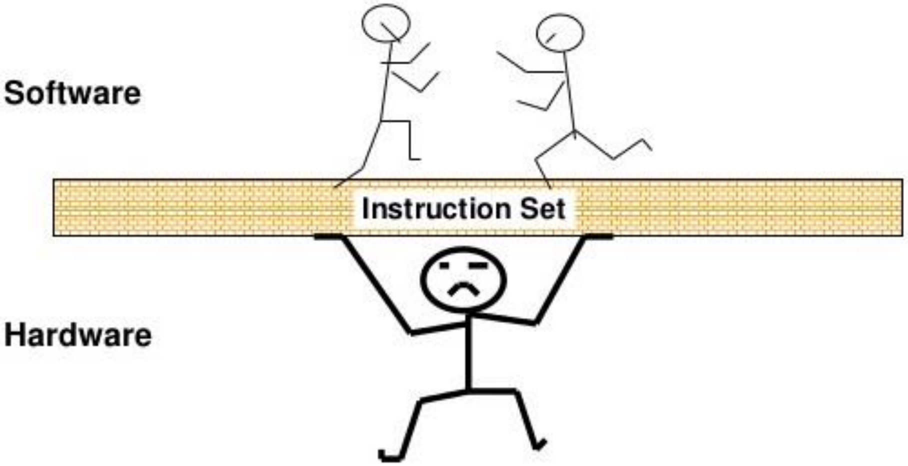
ساختار و زبان کامپیوتر

فصل سوم

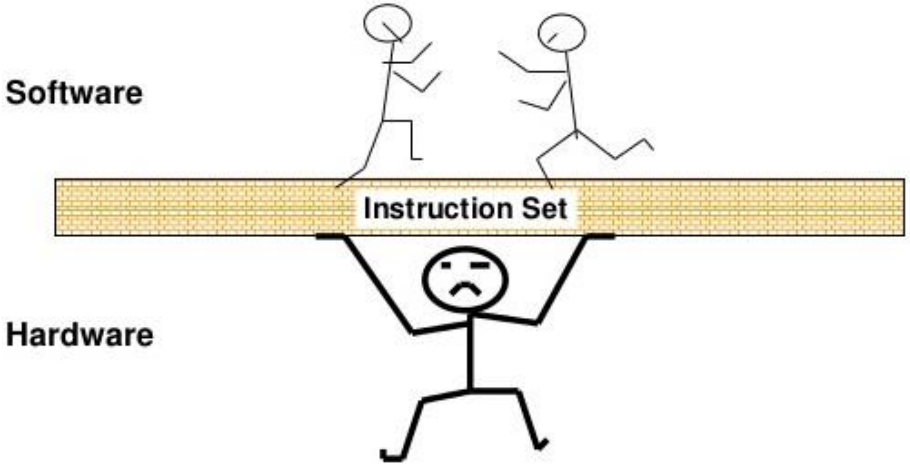
معماری مجموعه دستورالعمل‌ها



Computer Structure and Machine Language

Chapter Three

Instruction Set Architecture



Copyright Notice

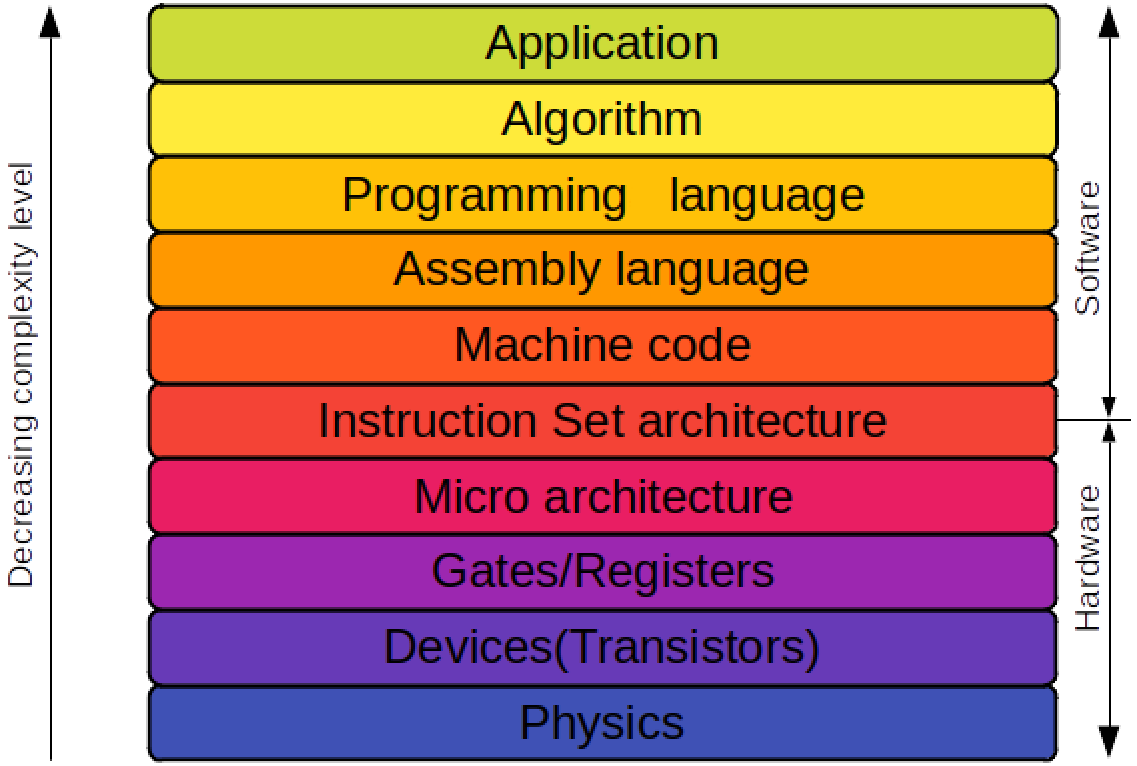
Parts (text & figures) of this lecture are adopted from:

- M. M. Mano, C. R. Kime & T. Martin, "Logic & Computer Design Fundamentals", 5th Ed., Pearson, 2015
- D. Patterson & J. Hennessey, "Computer Organization & Design, The Hardware/Software Interface, MIPS Edition", 6th Ed., MK publishing, 2020
