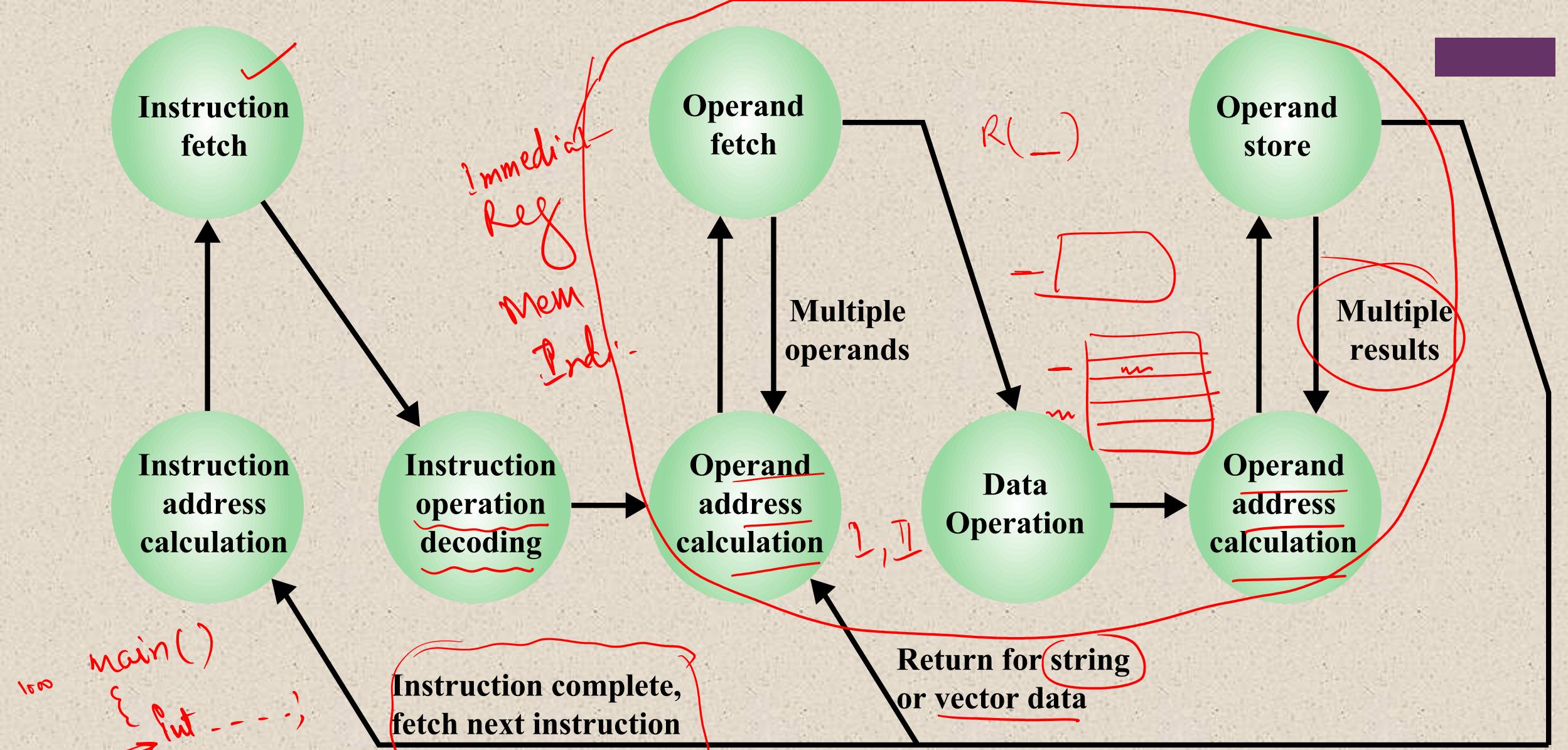
+

# Instruction Cycles

Instruction cycle consists of following phases

- **Fetching** an instruction from memory.
- **Decoding** the instruction.
- **Reading the effective address** from memory in case of the instruction having an indirect address.
- **Execution** of the instruction.





# Instruction Cycle State Diagram

# Source and result operands can be in one of four areas: (Concept of Addressing Modes)

## 1) Main or virtual memory

- As with next instruction references, the main or virtual memory address must be supplied

Direct Memory



immediate  
value  
(constant)

## 2) I/O device

- The instruction must specify the I/O module and device for the operation. If memory-mapped I/O is used, this is just another main or virtual memory address

## 3) Processor register

- A processor contains one or more registers that may be referenced by machine instructions.
- If more than one register exists each register is assigned a unique name or number and the instruction must contain the number of the desired register

8086 (A, B, C, D)  
AH AL  
PC  
SP  
BP  
SS  
CS  
DS

## 4) Immediate

- The value of the operand is contained in a field in the instruction being executed

Add R<sub>1</sub>, #123  
R<sub>1</sub> ← R<sub>1</sub> + 123

# Instruction Representation

- Within the computer each instruction is represented by a sequence of bits
- The instruction is divided into fields, corresponding to the constituent elements of the instruction

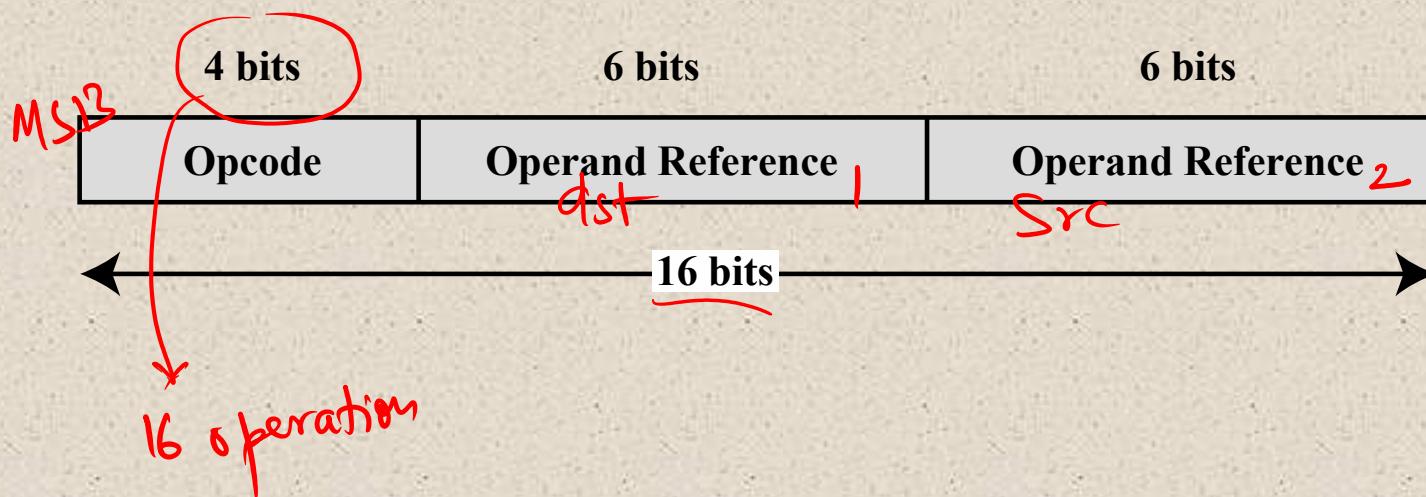


Figure 12.2 A Simple Instruction Format

+

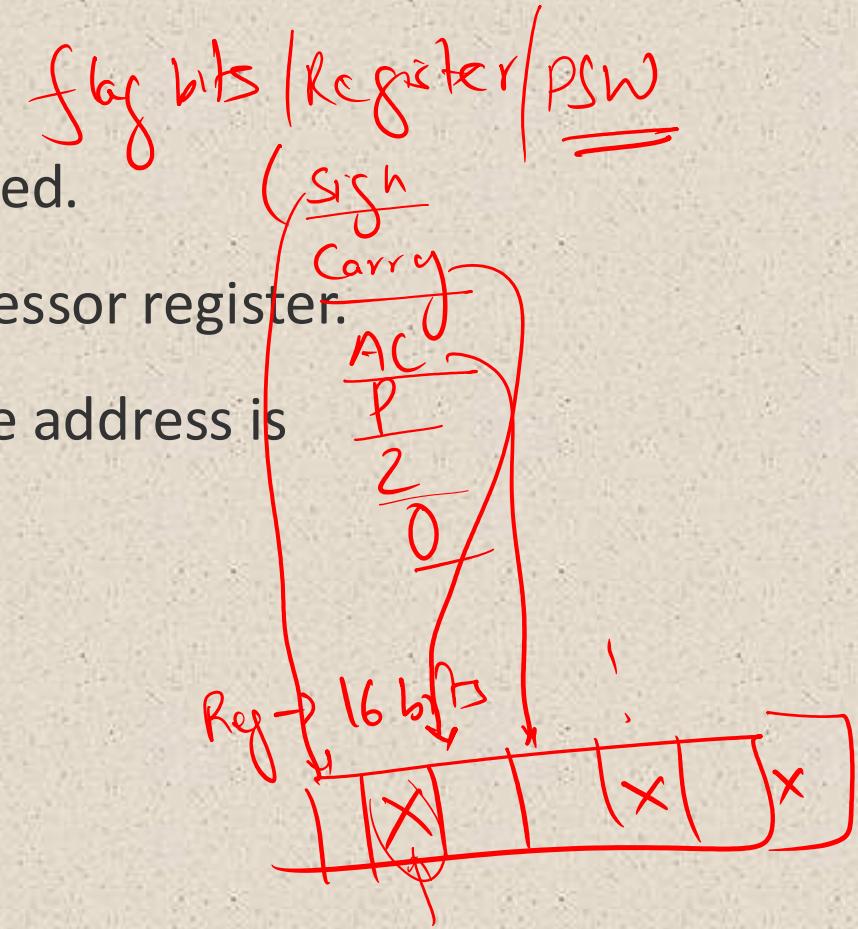
# Instruction Format

An instruction consists of bits and these bits are grouped up to make fields.

