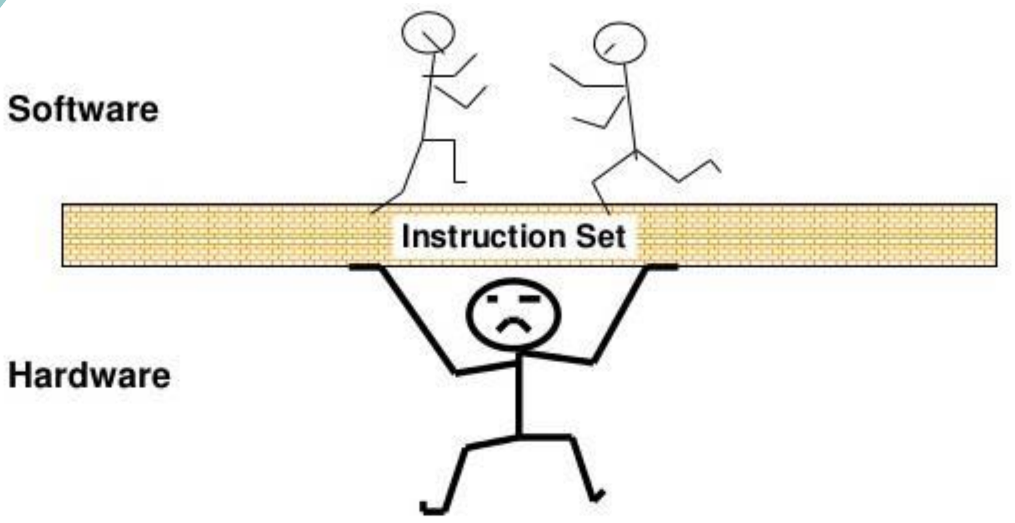
ساختار و زبان کامپیوتر

فصل سه

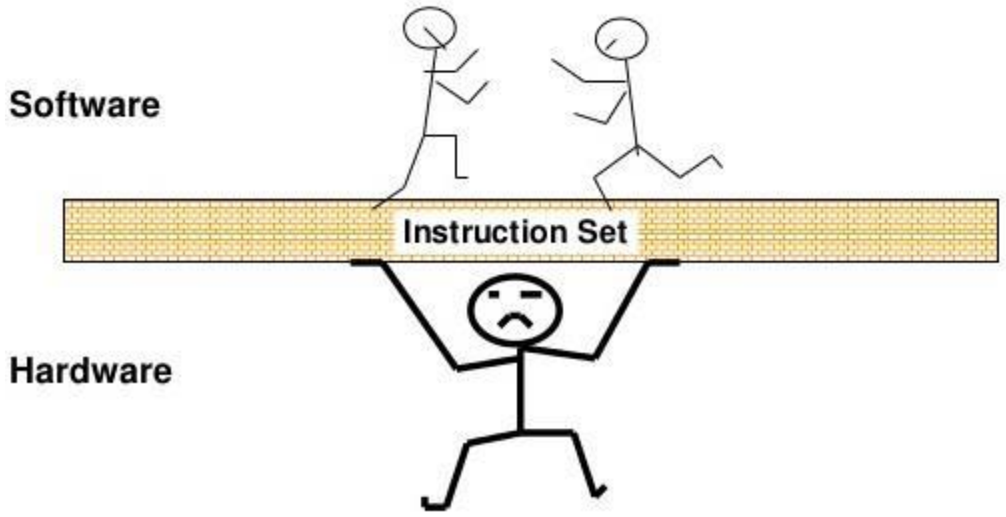
معماری مجموعه دستورات



Computer Structure & Machine Language

Chapter Three

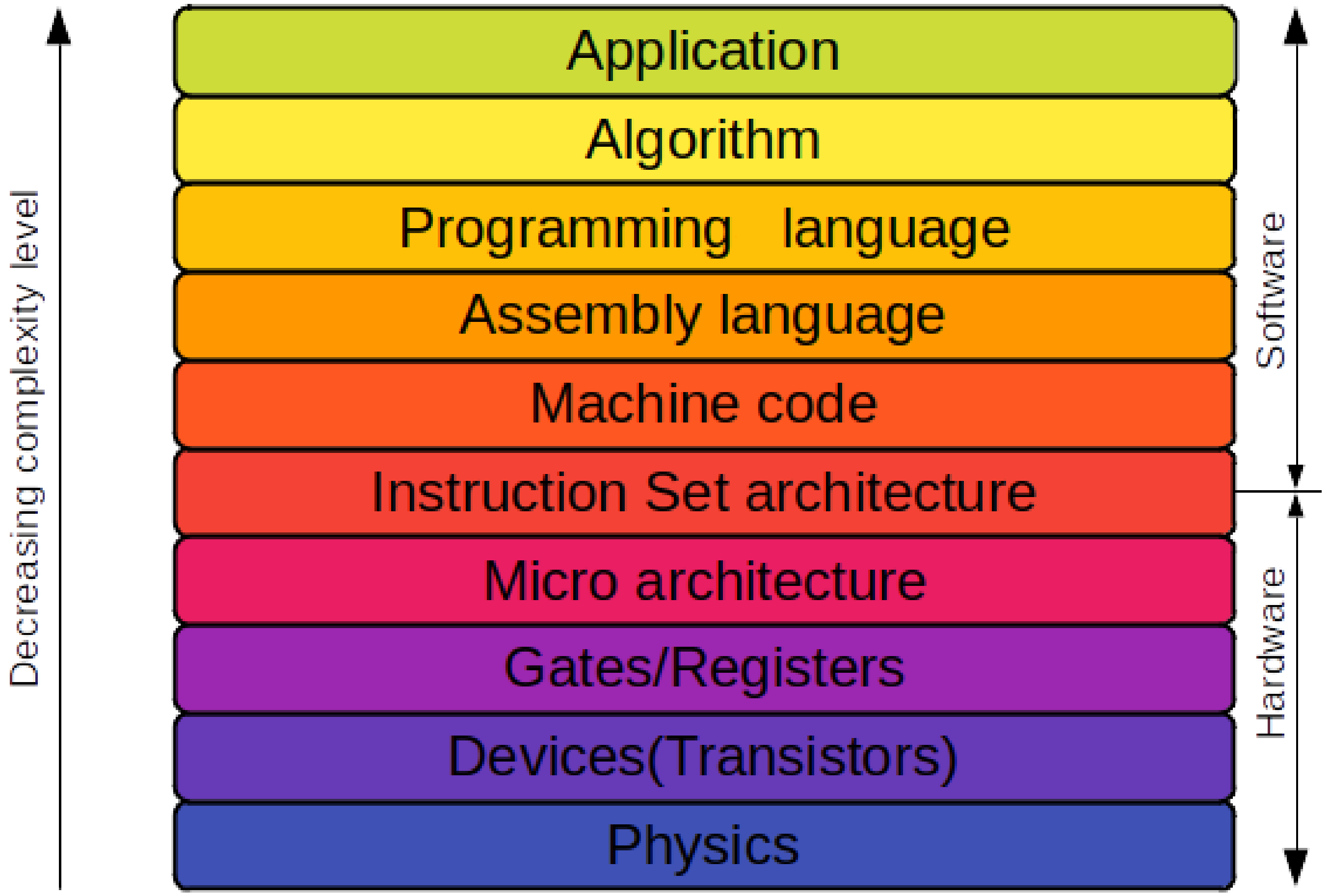
Instruction Set Architecture



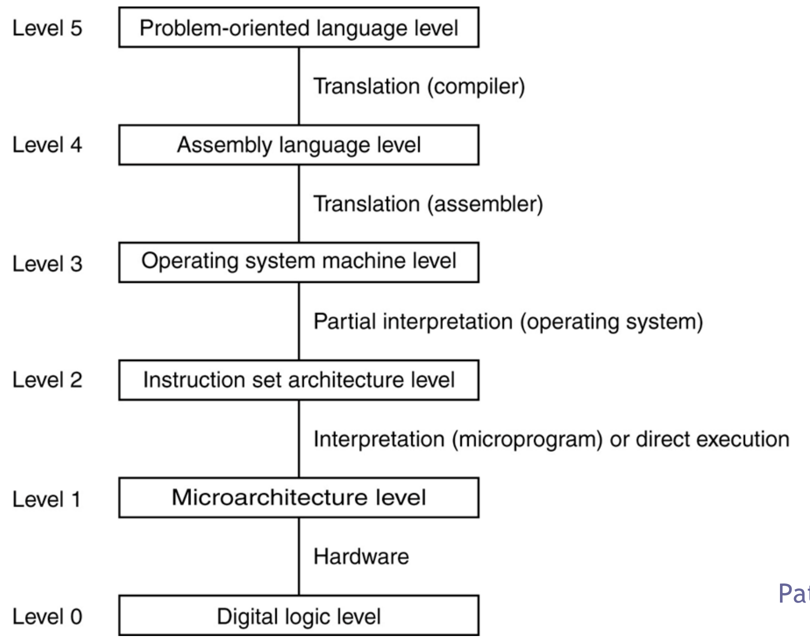
Copyright Notice

Contents

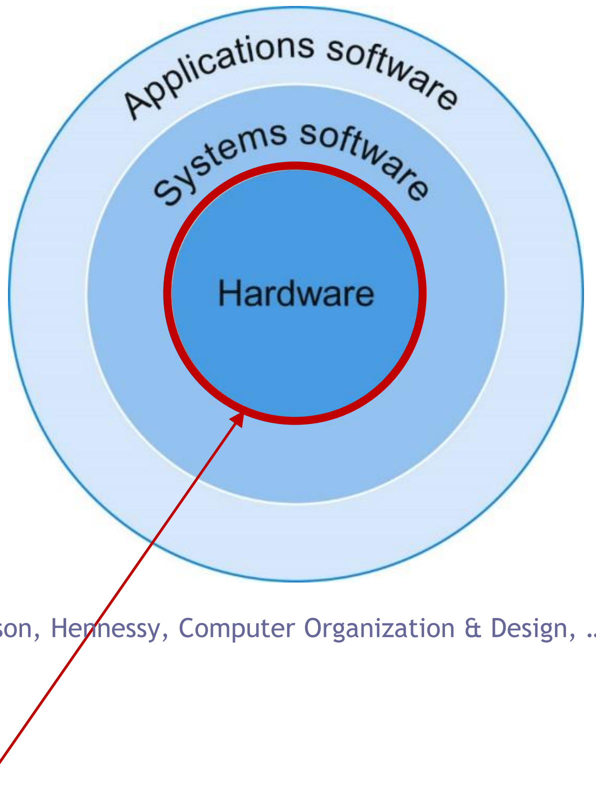
Computer Layers of Abstraction



ISA Placement


Level 5
Problem-oriented language level
Translation (compiler)
Level 4
Assembly language level
Translation (assembler)
Level 3
Operating system machine level
Partial interpretation (operating system)
Level 2
Instruction set architecture level
Interpretation (microprogram) or direct execution
Level 1
Microarchitecture level
Hardware
Level 0
Digital logic level