- A. Tanenbaum, "Structured Computer Organization", 5th Ed., Pearson, 2006

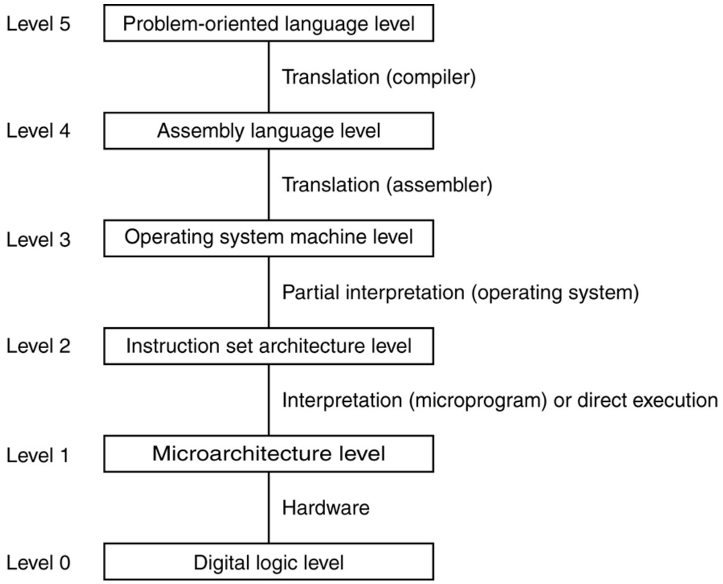
Contents

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

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

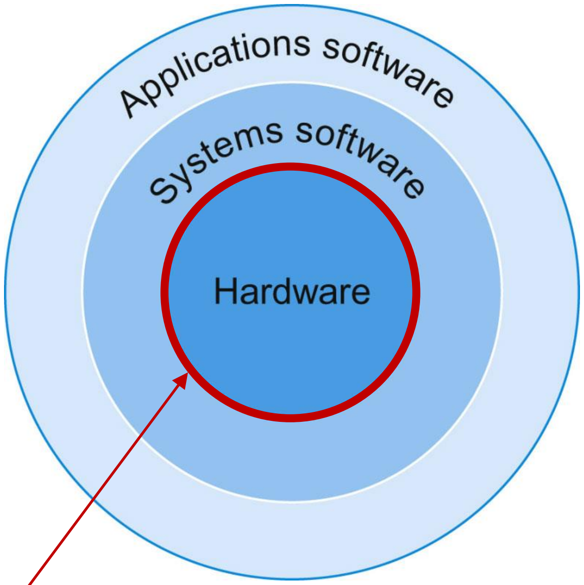
Interpretation (microprogram) or direct execution

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

- Issues not part of ISA (not visible to the 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

- ○ Introduction
- ○ Operand Addressing
  - Addressing Architectures
  - Addressing Modes
- ○ Instruction Sets

Where Operands Reside?