Some fields in instruction format are as follows

1. **Opcode** which tells about the operation to be performed.
2. **Address field** designating a memory address or a processor register.
3. **Mode field** specifying the way the operand or effective address is determined.

- (optional).
4. **Reserve bits.**





# Instruction Representation

- Opcodes are represented by abbreviations called mnemonics
  - Examples include:
    - ADD Add
    - SUB Subtract
    - MUL Multiply
    - DIV Divide
    - LOAD Load data from memory
    - STOR Store data to memory
  - Operands are also represented symbolically
    - Each symbolic opcode has a fixed binary representation
      - The programmer specifies the location of each symbolic operand

# Types of Instruction Format

+

## A. Three Address Instruction Format:

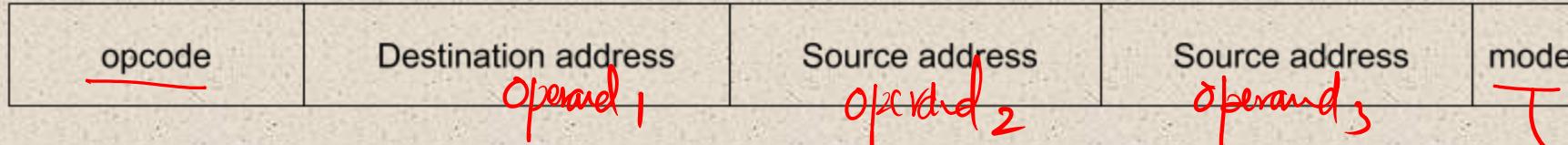


Diagram showing the breakdown of the instruction fields into bytes:

Add	Result Address	OP1 address	OP2 address
8 0000 0000	24	24 +	24

$$1GB = 1024 MB$$

$$= 1024 \times 1024 KB$$

$$= \frac{1024 \times 1024 \times 1024}{1024}$$

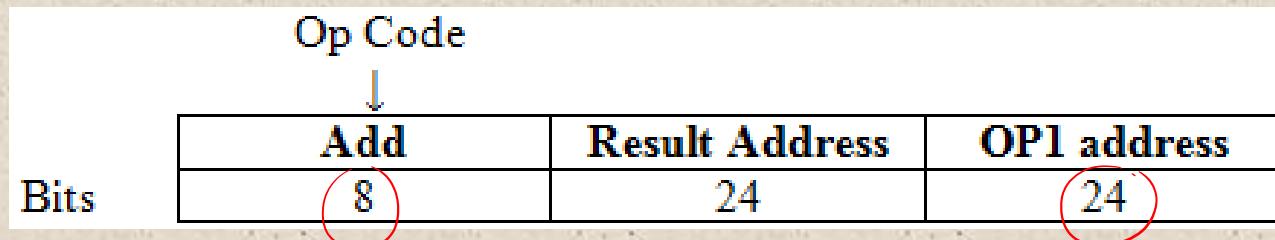
Here, the number of bytes required to encode an instruction is 10 bytes.  
Each address requires 24 bit = 3 bytes.

Since, there are three addresses and one opcode field.

Therefore  $3 \times 3 + 1 = 10$  bytes.

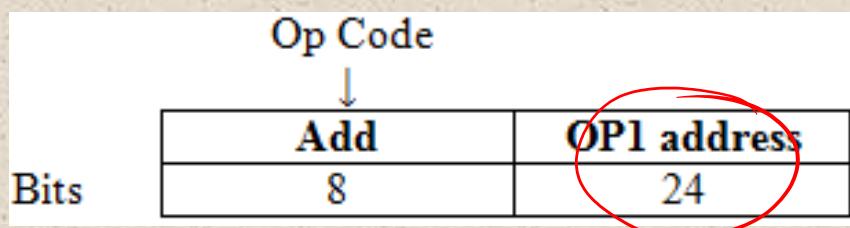
## Two Address Instruction Format:

opcode	Destination address	Source address	mode
--------	---------------------	----------------	------

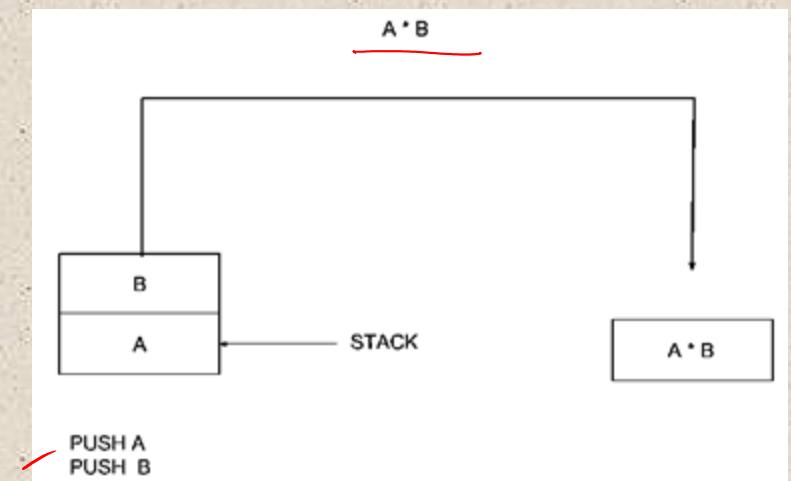


## One Address Instruction Format:

opcode	operand/address of operand	mode
--------	----------------------------	------



## Zero Address Instruction Format:



Top of stack ← Top of stack + second top of stack.

<u>Instruction</u>	<u>Comment</u>
SUB Y, A, B	$Y \leftarrow A - B$
MPY T, D, E	$T \leftarrow D \cdot E$
ADD T, T, C	$T \leftarrow T + C$
DIV Y, Y, T	$Y \leftarrow Y \div T$

(a) Three-address instructions

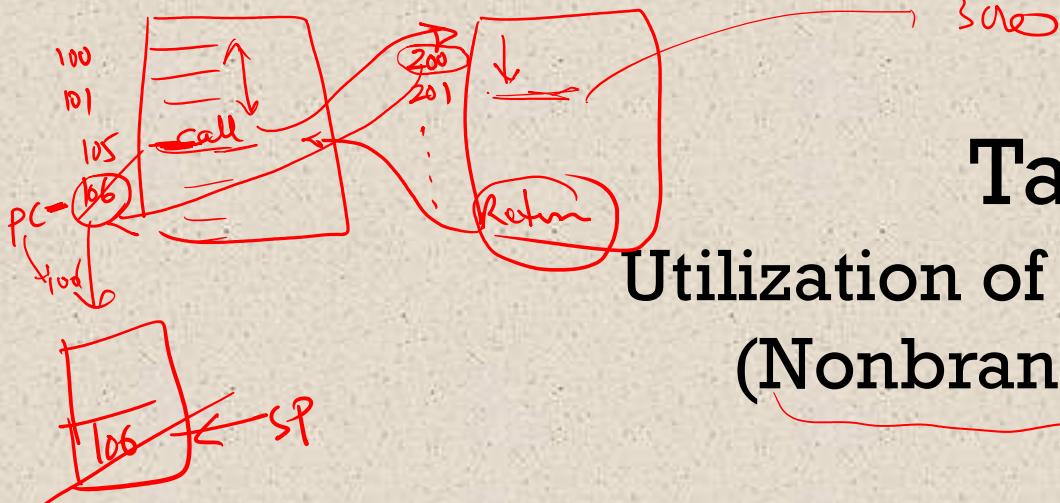
<u>Instruction</u>	<u>Comment</u>
MOVE Y, A	$Y \leftarrow A$
SUB Y, B	$Y \leftarrow Y - B$
MOVE T, D	$T \leftarrow D$
MPY T, E	$T \leftarrow T \cdot E$
ADD T, C	$T \leftarrow T + C$
DIV Y, T	$Y \leftarrow Y \div T$

(b) Two-address instructions

<u>Instruction</u>	<u>Comment</u>
LOAD D	$AC \leftarrow D$
MPY E	$AC \leftarrow AC \cdot E$
ADD C	$AC \leftarrow AC + C$
STOR Y	$Y \leftarrow AC$
LOAD A	$AC \leftarrow A$
SUB B	$AC \leftarrow AC - B$
DIV Y	$AC \leftarrow AC \div Y$
STOR Y	$Y \leftarrow AC$

(c) One-address instructions

Figure 12.3 Programs to Execute 
$$Y = \frac{A - B}{C + (D \cdot E)}$$



**Table 12.1**  
**Utilization of Instruction Addresses**  
**(Nonbranching Instructions)**

Number of Addresses	Symbolic Representation	Interpretation
3	OP A, B, C	A $\leftarrow$ B OP C
2	OP A, B	A $\leftarrow$ A OP B
1	OP A	AC $\leftarrow$ AC OP A
0	OP	T $\leftarrow$ (T - 1) OP T