Tanenbaum, Structured Computer Organization, 2006



Patterson, Hennessy, Computer Organization & Design, …, 2020

Instruction Set Architecture (ISA)

Instruction Set Architecture (ISA)

- How the machine appears to a machine language programmer
- What a compiler outputs
  - ignoring operating-system calls & symbolic assembly language
- Specifies:
  - Memory Model
  - Registers
  - Available data types
  - Available instructions

ISA Exclusion

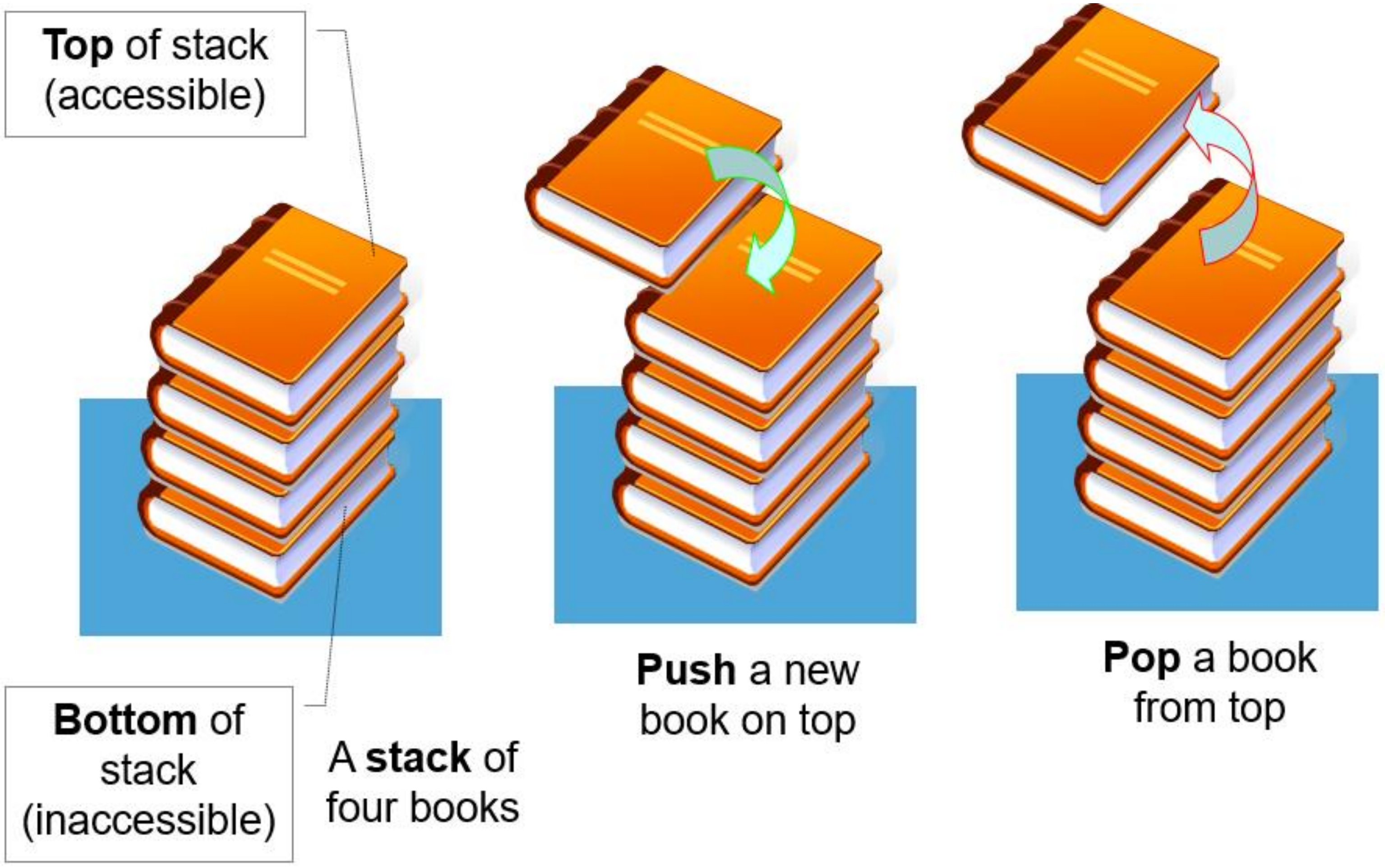
- Not part of ISA (not visible to compiler):
  - it is pipelined or not
  - it is superscalar or not
  - cache memory is used or not
  - …
- However some of these properties do affect performance & is better to be visible to the compiler writer!

Contents

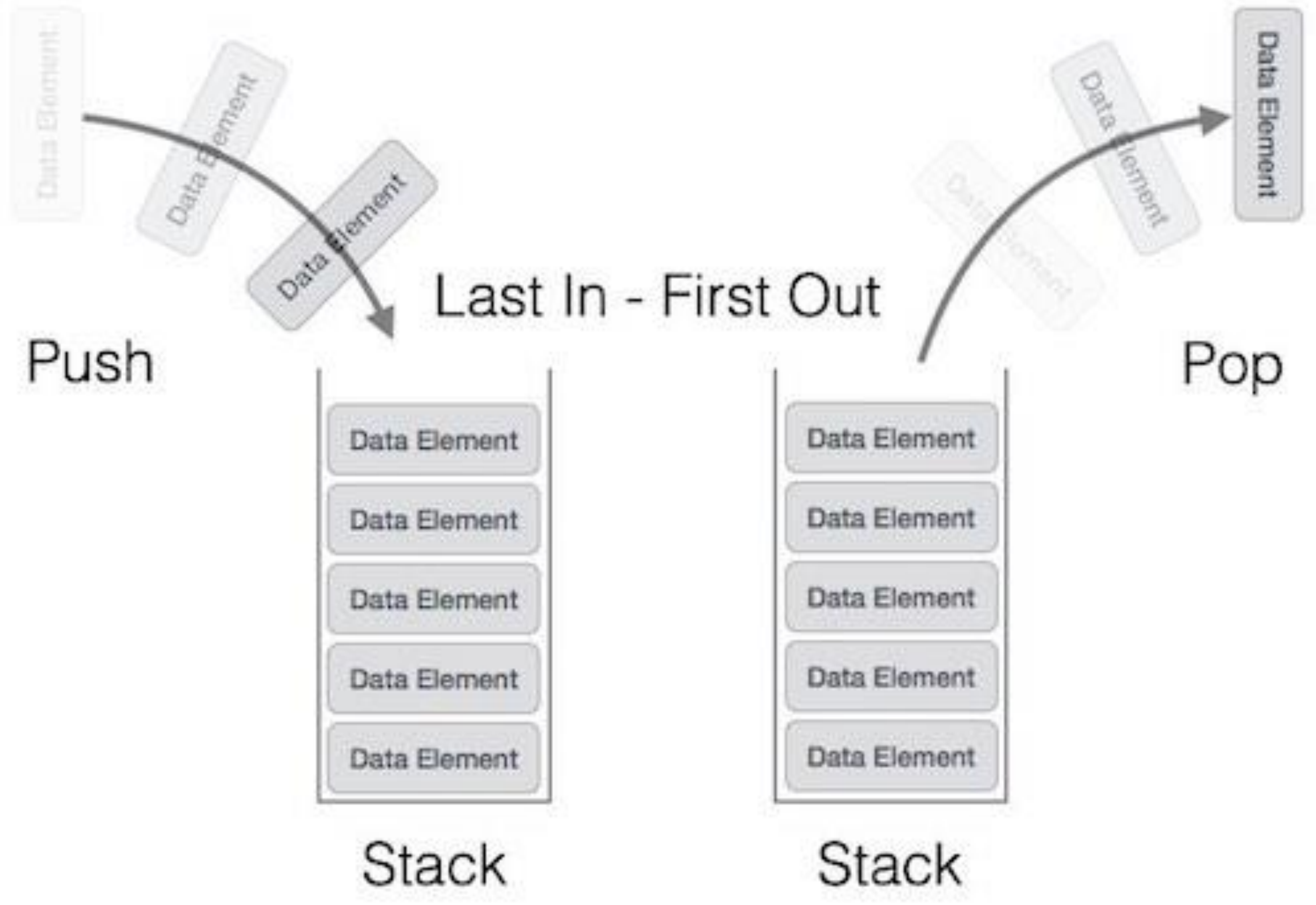
Where Operands Reside?

- Stack Machine
- Accumulator Machine
- Register-Memory Machine
- Register-Register Machine (Load-Store)

Stack Illustration


Top of stack (accessible)
Bottom of stack (inaccessible)
A **stack** of four books
Push a new book on top
Pop a book from top

Stack Representation



Stack Machine

- Zero-operand ISA
  - No operands for ALU operations (add, sub, …)
- "Push"
  - Loads mem into 1st reg ("top of stack")
- "Pop"
  - Does reverse
- "Add", "Sub", "Mul", and etc.
  - Combines contents of first two regs on top of stack

Example 1

Code sequence for **C = A + B**

Stack **Accumulator** **Register-Memory** **Reg-Reg**

?