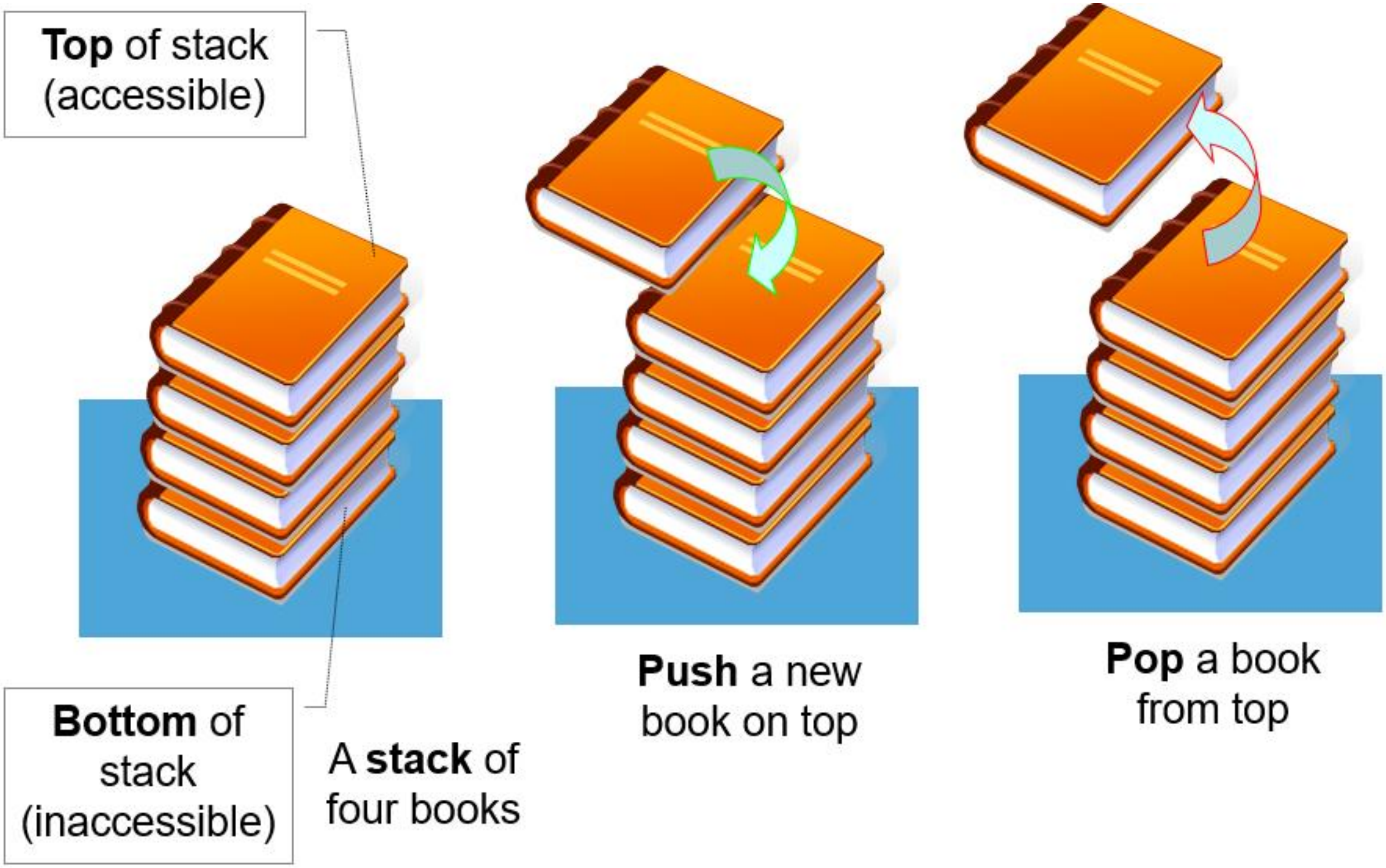
○ Stack Machine

○ Accumulator Machine

○ Register-Memory Machine

○ Register-Register Machine (Load-Store)