AC = accumulator

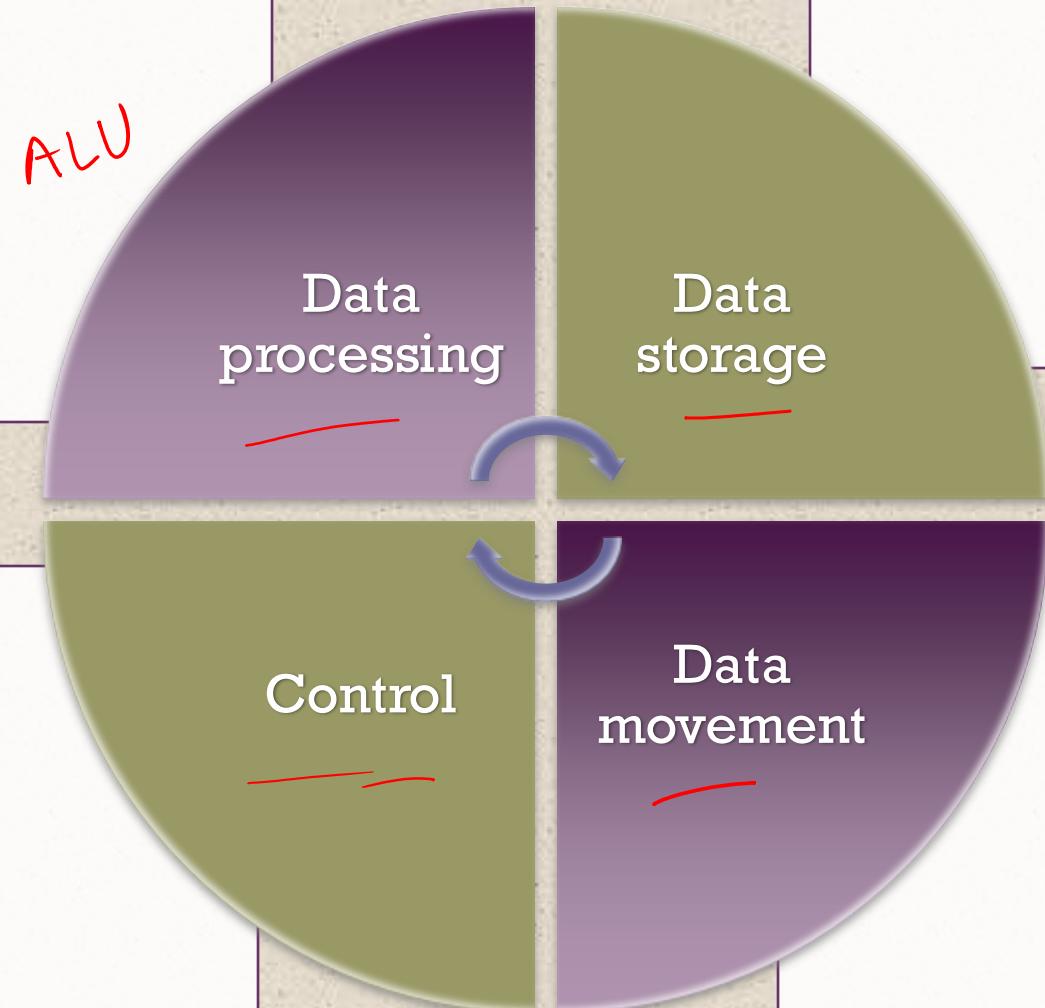
T = top of stack

(T - 1) = second element of stack

A, B, C = memory or register locations

# Instruction Types

- **Arithmetic** instructions provide computational capabilities for processing numeric data
- **Logic (Boolean)** instructions operate on the bits of a word as bits rather than as numbers, thus they provide capabilities for processing any other type of data the user may wish to employ



- **Test instructions** are used to test the value of a data word or the status of a computation
- **Branch** instructions are used to branch to a different set of instructions depending on the decision made

- **Movement of data** into or out of register and or memory locations

- **I/O instructions** are needed to transfer programs and data into memory and the results of computations back out to the user

# Instruction Set Design

Very complex because it affects so many aspects of the computer system

Defines many of the functions performed by the processor

Programmer's means of controlling the processor

Fundamental design issues:

## Operation repertoire

- How many and which operations to provide and how complex operations should be

## Data types

- The various types of data upon which operations are performed

## Instruction format

- Instruction length in bits, number of addresses, size of various fields, etc.

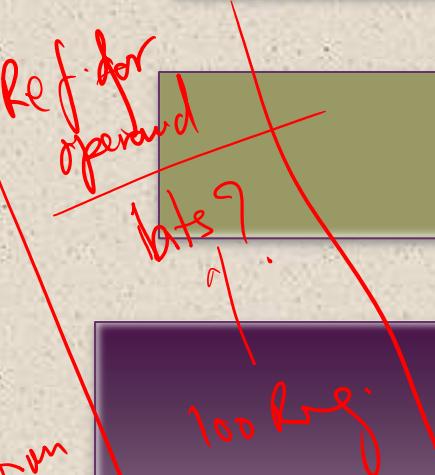
100 Regs in CPU

## Registers

- Number of processor registers that can be referenced by instructions and their use

## Addressing

- The mode or modes by which the address of an operand is specified



Type	Operation Name	Description
Data Transfer	Move (transfer)	Transfer word or block from source to destination
	Store	Transfer word from processor to memory
	Load (fetch)	Transfer word from memory to processor <i>Register</i>
	Exchange	Swap contents of source and destination
	Clear (reset)	Transfer word of 0s to destination
	Set	Transfer word of 1s to destination
	Push	Transfer word from source to top of stack
	Pop	Transfer word from top of stack to destination
	Add	Compute sum of two operands
Arithmetic	Subtract	Compute difference of two operands
	Multiply	Compute product of two operands
	Divide	Compute quotient of two operands
	Absolute	Replace operand by its absolute value
	Negate	Change sign of operand
	Increment	Add 1 to operand
	Decrement	Subtract 1 from operand
	AND	Perform logical AND
	OR	Perform logical OR
Logical	NOT (complement)	Perform logical NOT
	Exclusive-OR	Perform logical XOR
	Test	Test specified condition; set flag(s) based on outcome
	Compare	Make logical or arithmetic comparison of two or more operands; set flag(s) based on outcome
	Set Control Variables	Class of instructions to set controls for protection purposes, interrupt handling, timer control, etc.
	Shift	Left (right) shift operand, introducing constants at end
	Rotate	Left (right) shift operand, with wraparound end

MOVE

MOV

MVI

## Common Instruction Set Operations (page 1 of 2)

CMP R<sub>1</sub>, R<sub>2</sub>

R<sub>1</sub> - R<sub>2</sub>

CF, SF, ZF

1 0 0 X  
1 1 0 → R<sub>1</sub> < R<sub>2</sub>

0 1 0 X  
0 0 0 R<sub>1</sub> > R<sub>2</sub>

# Common Instruction Set Operations (page 2 of 2)

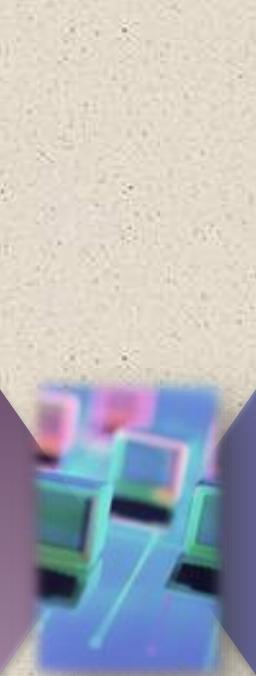
Type	Operation Name	Description
Transfer of Control	Jump (branch) <i>goto</i>	Unconditional transfer; load PC with specified address
	Jump <u>Conditional</u>	Test specified condition; either load PC with specified address or do nothing, based on condition
	Jump to Subroutine	Place current program control information in known location; jump to specified address
	Return	Replace contents of PC and other register from known location
	Execute	Fetch operand from specified location and execute as instruction; <u>do not modify PC</u>
	Skip	Increment PC to skip next instruction
	Skip <u>Conditional</u>	Test specified condition; either skip or do nothing based on condition
	Halt	Stop program execution
	Wait (hold)	Stop program execution; test specified condition repeatedly; resume execution when condition is satisfied
	No operation	No operation is performed, but program execution is continued
Input/Output	<u>Input (read)</u>	Transfer data from specified I/O port or device to destination (e.g., main memory or processor register)
	<u>Output (write)</u>	Transfer data from specified source to I/O port or device
	Start I/O	Transfer instructions to <u>I/O processor</u> to initiate I/O operation <i>DMA</i>
	Test I/O	Transfer status information from I/O system to specified destination
Conversion	Translate <i>decimal BCD</i>	Translate values in a section of memory based on a table of correspondences
	Convert	Convert the contents of a word from one form to another (e.g., packed decimal to binary)

XLAT

# Processor Actions for Various Types of Operations

<p><b>Data Transfer</b></p> <p>SV Gec ALE M/IO WR RD Int.</p>	<p>Transfer data from one location to another</p> <p>If memory is involved:</p> <ul style="list-style-type: none"> <li>Determine memory address</li> <li>Perform virtual-to-actual-memory address transformation</li> <li>Check cache</li> <li>Initiate memory read/write</li> </ul> <p>24 bit</p>
<p><b>Arithmetic</b></p>	<p>May involve data transfer, before and/or after</p> <p>Perform function in ALU</p> <p>Set condition codes and flags</p>
<p><b>Logical</b></p>	<p>Same as arithmetic</p>
<p><b>Conversion</b></p>	<p>Similar to arithmetic and logical. May involve special logic to perform conversion</p>
<p><b>Transfer of Control</b></p>	<p>Update program counter. For subroutine call/return, manage parameter passing and linkage</p>
<p><b>I/O</b></p>	<p>Issue command to I/O module</p> <p>If memory-mapped I/O, determine memory-mapped address</p> <p>I/O mapped I/O</p>

# Data Transfer



Most fundamental type of machine instruction

Must specify:

- Location of the source and destination operands
- The length of data to be transferred must be indicated
- The mode of addressing for each operand must be specified

# Examples of IBM EAS/390 Data Transfer Operations

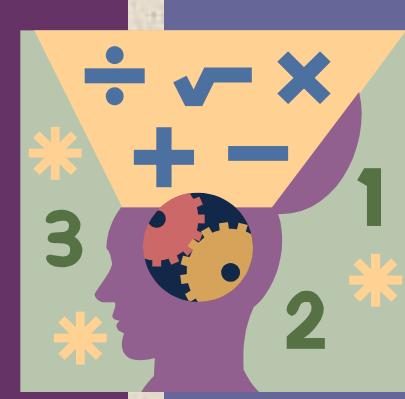
Operation Mnemonic	Name	Number of Bits Transferred	Description
L	Load	32	Transfer from memory to register
LH	Load Halfword	16	Transfer from memory to register
LR	Load	32	Transfer from register to register
LER	Load (Short)	32	Transfer from floating-point register to floating-point register
LE	Load (Short)	32	Transfer from memory to floating-point register
LDR	Load (Long)	64	Transfer from floating-point register to floating-point register
LD	Load (Long)	64	Transfer from memory to floating-point register
ST	Store	32	Transfer from register to memory
STH	Store Halfword	16	Transfer from register to memory
STC	Store Character	8	Transfer from register to memory
STE	Store (Short)	32	Transfer from floating-point register to memory
STD	Store (Long)	64	Transfer from floating-point register to memory