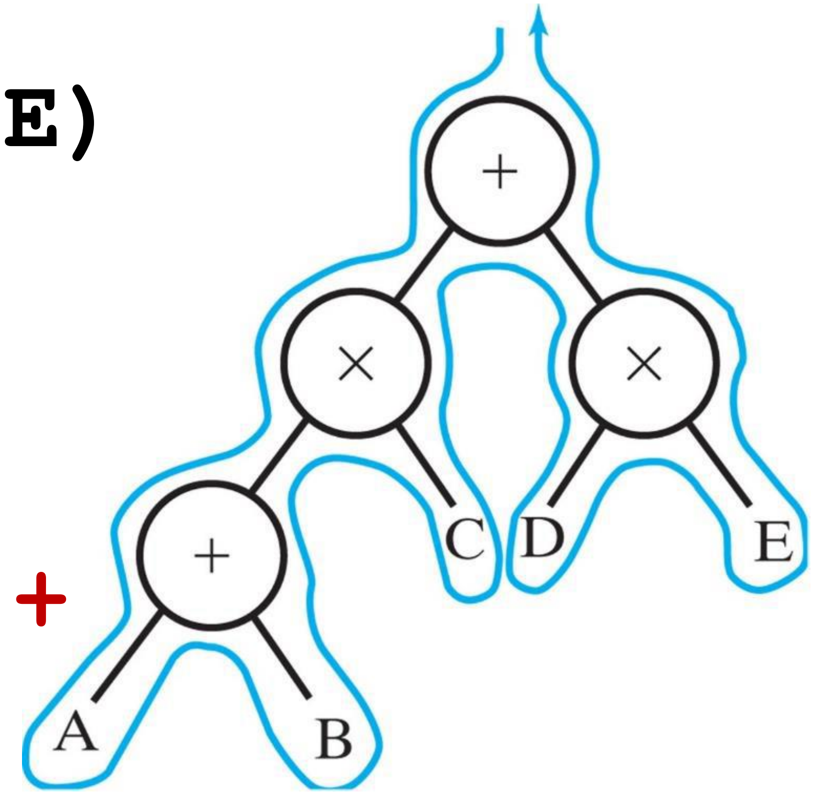
Example 1.1

Code sequence for **C = A + B**

Stack	**Accumulator**	**Register-Memory**	**Reg-Reg**
push A			
push B			
add			
pop C			

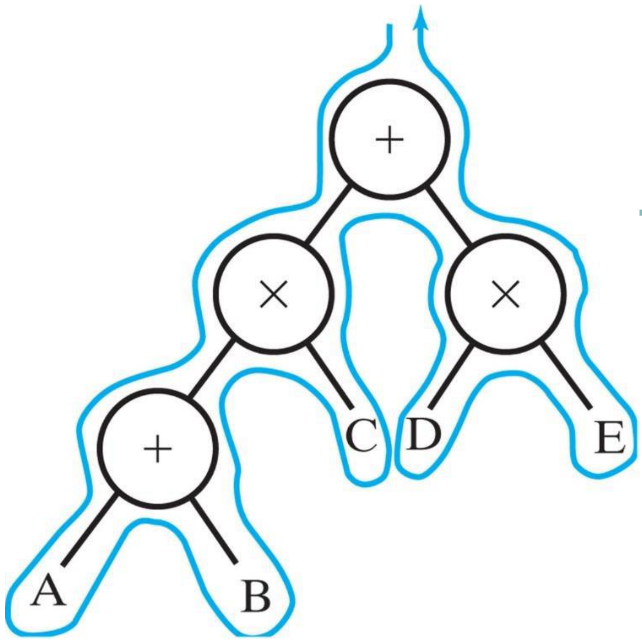
Postfix Notation

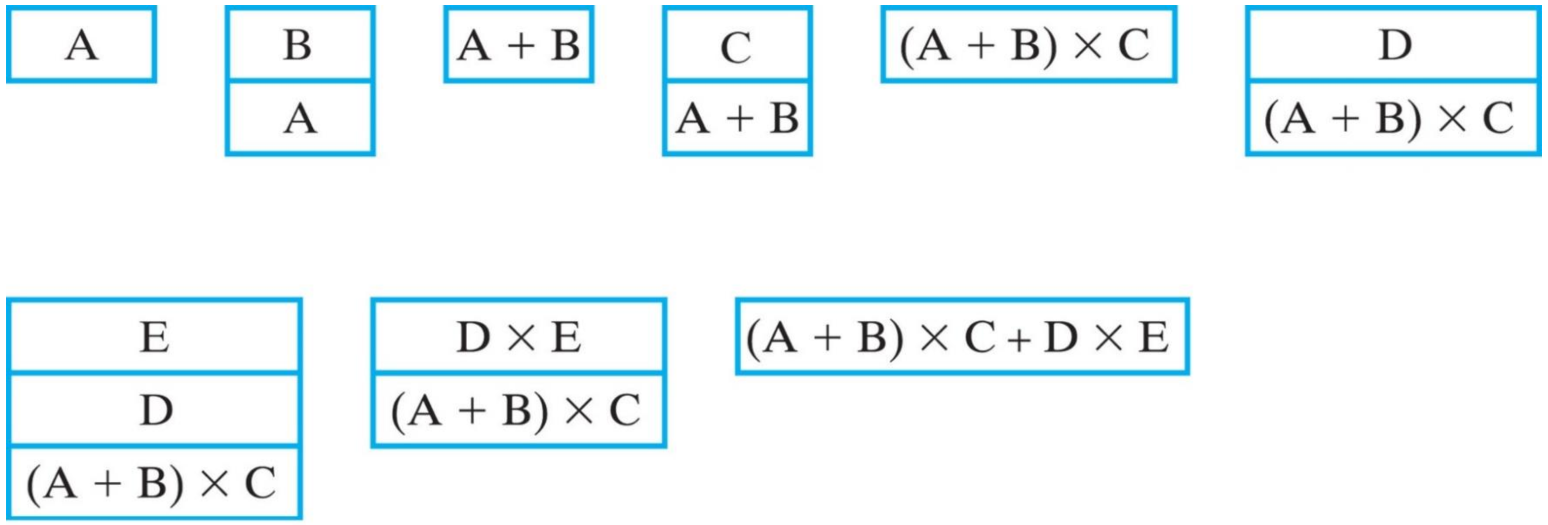
(A+B)×C+(D×E)



AB+C×DE×+

AB+C×DE×+





Accumulator Machine

- 1-operand ISA
- Only one register called accumulator
- Stores intermediate arithmetic & logic results
- Instructions include:
  - “STORE” (Store AC)
  - “LOAD” (Load AC)
  - “ADD mem” (AC $\leftarrow$ AC + mem)

Example 1

Code sequence for **C = A + B**

Stack **Accumulator** **Register-Memory** **Reg-Reg**

Example 1.2

Code sequence for **C = A + B**

<u>**Stack**</u>	<u>**Accumulator**</u>	<u>**Register-Memory**</u>	<u>**Reg-Reg**</u>
push A	load A		
push B	add B		
add	store C		
pop C			

Register-Memory Machine

- Two or three Operands ISA
- A set of general purpose registers available
- Operands can be register or memory
- Arithmetic & logic instructions can use data in registers and/or memory
- Usually only one operand can be in memory

Example 1

Code sequence for **C = A + B**

Stack **Accumulator** **Register-Memory** **Reg-Reg**

?

Example 1.3

Code sequence for **C = A + B**

<u>**Stack**</u>	<u>**Accumulator**</u>	<u>**Register-Memory**</u>	<u>**Reg-Reg**</u>
push A	load A	move R1,A	
push B	add B	add R1,B	
add	store C	move C,R1	
pop C			

Register-Register Machine

- Also called Load-Store Machine
- Two or three operands ISA
- A set of general purpose registers available
- Arithmetic & logical instructions can only access registers
- Access to memory only with Load & Store

Example 1

Code sequence for **C = A + B**

Stack **Accumulator** **Register-Memory** **Reg-Reg**

?

Example 1.4

Code sequence for **C = A + B**

Stack	**Accumulator**	**Register-Memory**	**Reg-Reg**
push A	load A	move R1,A	load R1,A
push B	add B	add R1,B	load R2,B
add	store C	move C,R1	add R3,R1,R2
pop C			store C,R3

Example 2

$X = (A + B) * (C + D)$

Register-Memory

Reg-Reg

Stack

Accumulator

?

ISA Classes

X = (A + B) * (C + D)

Register-Memory

```
ADD R1,A,B     MOV R1,A
ADD R2,C,D     ADD R1,B
MUL X,R1,R2    MOV R2,C
               ADD R2,D
               MUL R1,R2
               MOV X,R1
```

Stack

```
PUSH A
PUSH B
ADD
PUSH C
PUSH D
ADD
MUL
POP X
```

Accumulator

```
LOAD    A
ADD     B
STORE   T
LOAD    C
ADD     D
MUL     T
STORE   X
```

Reg-Reg

```
LOAD    R1,A
LOAD    R2,B
LOAD    R3,C
LOAD    R4,D
ADD     R1,R1,R2
ADD     R3,R3,R4
MUL     R1,R1,R2
STORE   X,R1
```

Contents

Addressing Modes

- Implicit
- Immediate
- Register (direct)
- Register indirect
- Base or displacement addressing
- Indexed addressing
- Auto-increment / Auto-decrement
- PC-relative
- Memory direct
- Memory indirect

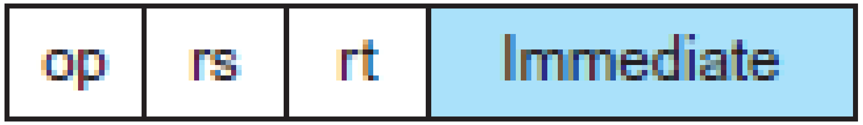
Implied Addressing

- The operand is specified implicitly in the instruction, such as:
  - Register-reference instructions that use an accumulator register
  - Zero-address instructions in a stack machine
    - Operands are implied to be on top of stack

Immediate Addressing

- Operand is a constant within instruction
- MIPS-32 example:

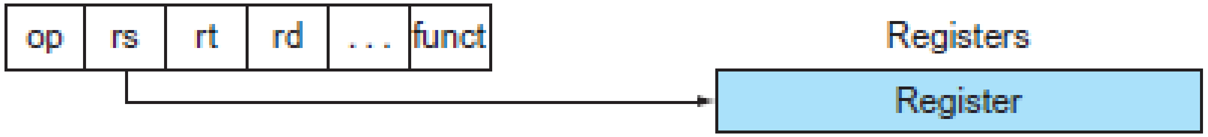
```
addi $s0,$s1,10    # $s0 ← $s1 + 10
```



Register (Direct) Addressing

- Operand is a **register**
- MIPS-32 example:

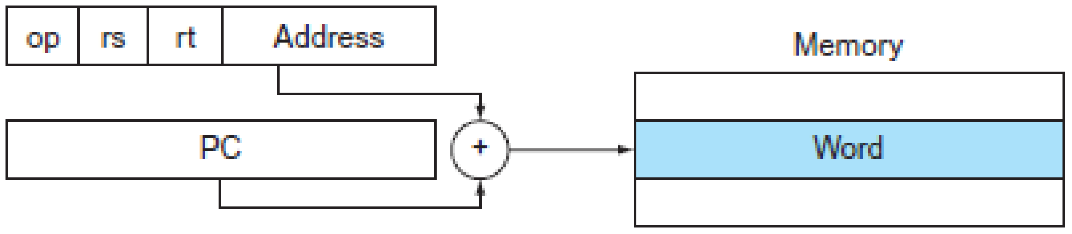
```
add $s0,$s1,$s2    # $s0 ← $s1 + $s2
```



PC-Relative Addressing

- Address is sum of PC and a constant within instruction:
  - MIPS-32 example:

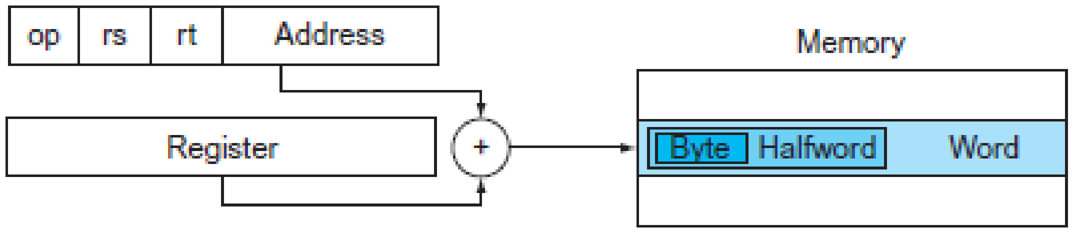
```
bne $s1,$s2,L1   # if($s1!=$s2)
                  # goto PC+L1
```



Base or Displacement Addressing

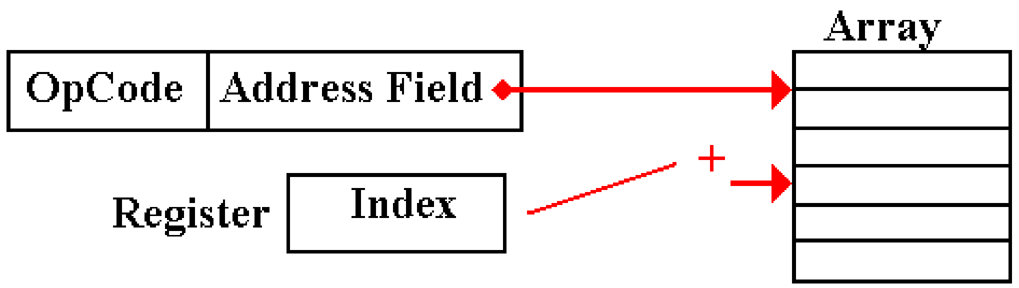
- Address of operand (in memory) is sum of a base register and a constant displacement within instruction
  - MIPS-32 example:

```
lw $s1,10($sp)   # $s1 ← Mem[$sp+10]
```



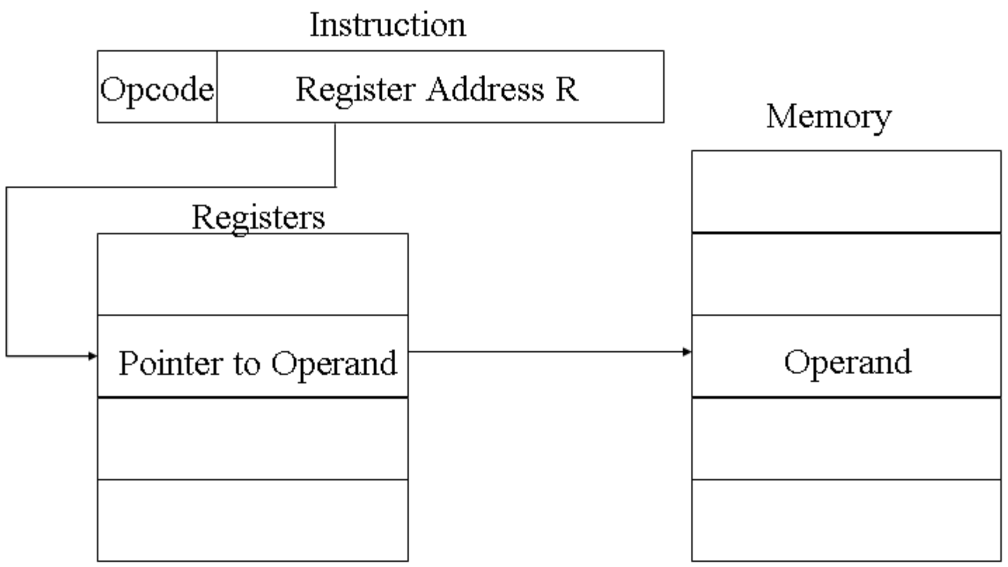
Indexed Addressing

- Address of operand (in memory) is sum of an index register and a constant within the instruction



Register Indirect Addressing

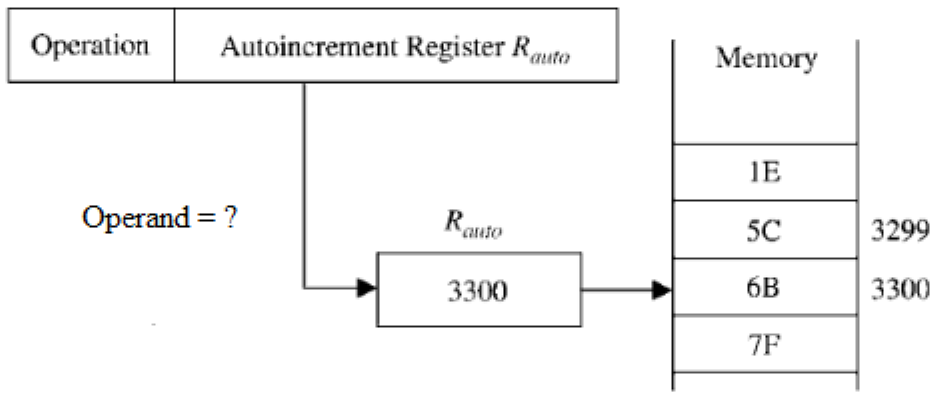
- Operand’s address is in a register



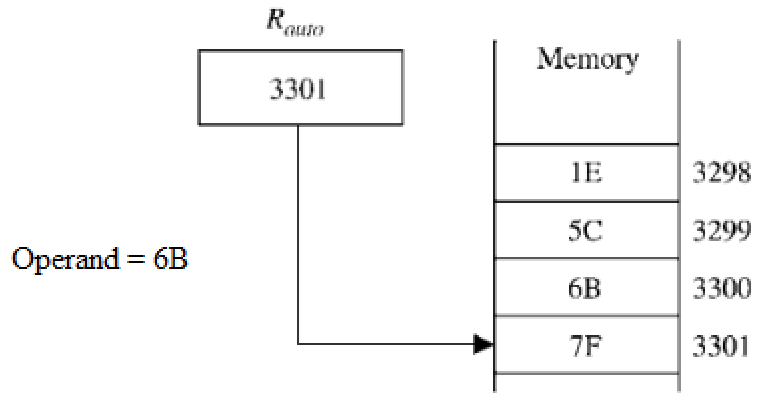
Auto-increment/ Auto-decrement

- Same as Register Indirect, except that register value is incremented/ decremented after/before instruction execution