Stack Illustration



Top of stack (accessible)

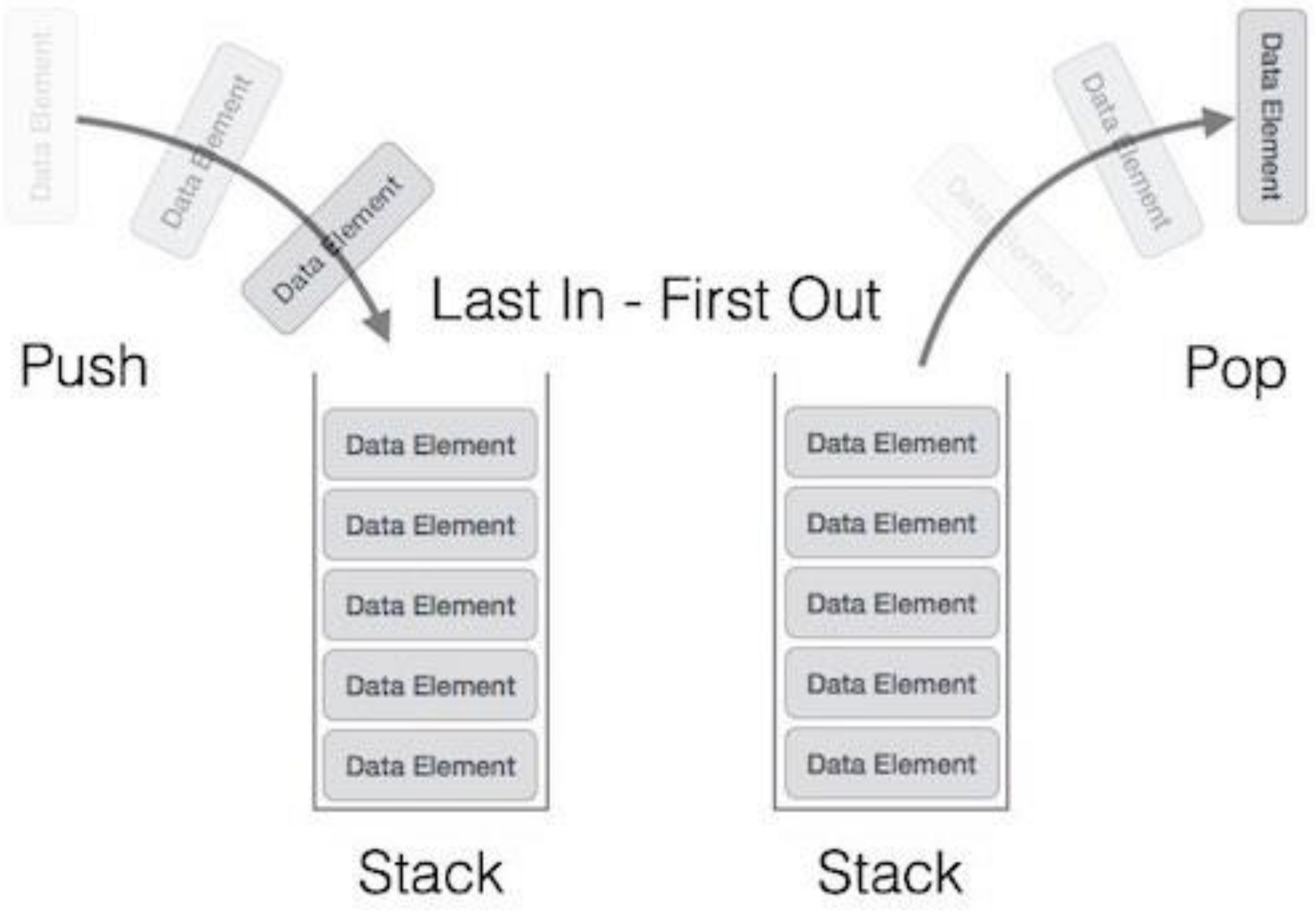
Bottom of stack (inaccessible)

A **stack** of four books

Push a new book on top

Pop a book from top

Stack Representation



Stack Machine

- "Zero-operand" ISA
  - ALU operations (add, sub, …) don't need any operands
- "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