Load

Store



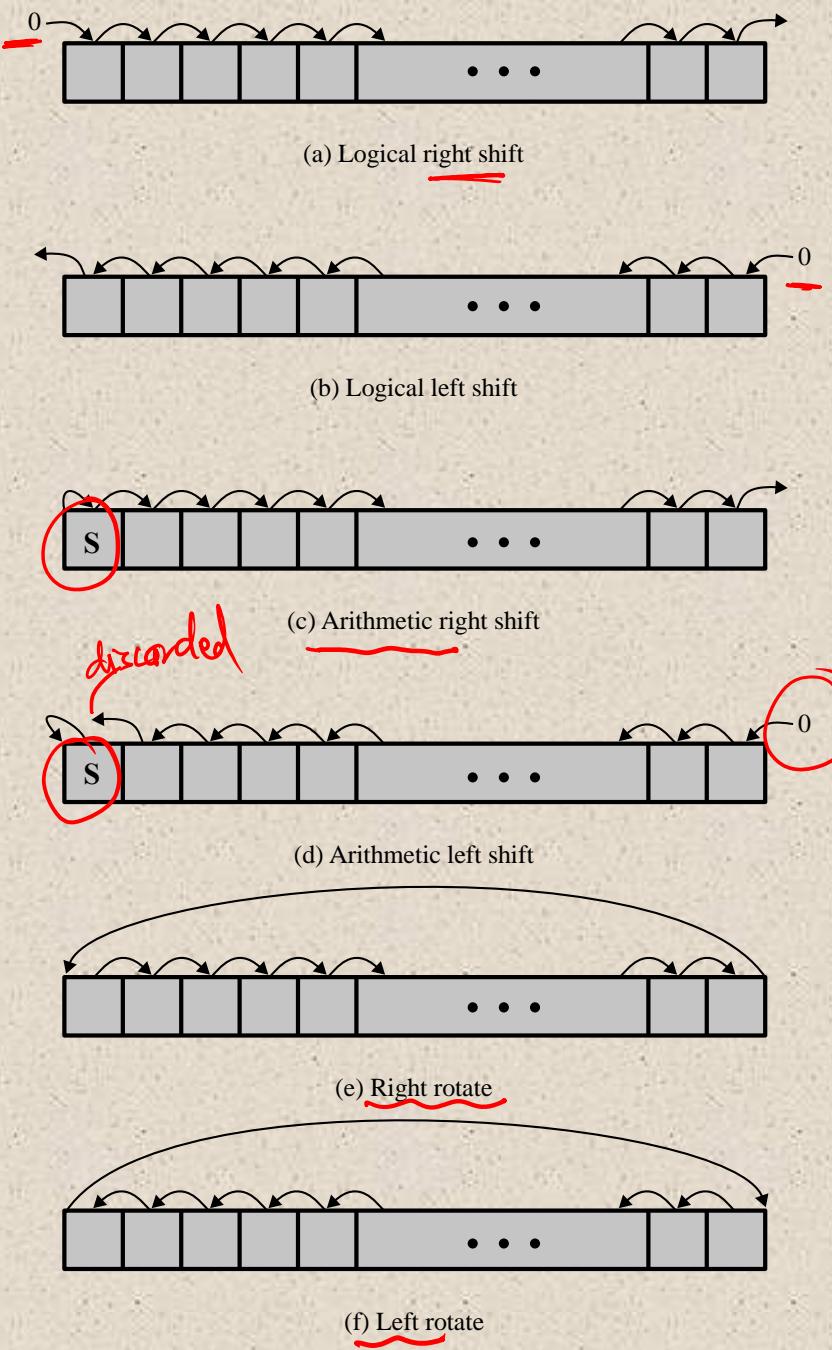
- Most machines provide the basic arithmetic operations of add, subtract, multiply, and divide
- These are provided for signed integer (fixed-point) numbers
- Often they are also provided for floating-point and packed decimal numbers
- Other possible operations include a variety of single-operand instructions:
  - Absolute
    - Take the absolute value of the operand
  - Negate
    - Negate the operand
  - Increment
    - Add 1 to the operand
  - Decrement
    - Subtract 1 from the operand



## Arithmetic

# Basic Logical Operations

P	Q	NOT P	P AND Q	P OR Q	P XOR Q	P=Q
0	0	1	0	0	0	1
0	1	1	0	1	1	0
1	0	0	0	1	1	0
1	1	0	1	1	0	1



$R_1 \gg 2$   
 1001  
 1st  $\rightarrow 0100$   
 2nd  $\rightarrow \underline{0}010$

Figure 12.6 Shift and Rotate Operations

# Examples of Shift and Rotate Operations

Input	Operation	Result
10100 <u>110</u>	Logical right shift ( <u>3 bits</u> )	<u>00010100</u>
<u>10100110</u>	Logical left shift (3 bits)	0011 <u>0000</u>
10100110	Arithmetic right shift (3 bits)	0 <u>1110100</u>
<u>10100110</u>	Arithmetic left shift (3 bits)	0 <u>10110000</u>
10100110	Right rotate (3 bits)	11010100
10100110	Left rotate (3 bits)	00110101



Instructions  
that change  
the format or  
operate on  
the format of  
data

An example  
is converting  
from decimal  
to binary

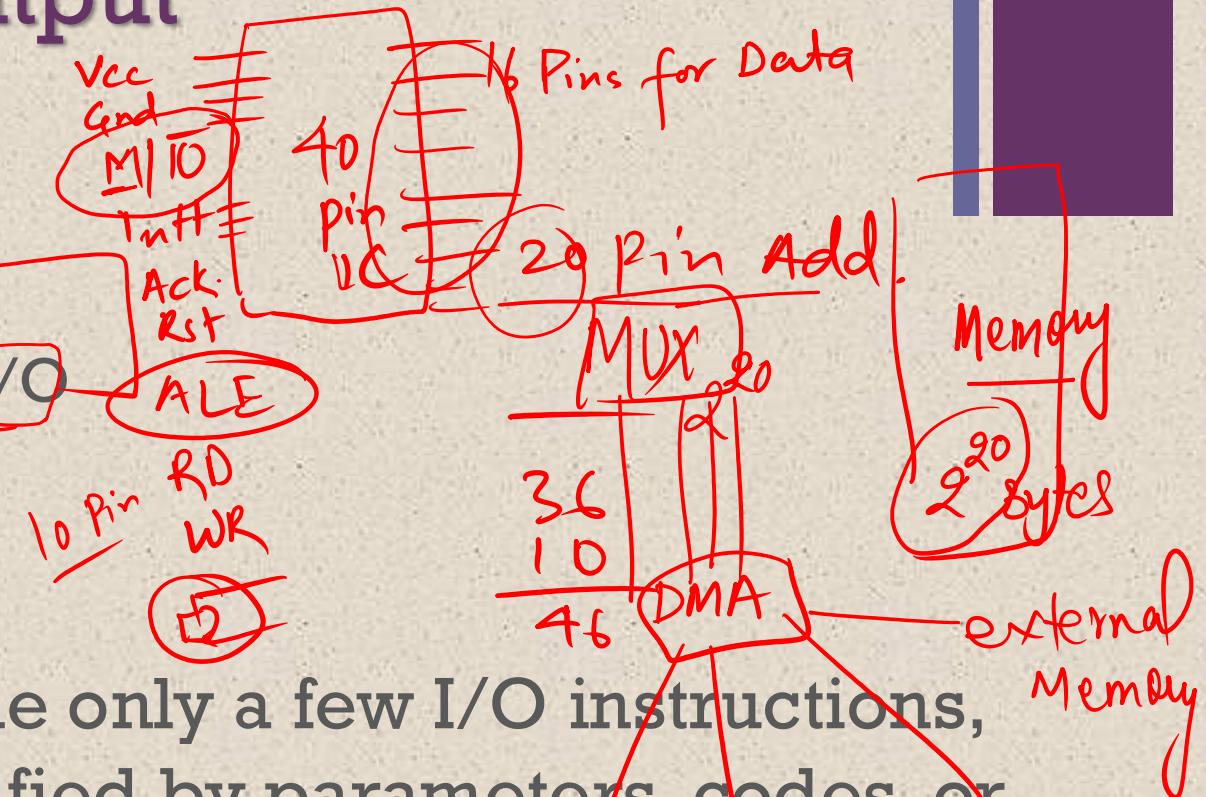
## Conversion

An example of a  
more complex  
editing  
instruction is the  
EAS/390  
Translate (TR)  
instruction

# Input/Output

## Variety of approaches taken:

- Isolated programmed I/O
- Memory-mapped programmed I/O
- DMA
- Use of an I/O processor



■ Many implementations provide only a few I/O instructions, with the specific actions specified by parameters, codes, or command words

$$\begin{array}{c} \text{1 Pin} \\ \hline 0 \mid 1 \\ \text{2} \end{array} \quad \begin{array}{c} \text{2 Pin} \\ \hline 2 \mid 2 \\ \text{2} \end{array}$$



# System Control

Instructions that can be executed only while the processor is in a certain privileged state or is executing a program in a special privileged area of memory

Typically these instructions are reserved for the use of the operating system

Examples of system control operations:

A system control instruction may read or alter a control register

An instruction to read or modify a storage protection key

Access to process control blocks in a multiprogramming system

## + Transfer of Control

### ■ Reasons why transfer-of-control operations are required:

- It is essential to be able to execute each instruction more than once
- Virtually all programs involve some decision making
- It helps if there are mechanisms for breaking the task up into smaller pieces that can be worked on one at a time