Auto-increment Addressing Mode

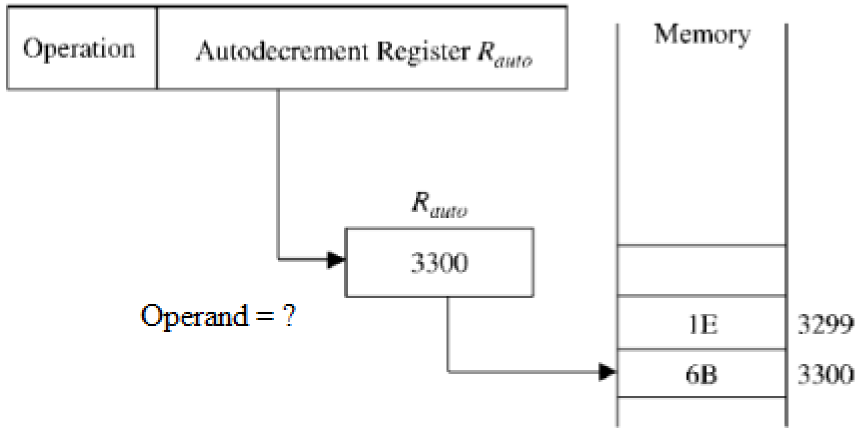


(a) Before execution

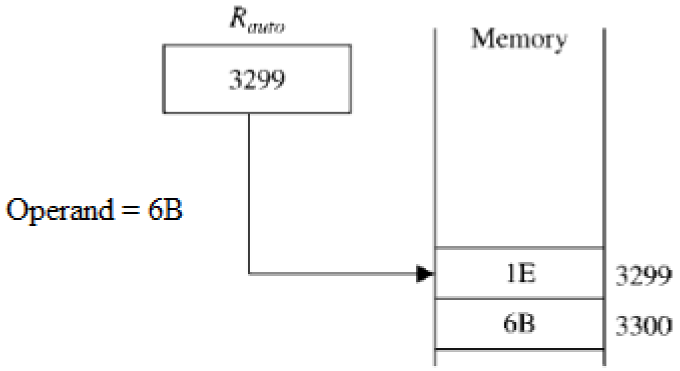


(b) After execution

Auto-decrement Addressing Mode



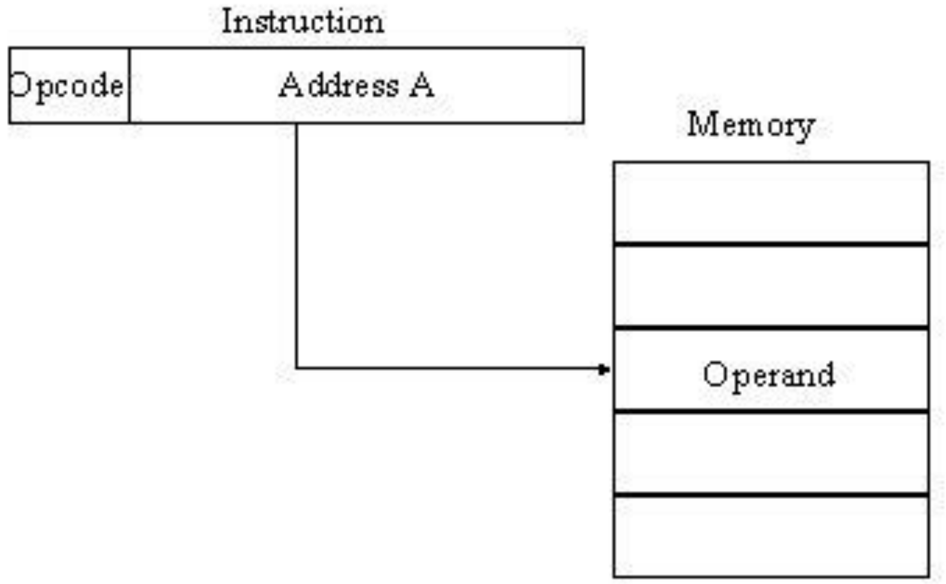
(a) Before execution



(b) After execution

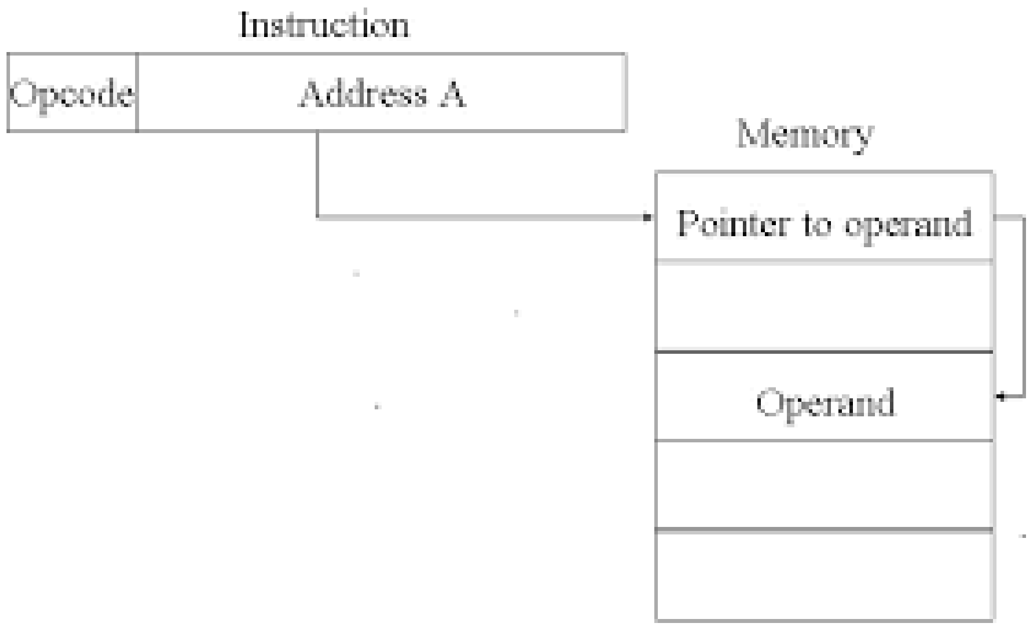
Memory Direct Addressing

- Operand is directly addressed in the instruction



Memory Indirect Addressing

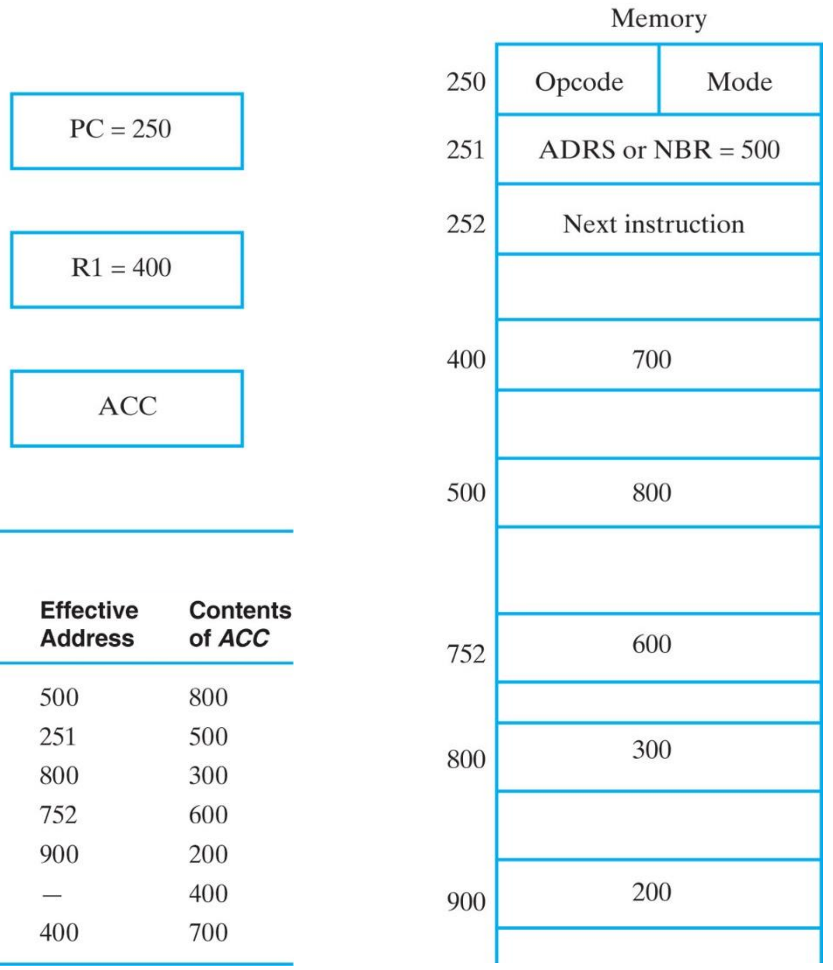
- Operand's address is in a memory location addressed in the instruction



Summary

- No address field in the instruction:
  - Implied Addressing: Operand is an implied register
  - Immediate Addressing: Operand is a constant value named in the instruction
- Register Addressing:
  - Direct: Operand is in a register named in the instruction
  - Indirect, autodec/inc: Operand address is in a register named in the instruction
- Memory Addressing:
  - Direct: Operand is in the memory, its address is in the instruction
  - Indirect: Operand is in the memory, the address of its address is in the instruction
- Register & Memory Addressing:
  - Relative Addressing: Effective address = (PC) + constant
  - Base Register Addressing: Effective address = (a base reg) + constant
  - Indexed Addressing: Effective address = (an index reg) + constant

Symbolic Conventions

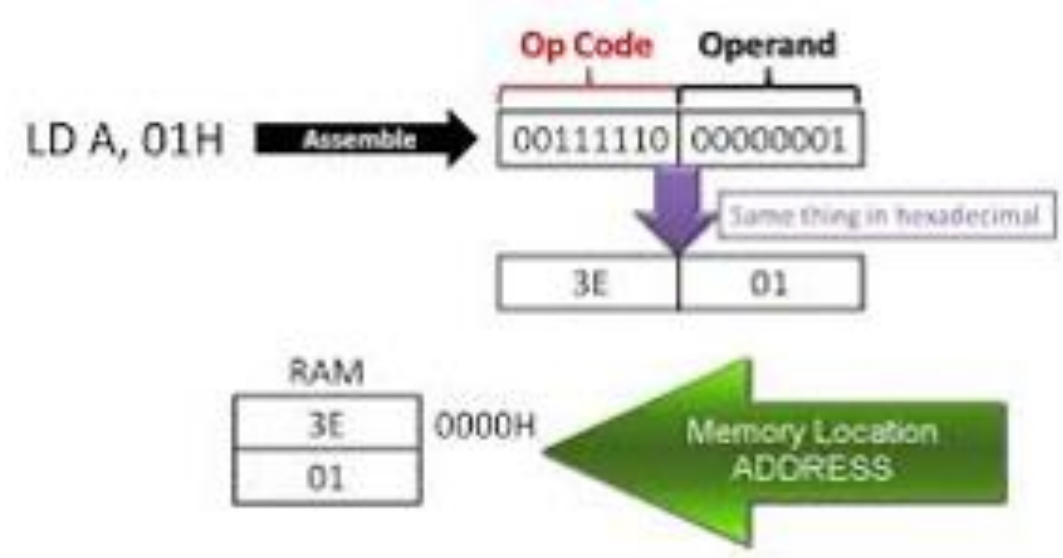


Addressing Mode	Symbolic Convention	Register Transfer	Effective Address	Contents of *ACC*
Direct	LDA ADRS	$ACC \leftarrow M[ADRS]$	500	800
Immediate	LDA #NBR	$ACC \leftarrow NBR$	251	500
Indirect	LDA [ADRS]	$ACC \leftarrow M[M[ADRS]]$	800	300
Relative	LDA $ADRS	$ACC \leftarrow M[ADRS + PC]$	752	600
Index	LDA ADRS (R1)	$ACC \leftarrow M[ADRS + R1]$	900	200
Register	LDA R1	$ACC \leftarrow R1$	—	400
Register-indirect	LDA (R1)	$ACC \leftarrow M[R1]$	400	700

Contents

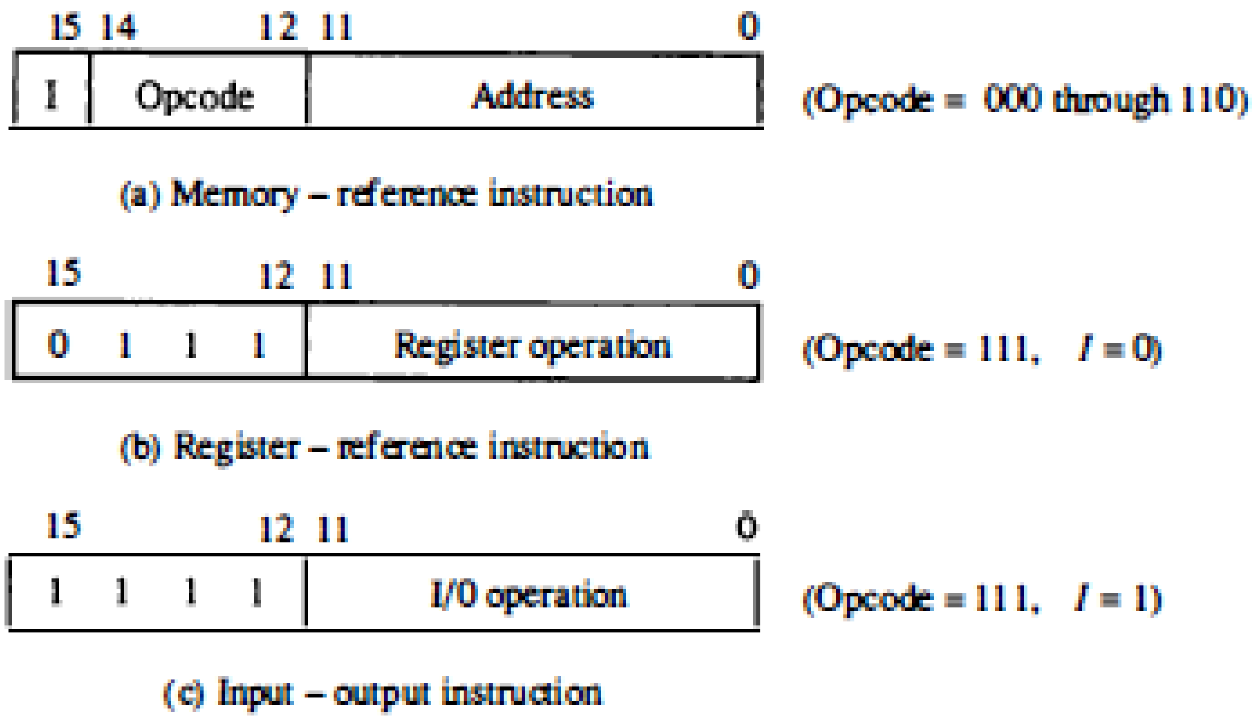
Instruction & Operand format

Which bits designate what?



Can we have several formats in one machine?