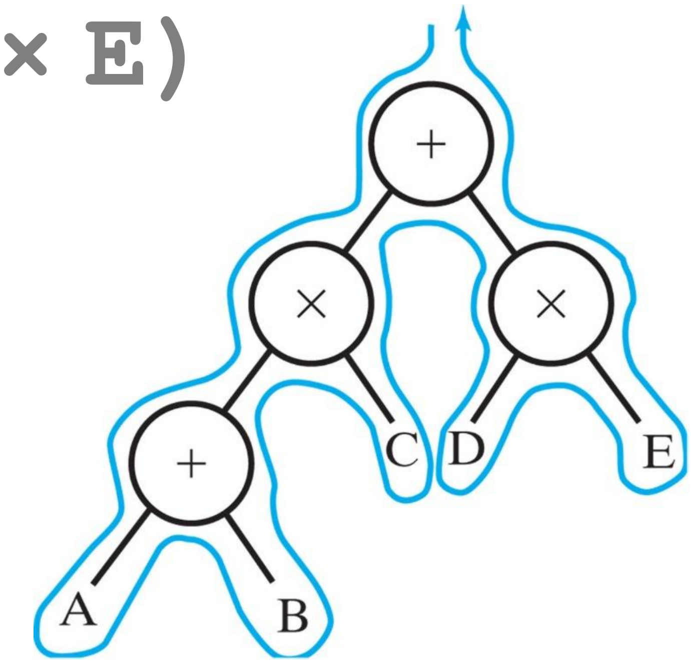
Code sequence for **`C = A + B`**

Stack	Accumulator	Register-Memory	Reg-Reg
`Push A`			
`Push B`			
`Add`			
`Pop  C`			

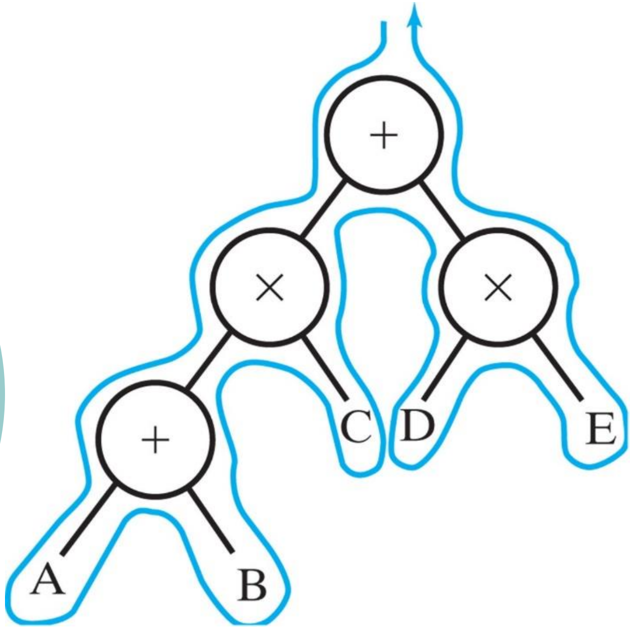
Postfix Notation

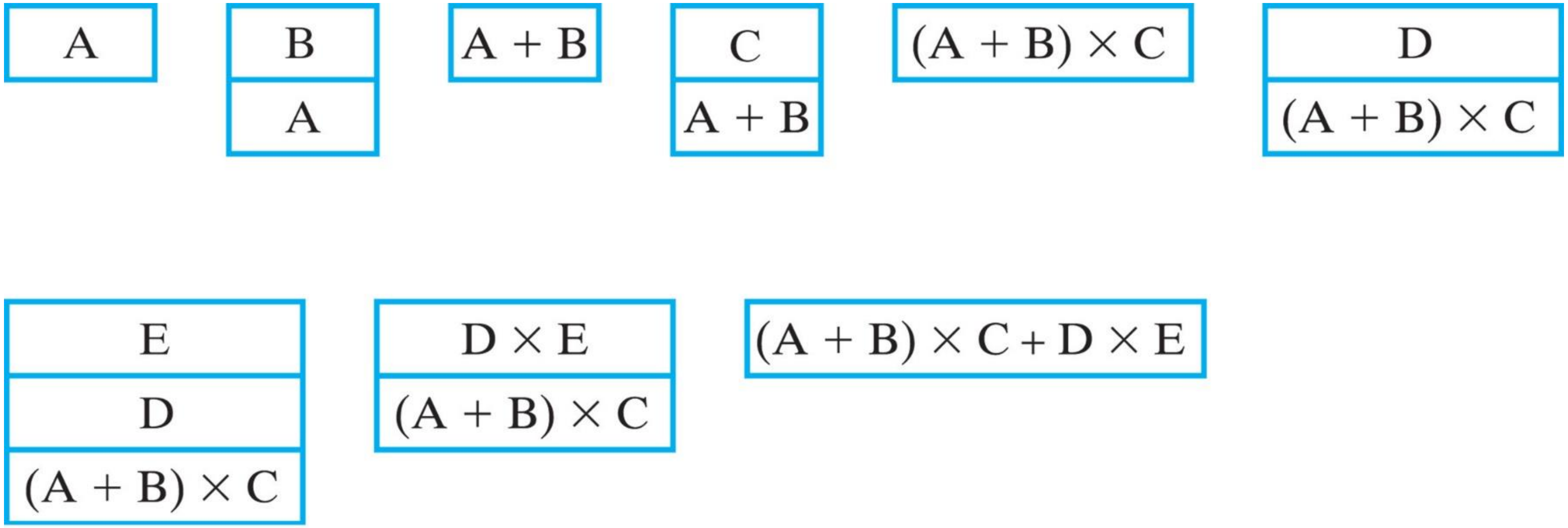
(A + B) × C + (D × E)