### ■ Most common transfer-of-control operations found in instruction sets:

- Branch
- Skip
- Procedure call

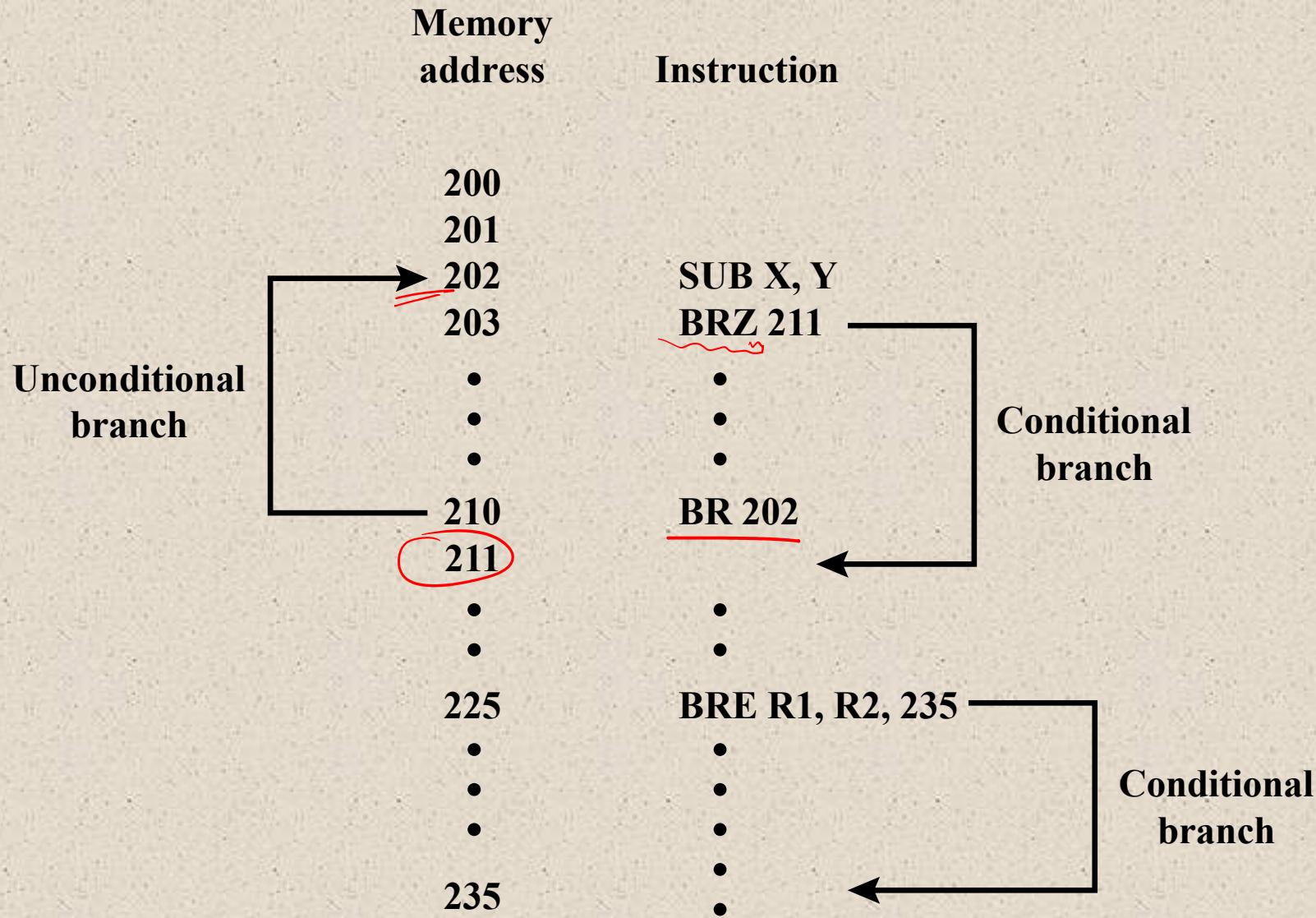
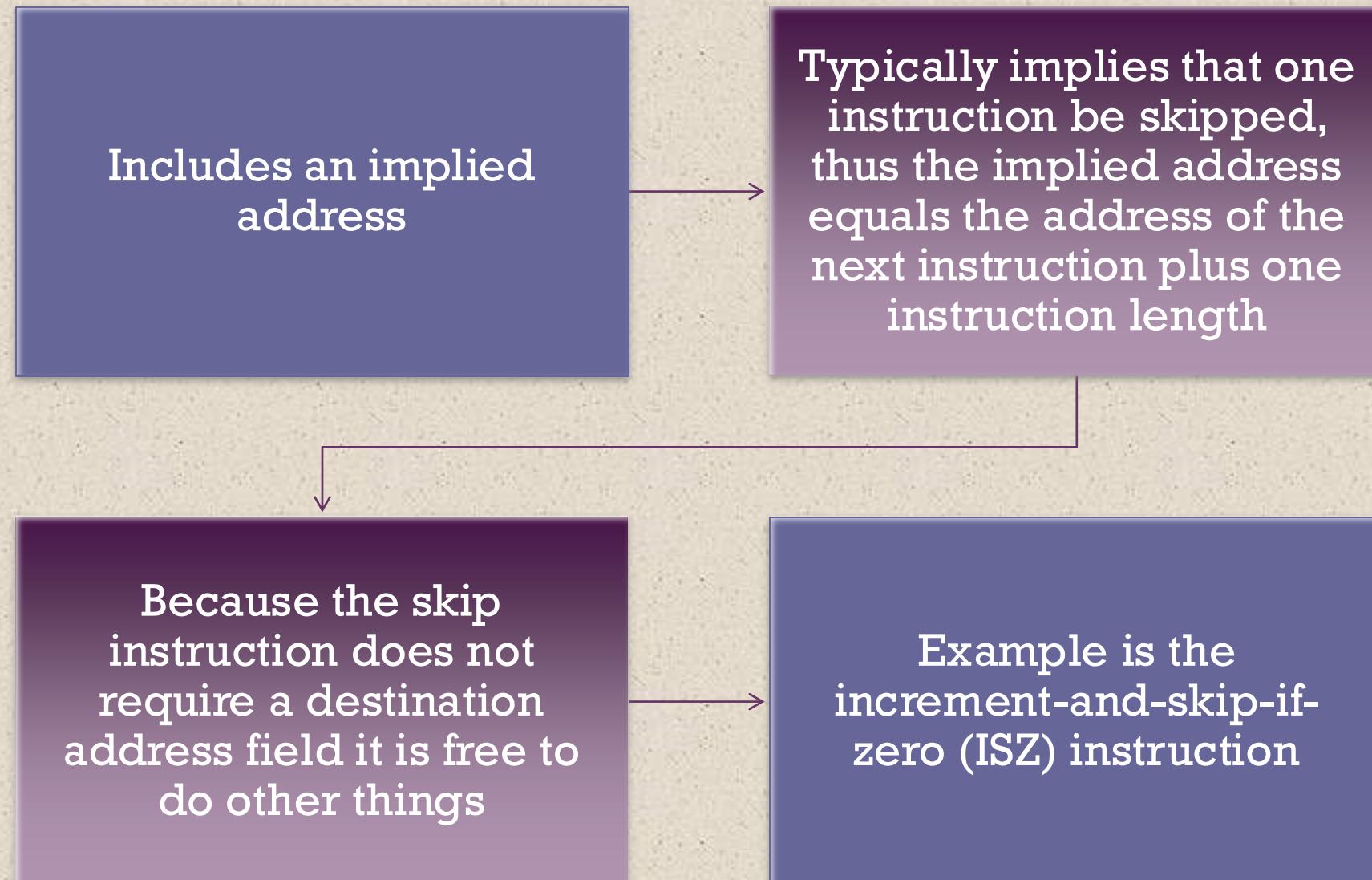