Example: Mano's Basic Computer

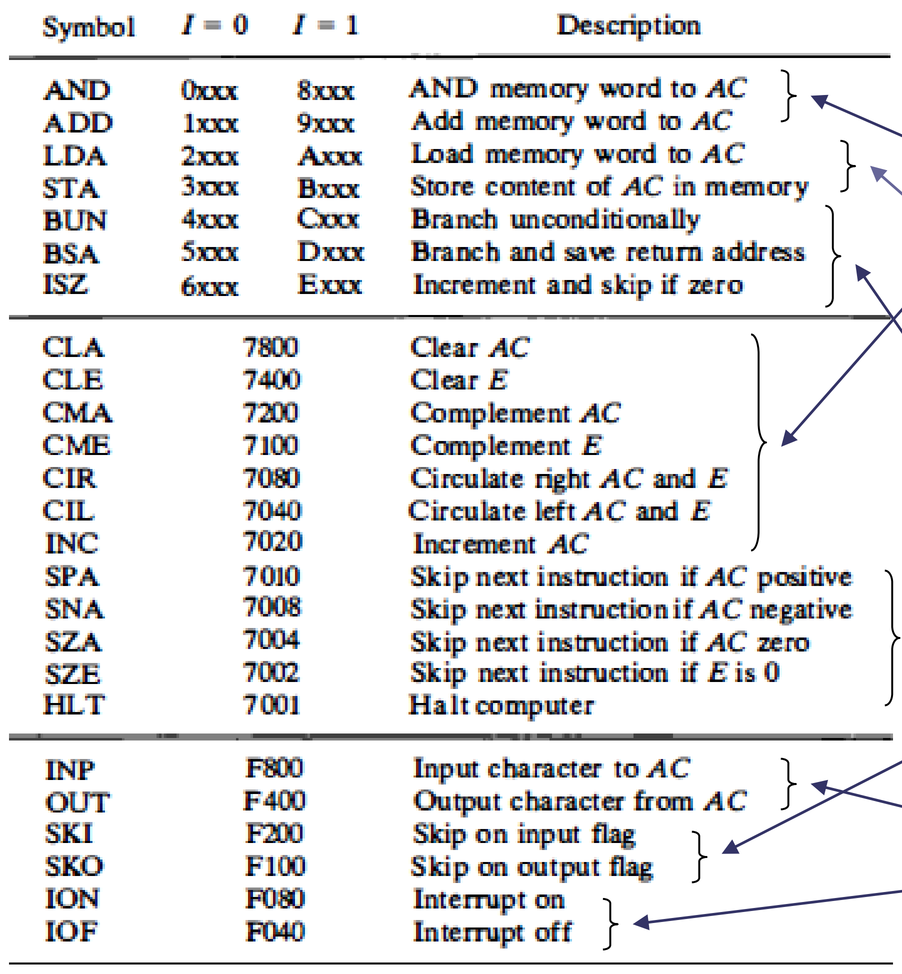


(a) Memory – reference instruction

(b) Register – reference instruction

(c) Input – output instruction

Mano’s Basic Computer (cont)



Symbol	*I* = 0	*I* = 1	Description
AND	0xxx	8xxx	AND memory word to *AC*
ADD	1xxx	9xxx	Add memory word to *AC*
LDA	2xxx	Axxx	Load memory word to *AC*
STA	3xxx	Bxxx	Store content of *AC* in memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero
CLA	7800		Clear *AC*
CLE	7400		Clear *E*
CMA	7200		Complement *AC*
CME	7100		Complement *E*
CIR	7080		Circulate right *AC* and *E*
CIL	7040		Circulate left *AC* and *E*
INC	7020		Increment *AC*
SPA	7010		Skip next instruction if *AC* positive
SNA	7008		Skip next instruction if *AC* negative
SZA	7004		Skip next instruction if *AC* zero
SZE	7002		Skip next instruction if *E* is 0
HLT	7001		Halt computer
INP	F800		Input character to *AC*
OUT	F400		Output character from *AC*
SKI	F200		Skip on input flag
SKO	F100		Skip on output flag
ION	F080		Interrupt on
IOF	F040		Interrupt off

Arithmetic / Logic Instructions

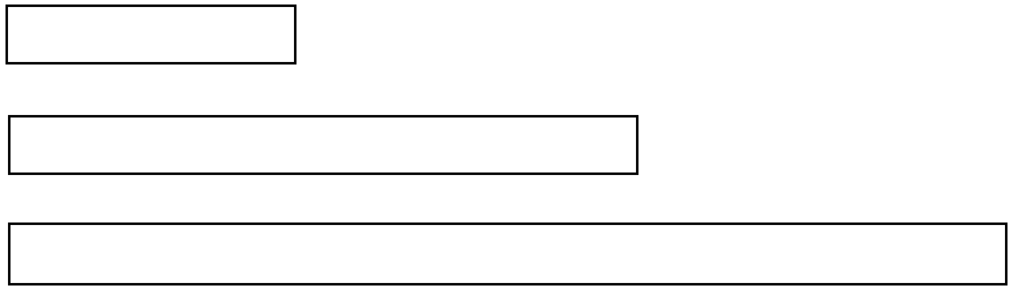
Move Instructions

Program Control Instructions

I/O Instructions

Instruction Length

Variable:



x86 – Instructions vary from 1 to 17 Bytes long

VAX – from 1 to 54 Bytes

Fixed:

MIPS, PowerPC:

all instruction are 4 Bytes long

Variable-length Instructions

- Require multi-step fetch and decode
- Allow for a more flexible and compact instruction set
- CISC processors like x86 & VAX
  - (Complex Instruction Set Computing)

Fixed-length Instructions

- Allow easy fetch and decode
- Simplify pipelining and parallelism
- RISC processors like MIPS & PowerPC
  - (Reduced Instruction Set Computing)

Contents

CISC/RISC Instruction Sets

- Complex instruction set computers
  - provide hardware support for high-level language operations
  - have compact programs
- Reduced instruction set computers
  - Simple instructions and flexibility
  - provide
    - higher throughput
    - faster execution

RISC Architecture

- Memory accesses are restricted to load & store instructions, and data manipulation instructions are register-to-register
- Addressing modes are limited in number
- Instruction formats are all of the same length
- Instructions perform elementary operations

CISC Architecture

- Memory access is directly available to most types of instructions
- Addressing modes are substantial in number
- Instruction formats are of different lengths
- Instructions perform both elementary and complex operations

Hybrid Solution

- RISC core & CISC interface
- Actual ISAs range between those which are purely RISC and those which are purely CISC
- CISC instructions are converted to a sequence of RISC-like operations processed by the RISC-like hardware
- Taking advantage of both architectures

Contents

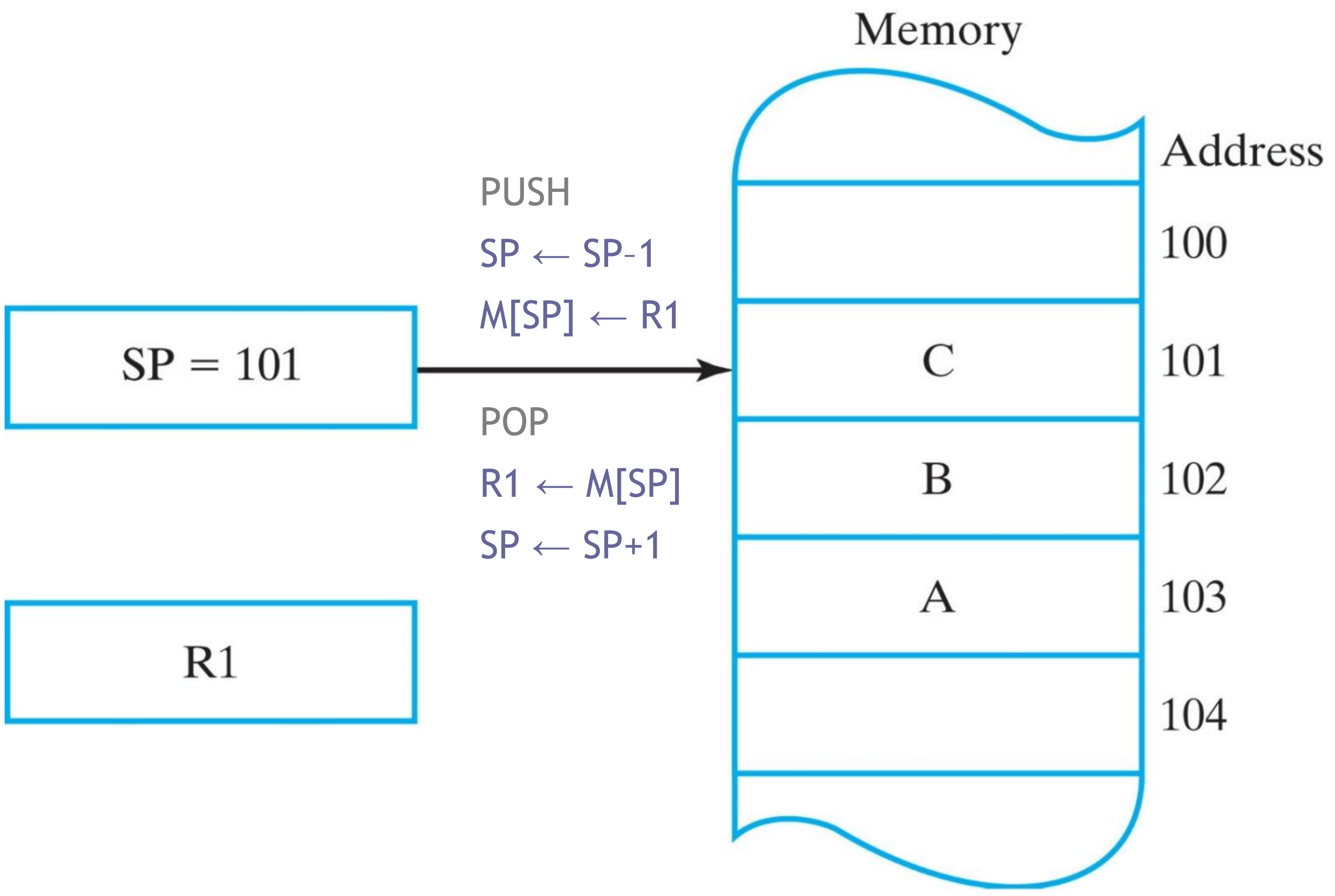
Computer Instruction Classification

- Data Transfer instructions
  - cause transfer of data from one location to another without changing the binary information content
- Data manipulation instructions
  - perform arithmetic, logic, and shift operations
- Program control instructions
  - provide decision-making capabilities and change the path taken by the program when executed in the computer
- Other instructions to provide special operations for particular applications