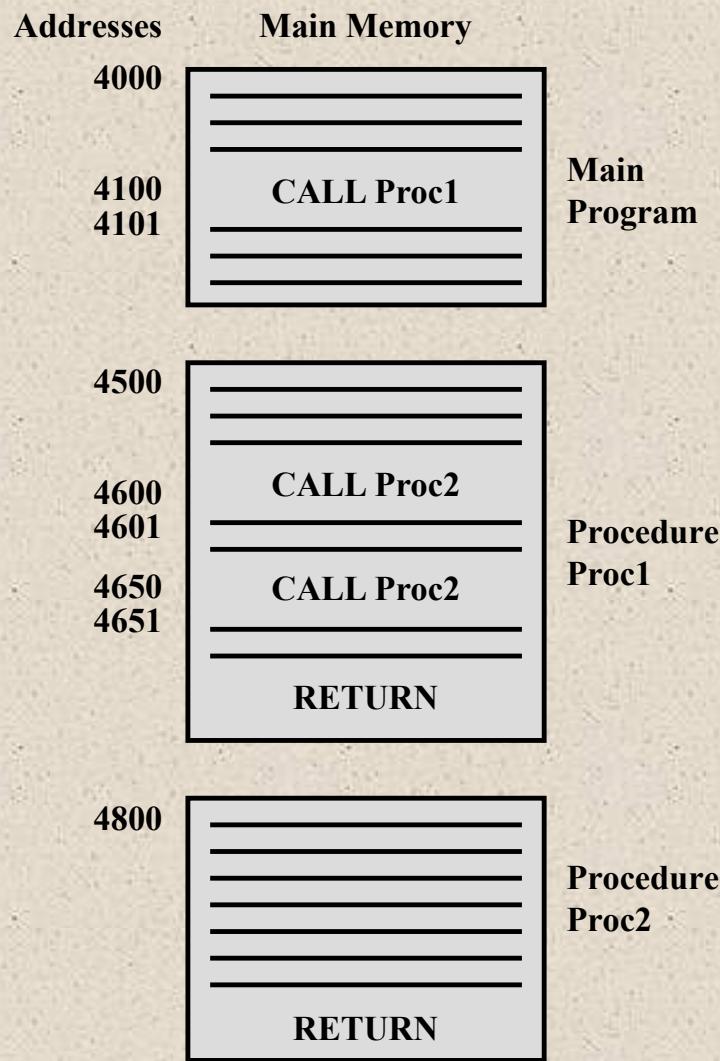
Figure 12.7 Branch Instructions

# Skip Instructions

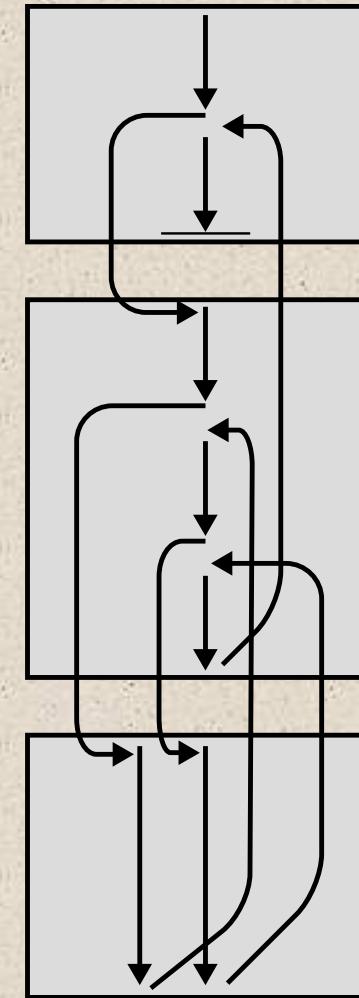


# + Procedure Call Instructions

- Self-contained computer program that is incorporated into a larger program
  - At any point in the program the procedure may be invoked, or *called*
  - Processor is instructed to go and execute the entire procedure and then return to the point from which the call took place
- Two principal reasons for use of procedures:
  - Economy
    - A procedure allows the same piece of code to be used many times
  - Modularity
- Involves two basic instructions:
  - A call instruction that branches from the present location to the procedure
  - Return instruction that returns from the procedure to the place from which it was called

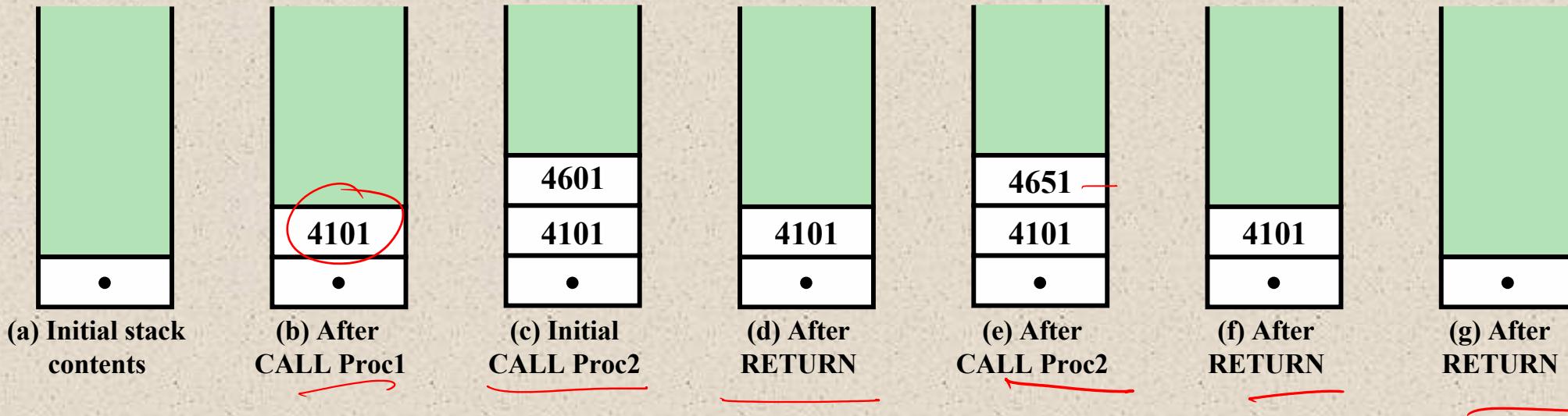


(a) Calls and returns



(b) Execution sequence

Figure 12.8 Nested Procedures



**Figure 12.9 Use of Stack to Implement Nested Procedures of Figure 12.8**

# x86 Status Flags

Status Bit	Name	Description
<u>CF</u>	Carry	Indicates carrying or borrowing out of the left-most bit position following an arithmetic operation. Also modified by some of the shift and rotate operations.
<u>PF</u>	Parity	Parity of the least-significant byte of the result of an arithmetic or logic operation. 1 indicates even parity; 0 indicates odd parity.
<u>AF</u>	Auxiliary Carry	Represents carrying or borrowing between half-bytes of an 8-bit arithmetic or logic operation. Used in binary-coded decimal arithmetic.
<u>ZF</u>	Zero	Indicates that the result of an arithmetic or logic operation is 0.
<u>SF</u>	Sign	Indicates the sign of the result of an arithmetic or logic operation.
<u>OF</u>	Overflow	Indicates an arithmetic overflow after an addition or subtraction for twos complement arithmetic.

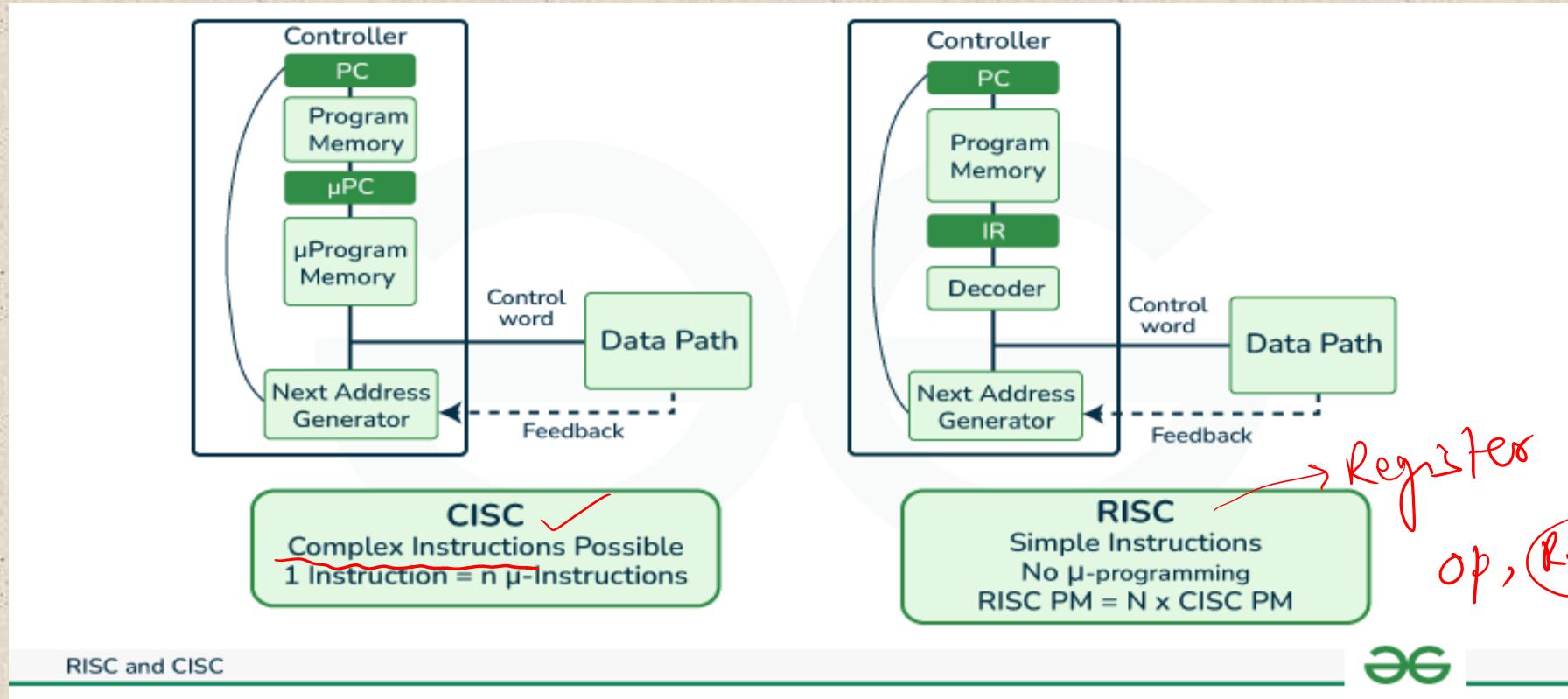
# x86 Condition Codes for Conditional Jump Instructions

Symbol	Condition Tested	Comment
A, NBE	CF=0 AND ZF=0	Above; Not below or equal (greater than, unsigned)
AE, NB, NC	CF=0	Above or equal; Not below (greater than or equal, unsigned); Not carry
B, NAE, C	CF=1	Below; Not above or equal (less than, unsigned); Carry set
BE, NA	CF=1 OR ZF=1	Below or equal; Not above (less than or equal, unsigned)
E, Z	ZF=1	Equal; Zero (signed or unsigned)
G, NLE	[(SF=1 AND OF=1) OR (SF=0 and OF=0)] AND [ZF=0]	Greater than; Not less than or equal (signed)
GE, NL	(SF=1 AND OF=1) OR (SF=0 AND OF=0)	Greater than or equal; Not less than (signed)
L, NGE	(SF=1 AND OF=0) OR (SF=0 AND OF=1)	Less than; Not greater than or equal (signed)
LE, NG	(SF=1 AND OF=0) OR (SF=0 AND OF=1) OR (ZF=1)	Less than or equal; Not greater than (signed)
NE, NZ	ZF=0	Not equal; Not zero (signed or unsigned)
NO	OF=0	No overflow
NS	SF=0	Not sign (not negative)
NP, PO	PF=0	Not parity; Parity odd
O	OF=1	Overflow
P	PF=1	Parity; Parity even
S	SF=1	Sign (negative)

(Table can be found on page 440 in the textbook.)

# RISC vs CISC

+

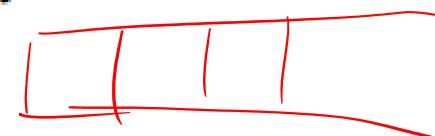


- Complex instruction, so complex decoding.
- Instructions are larger than one-word size.
- May take more than a single CC to get executed.
- Less number of general-purpose registers as operations get performed in memory itself.
- Complex Addressing Modes. 
- More Data types. 

- Simpler instruction & simple decoding.
- Instruction comes undersize of one word.
- takes a single clock cycle to get executed.
- More number of general-purpose registers.
- Simple Addressing Modes.
- Fewer Data types.
- A pipeline can be achieved.

Q.5 Consider the following processor design characteristics.

- I. Register-to-register arithmetic operations only
- II. Fixed-length instruction format
- III. Hardwired control unit

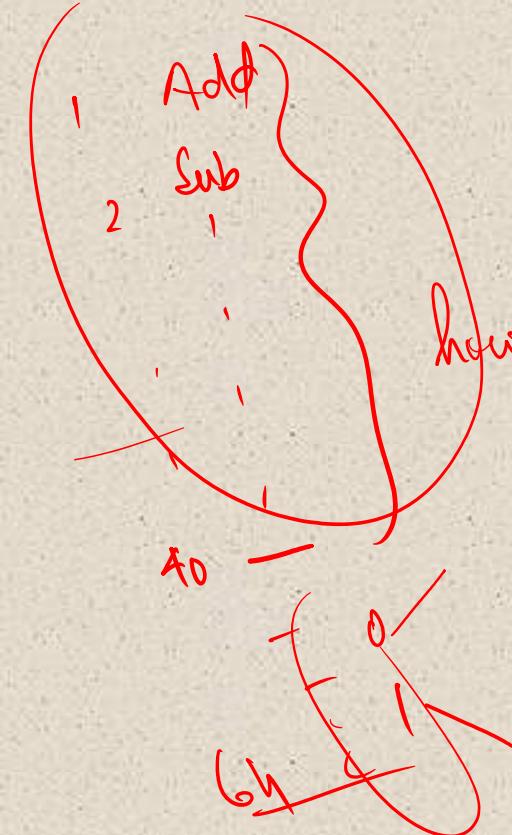


Which of the characteristics above are used in the design of a RISC processor?

- (A) I and II only
- (B) II and III only
- (C) I and III only
- (D) I, II and III

+

- A processor has 40 distinct instructions and 24 general purpose registers. A 32-bit instruction word has an opcode, two register operands and an immediate operand. The number of bits available for the immediate operand field is 16 bits.

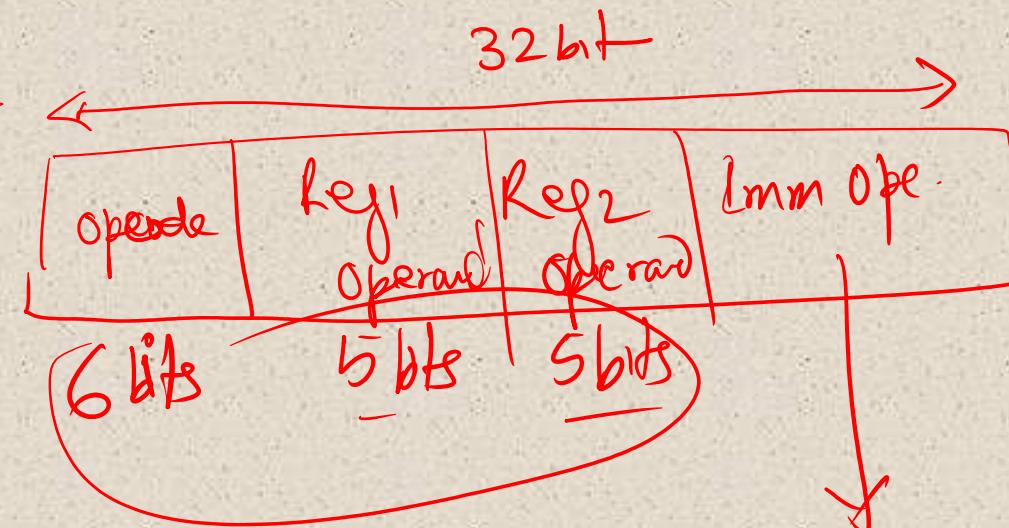


$$\text{GPR} = 24 = \lceil \log_2^{24} \rceil \approx 5$$

how many bits =  $2^5 = 32$

$$\lceil \log_2^{40} \rceil \approx 6$$

$$2^6 = 64$$



$$\text{immediate bits} = 32 - (6 + 5 + 5) \\ = 16 \text{ bits}$$

- + For computers based on three-address instruction formats, each address field can be used to specify which of the following:
- (S1) A memory operand
  - (S2) A processor register
  - (S3) An implied accumulator register

A Either S1 or S2

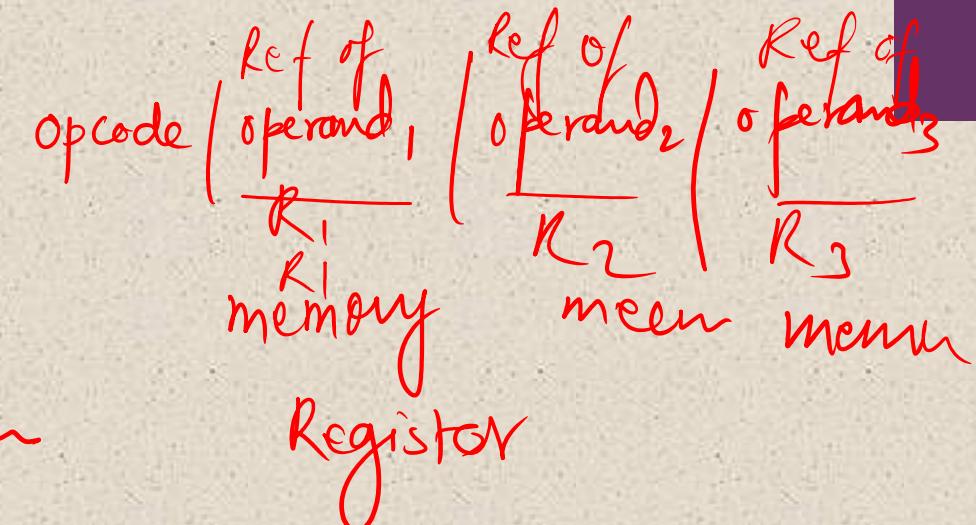
B Either S2 or S3

C Only S2 and S3

D All of S1, S2 and S3

One Address Instruction

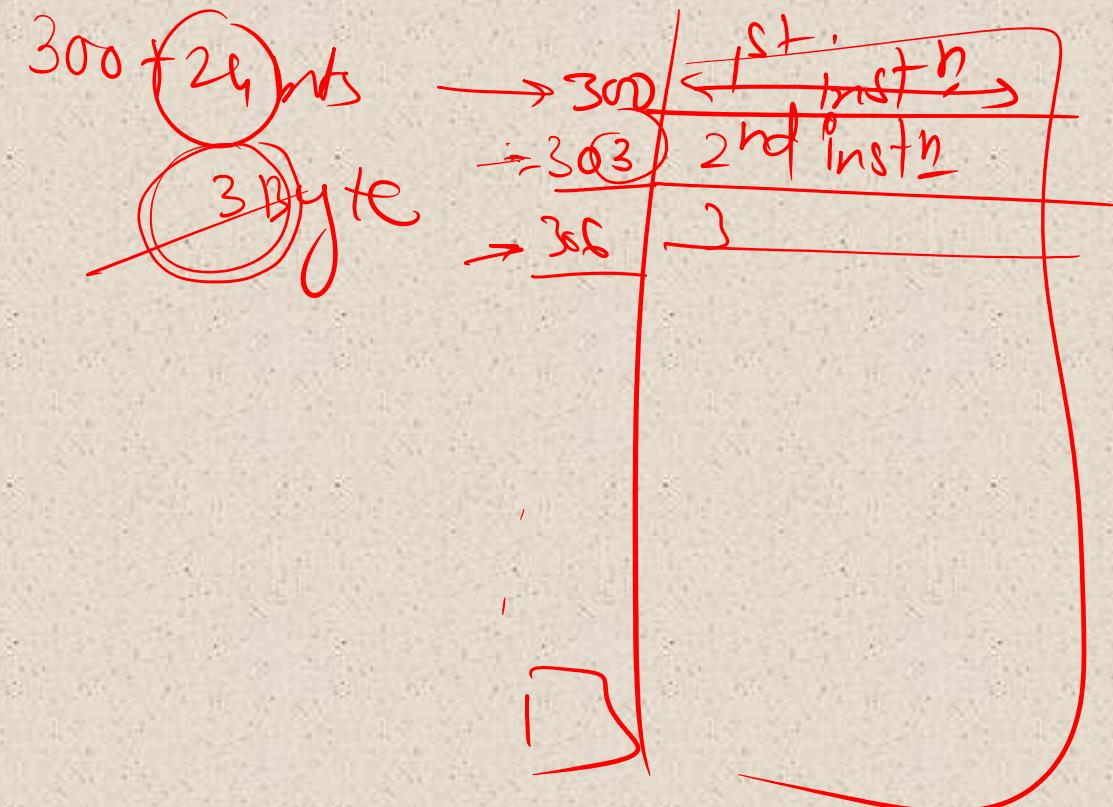
$\text{ADD } \underline{\underline{R_1}} ; \quad AC \leftarrow AC + R_1$



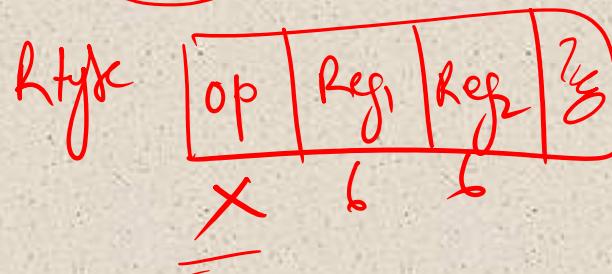
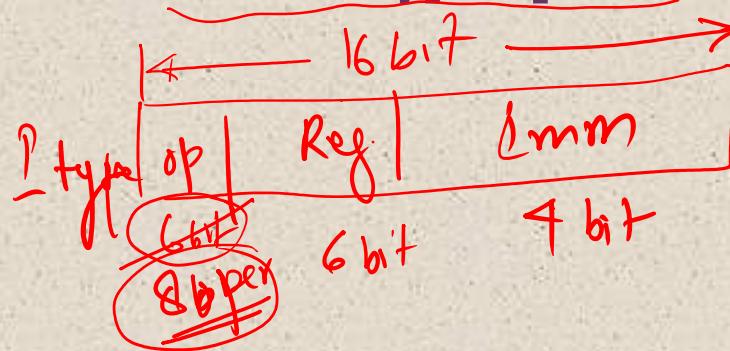


A CPU has 24-bit instructions. A program starts at address 300 (in decimal). Which one of the following is a legal program counter (all values in decimal)?