Typical Data Transfer Instructions

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

Memory Stack



PUSH

SP ← SP-1

M[SP] ← R1

POP

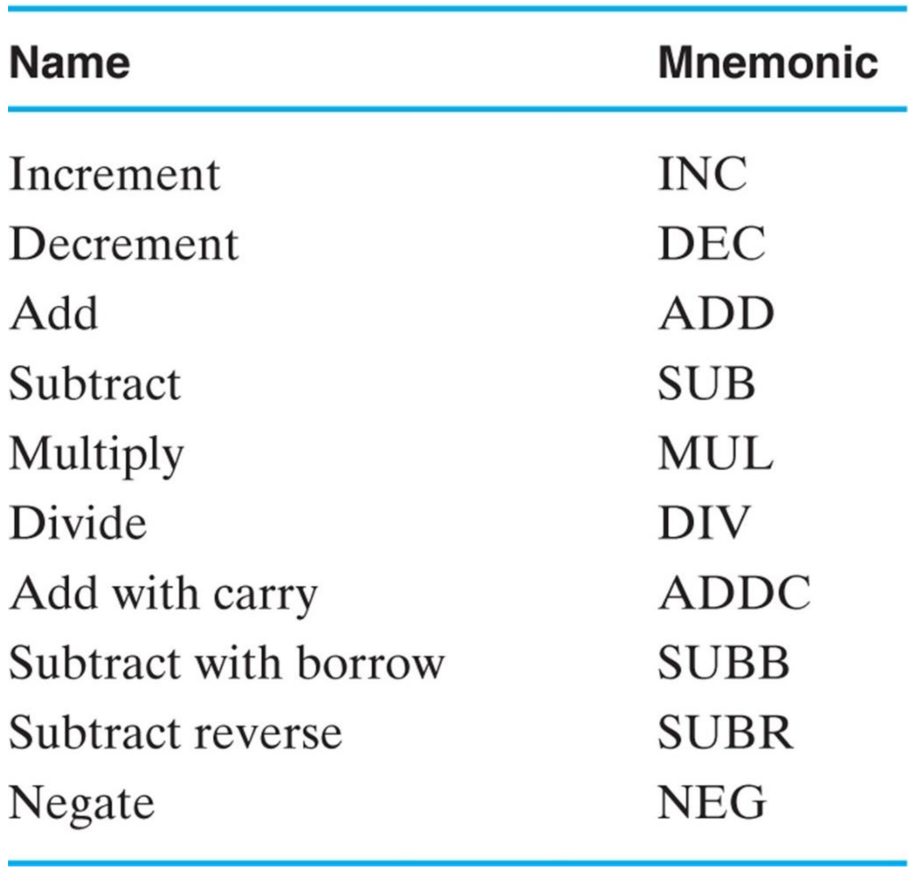
R1 ← M[SP]

SP ← SP+1

Data Manipulation Instructions

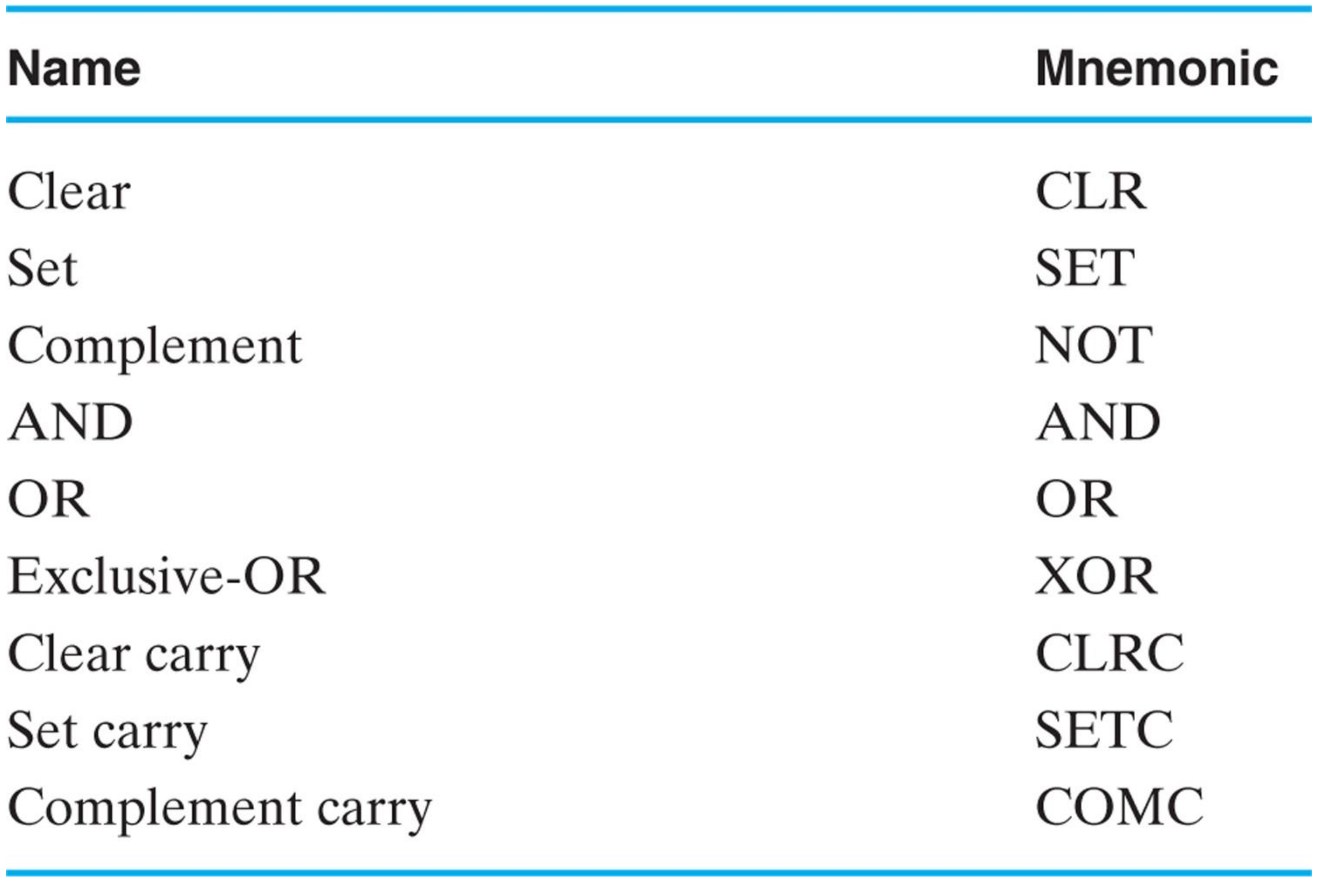
- Arithmetic instructions
- Logic and bit-manipulation instructions
- Shift instructions

Typical Arithmetic Instructions



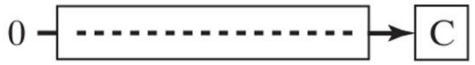
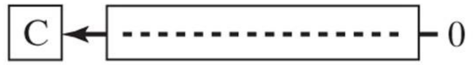
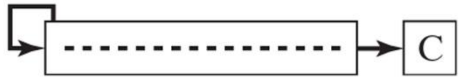
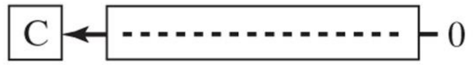
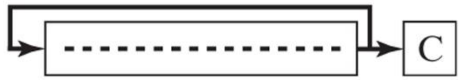
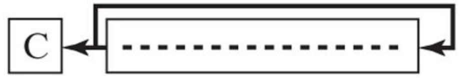
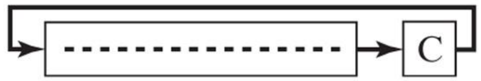
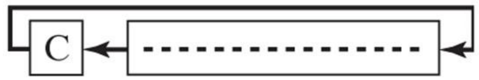
Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with carry	ADDC
Subtract with borrow	SUBB
Subtract reverse	SUBR
Negate	NEG

Typical Logical & Bit-Manipulation Instructions



Name	Mnemonic
Clear	CLR
Set	SET
Complement	NOT
AND	AND
OR	OR
Exclusive-OR	XOR
Clear carry	CLRC
Set carry	SETC
Complement carry	COMC