- A. 400
- B. 500
- C. 600
- D. 700



A processor has 64 registers and uses 16-bit instruction format. It has two types of instructions: I-type and R-type. Each I-type instruction contains an opcode, a register name, and a 4-bit immediate value. Each R-type instruction contains an opcode and two register names. If there are 8 distinct I-type opcodes, then the maximum number of distinct R-type opcodes is \_\_\_\_\_. Bits required for registers =  $\lceil \log_2 64 \rceil = 6 \text{ bit}$



$$\text{Total insts are } = 2^{16}$$

$$\underline{\text{I-type} + \text{R-type}} = 2^{16}$$

$$\frac{8 \times 2^6 \times 2^4}{2^3 \times 2^6 \times 2^4} + \underline{x \cdot 2^6 \cdot 2^6} = 2^{16}$$

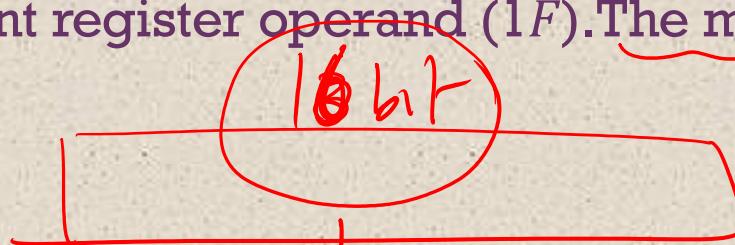
$$\frac{2^3 \times 2^6 \times 2^4}{2^3 \times 2^6 \times 2^4} + \underline{x \cdot 2^{12}} = 2^{16}$$

$$2 + \underline{x} = 2^4 \Rightarrow x = 16 - 2 = 14$$

14 Distinct opcode for R-type.

+

A processor has 16 integer registers ( $R_0, R_1, \dots, R_{15}$ ) and 64 floating point registers ( $F_0, F_1, \dots, F_{63}$ ). It uses a 2-byte instruction format. There are four categories of instructions: Type-1, Type-2, Type-3, and Type-4. Type-1 category consists of four instructions, each with 3 integer register operands (3Rs). Type-2 category consists of eight instructions, each with 2 floating point register operands (2Fs). Type-3 category consists of fourteen instructions, each with one integer register operand and one floating point register operand (1R+1F). Type-4 category consists of  $N$  instructions, each with a floating point register operand (1F). The maximum value of  $N$  is \_\_\_\_\_.



$$\text{Total } mst^n = 2^{16}$$

$$\begin{aligned}
 T_1 &= 4 \times 2^4 \times 2^4 \times 2^4 = 2^{15} \\
 T_2 &= 8 \times 2^6 \times 2^6 = 2^{14} \\
 T_3 &= 14 \times 2^4 \times 2^6 = 2^{11} \\
 T_4 &= N \times 2^6
 \end{aligned}$$

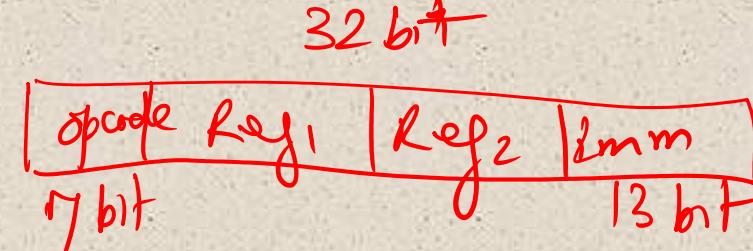
$(T_1 + T_2 + T_3 + T_4) = 2^{16}$   
 $\underline{\underline{N = 32}}$



Consider a 32-bit processor which supports 70 instructions. Each instruction is 32 bit long and has 4 fields namely opcode, two register identifiers and an immediate operand of unsigned integer type. Maximum value of the immediate operand that can be supported by the processor is 8191. How many registers the processor has?

- A. 32
- B. 64
- C. 128
- D. 16

$$\lceil \log_2 70 \rceil \approx 7 \text{ bits}$$



$$\begin{aligned}
 R_1 + R_2 &= 32 - (7 + 13) \\
 &= 12 \text{ bits}
 \end{aligned}$$

One register = 6 bits

$2^6$  No. of Register will be in processor

+ A processor has 16 integer registers ( $R_0, R_1, \dots, R_{15}$ ) and 64 floating point registers ( $F_0, F_1, \dots, F_{63}$ ). It uses a 2-byte instruction format. There are four categories of instructions: Type-1, Type-2, Type-3, and Type-4.

Type-1 category consists of four instructions, each with 3 integer register operands ( $3Rs$ )

Type-2 category consists of eight instructions, each with 2 floating point register operands ( $2Fs$ ).

Type-3 category consists of fourteen instructions, each with one integer register operand and one floating point register operand ( $1R + 1F$ ).

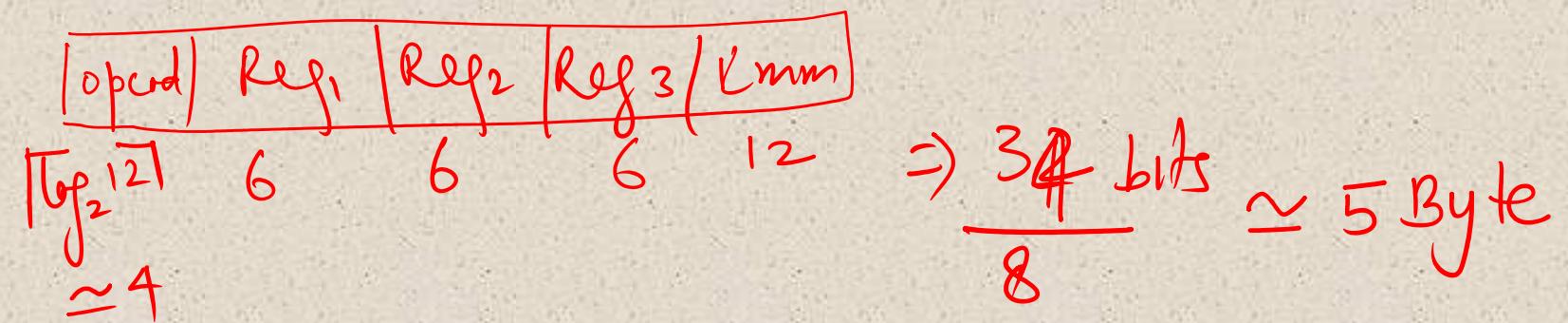
Type-4 category consists of  $N$  instructions, each with a floating point register operand ( $1F$ ).

The maximum value of  $N$  is \_\_\_\_\_.

*Repeated*

→ 6 bits

- Consider a processor with 64 registers and an instruction set of size twelve. Each instruction has five distinct fields, namely, opcode, two source register identifiers, one destination register identifier, and a twelve-bit immediate value. Each instruction must be stored in memory in a byte-aligned fashion. If a program has 100 instructions, the amount of memory (in bytes) consumed by the program text is \_\_\_\_\_.



$$100 \text{ instn} \times 5 \text{ B} = 500 \text{ Bytes}$$

A processor with 16 general purpose registers uses a 32-bit instruction format. The instruction format consists of an opcode field, an addressing mode field, two register operand fields, and a 16-bit scalar field. If 8 addressing modes are to be supported, the maximum number of unique opcodes possible for every addressing mode is

---



- A processor uses a 32-bit instruction format and supports byte-addressable memory access. The ISA of the processor has 150 distinct instructions. The instructions are equally divided into two types, namely R-type and I-type, whose formats are shown below.

R-type Instruction Format:

OPCODE | UNUSED | DST Register | SRC Register1 | SRC Register 2

I-type Instruction Format:

OPCODE | DST Register | SRC Register | # Immediate value/address

In the OPCODE, 1 bit is used to distinguish between I-type and R-type instructions and the remaining bits indicate the operation. The processor has 50 architectural registers, and all register fields in the instructions are of equal size. Let  $X$  be the number of bits used to encode the UNUSED field,  $Y$  be the number of bits used to encode the OPCODE field, and  $Z$  be the number of bits used to encode the immediate value/address field. The value of  $X + 2Y + Z$  is \_\_\_\_\_

# Summary

## Chapter 12

- Machine instruction characteristics
  - Elements of a machine instruction
  - Instruction representation
  - Instruction types
  - Number of addresses
  - Instruction set design
- Types of operands
  - Numbers
  - Characters
  - Logical data

## Instruction Sets: Characteristics and Functions

- Intel x86 and ARM data types
- Types of operations
  - Data transfer
  - Arithmetic
  - Logical
  - Conversion
  - Input/output
  - System control
  - Transfer of control
- Intel x86