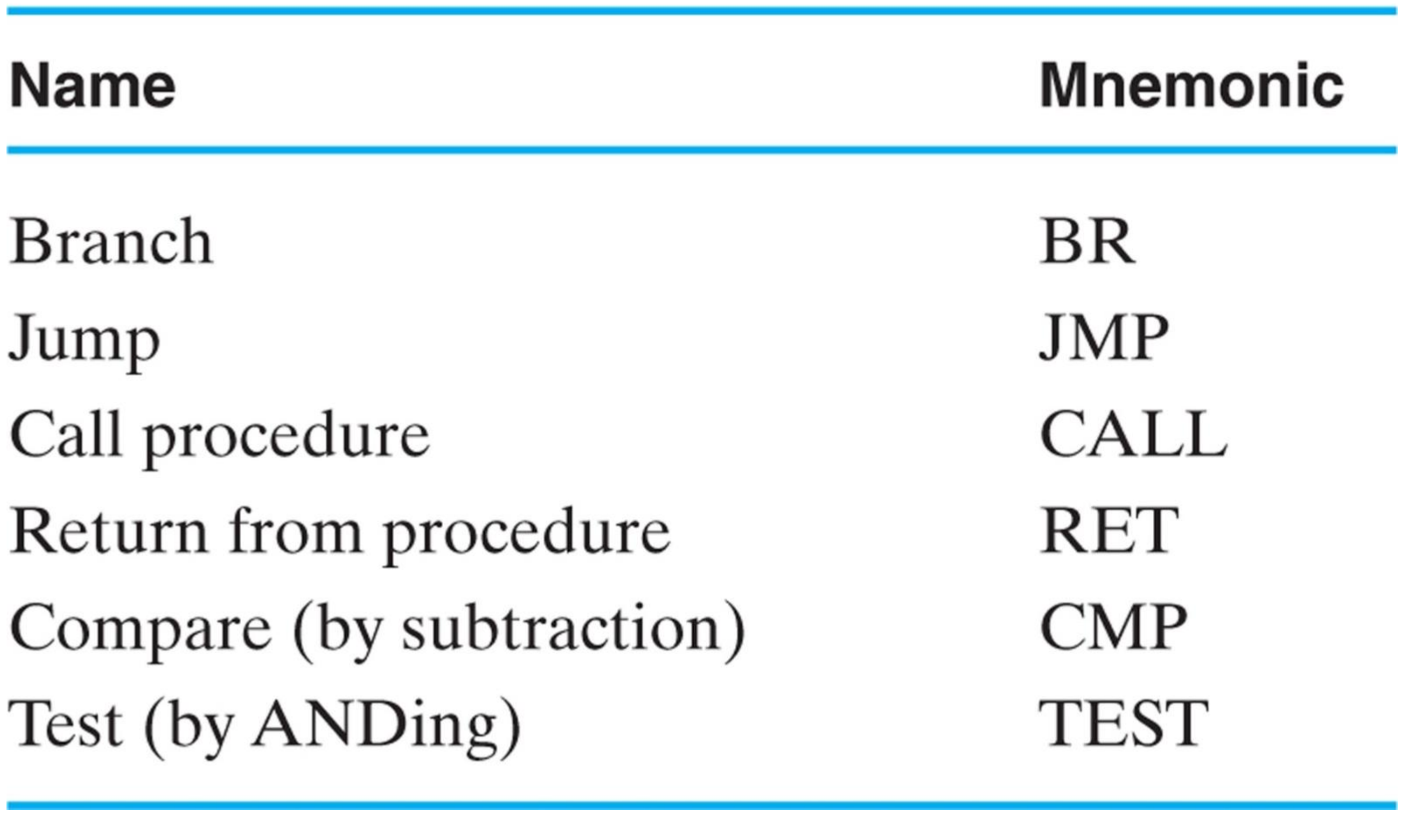
Typical Shift Instructions

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

Computer Instruction Classification

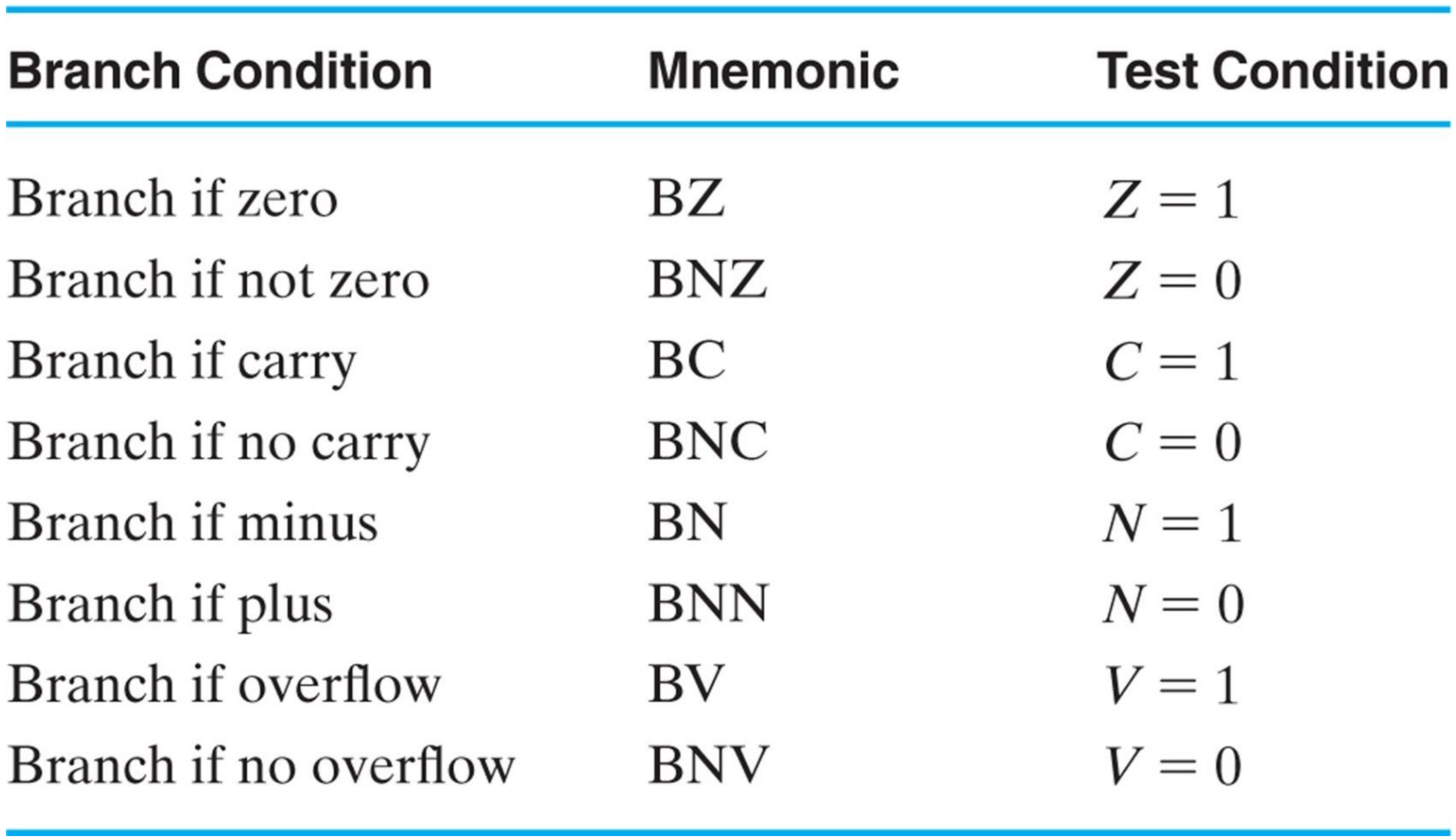
- Data Transfer instructions
  - cause transfer of data from one location to another without changing the binary information content
- Data manipulation instructions
  - perform arithmetic, logic, and shift operations
- Program control instructions
  - provide decision-making capabilities and change the path taken by the program when executed in the computer
- Other instructions to provide special operations for particular applications

Typical Program Control Instructions



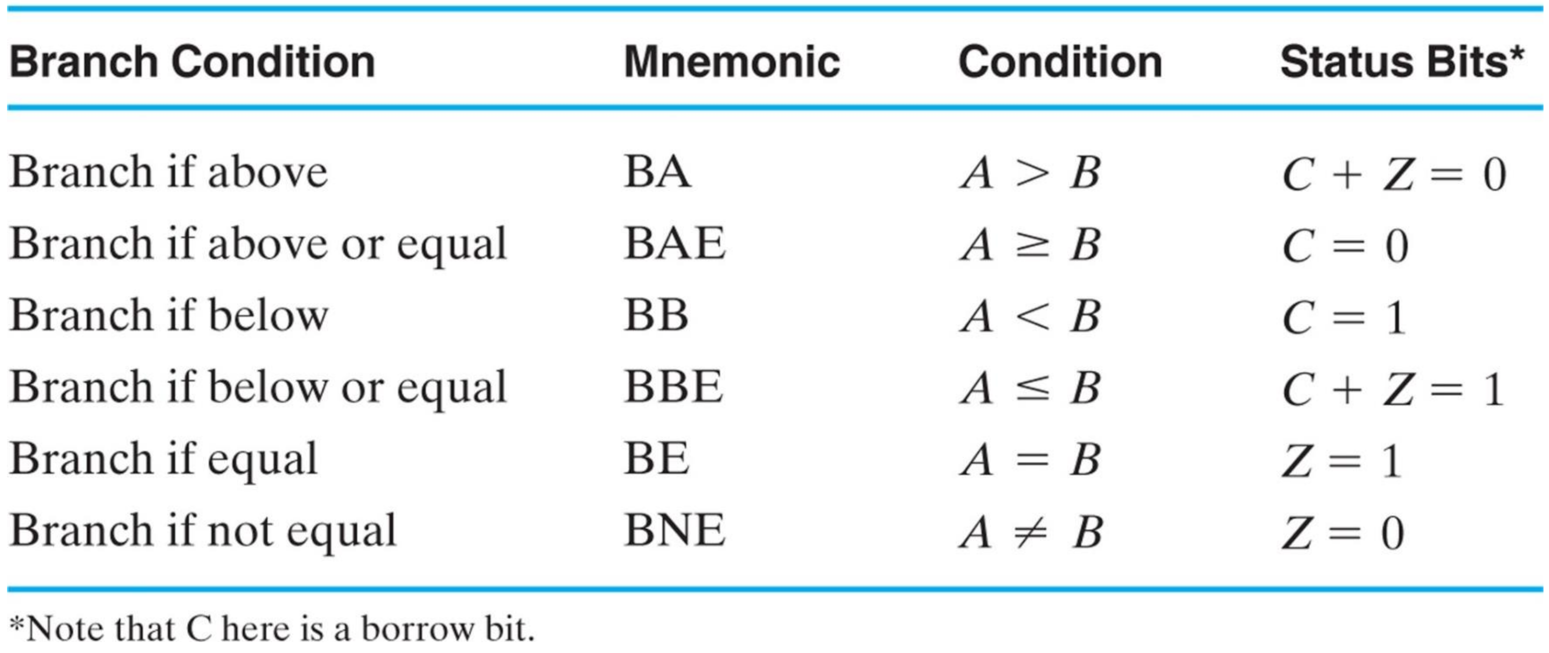
Name	Mnemonic
Branch	BR
Jump	JMP
Call procedure	CALL
Return from procedure	RET
Compare (by subtraction)	CMP
Test (by ANDing)	TEST

Conditional Branch Instructions



Branch Condition	Mnemonic	Test Condition
Branch if zero	BZ	$Z = 1$
Branch if not zero	BNZ	$Z = 0$
Branch if carry	BC	$C = 1$
Branch if no carry	BNC	$C = 0$
Branch if minus	BN	$N = 1$
Branch if plus	BNN	$N = 0$
Branch if overflow	BV	$V = 1$
Branch if no overflow	BNV	$V = 0$

Cond. Branch Instr. (Unsigned Numbers)



Branch Condition	Mnemonic	Condition	Status Bits*
Branch if above	BA	$A > B$	$C + Z = 0$
Branch if above or equal	BAE	$A \geq B$	$C = 0$
Branch if below	BB	$A < B$	$C = 1$
Branch if below or equal	BBE	$A \leq B$	$C + Z = 1$
Branch if equal	BE	$A = B$	$Z = 1$
Branch if not equal	BNE	$A \neq B$	$Z = 0$

*Note that C here is a borrow bit.

Cond. Branch Instr. (Signed Numbers)

Branch Condition	Mnemonic	Condition	Status Bits
Branch if greater	BG	$A > B$	$(N \oplus V) + Z = 0$
Branch if greater or equal	BGE	$A \geq B$	$N \oplus V = 0$
Branch if less	BL	$A < B$	$N \oplus V = 1$
Branch if less or equal	BLE	$A \leq B$	$(N \oplus V) + Z = 1$
Branch if equal	BE	$A = B$	$Z = 1$
Branch if not equal	BNE	$A \neq B$	$Z = 0$

Outlines

- Computer Layers of Abstraction
- Instruction Set Architecture
- Addressing Architectures
- Addressing Modes
- Instruction Formats
- CISC/RISC Instruction Sets
- Computer Instruction Classification