A B + C × D E × +



A B + C × D E × +





A

B
A

A + B

C
A + B

(A + B) × C

D
(A + B) × C

E
D
(A + B) × C

D × E
(A + B) × C

(A + B) × C + D × E

Accumulator Machine

- "1-operand" ISA
- Only 1 register called "accumulator"
- Stores intermediate arithmetic & logic results
- Instructions include:
  - "STORE" (Store AC)
  - "LOAD" (Load AC)
  - "ADD mem" (AC ← AC + mem)

Example 1

Code sequence for `C = A + B`

Stack Accumulator Register-Memory Reg-Reg

?

Instruction Set Architecture

Example 1-2

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

Stack	Accumulator	Register-Memory	Reg-Reg
`Push A`	**`Load  A`**		
`Push B`	**`Add   B`**		
`Add`	**`Store C`**		
`Pop  C`			

Register-Memory Machine

- 2 or 3 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
- Move operands with move instruction

Instruction Set Architecture

Example 1

Code sequence for `C = A + B`

Stack Accumulator Register-Memory Reg-Reg

?

Example 1-3

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

Stack	Accumulator	Register-Memory	Reg-Reg
`Push A`	`Load  A`	`Mov R1, A`	
`Push B`	`Add   B`	`Add R1, B`	
`Add`	`Store C`	`Mov C, R1`	
`Pop  C`			

Register-Register Machine

- Also called Load-Store Machine
- 2 or 3 operands ISA
- A set of general purpose registers
- Arithmetic & logical instructions can only access registers
- Access to memory only with Load & Store instructions

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`	`Mov R1, A`	`Load R1,A`
`Push B`	`Add B`	`Add R1, B`	`Load R2,B`
`Add`	`Store C`	`Mov C, R1`	`Add R3,R1,R2`
`Pop C`			`Store C,R3`

Example 2

Register-Memory

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

Reg-Reg

Stack

Accumulator

?

Example 2

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

Register-Memory

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

```
MOV   R1, A
ADD   R1, B
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, R3
STORE    R1
```

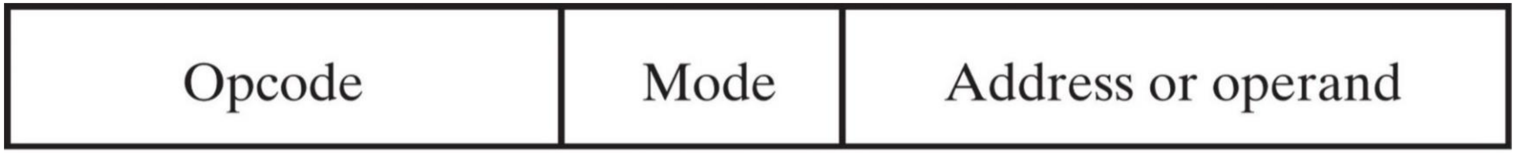
Contents

- Introduction
- Operand Addressing
  - Addressing Architectures
  - Addressing Modes
- CISC vs. RISC ISA
- Instruction Sets

Instruction Set Architecture

Definition

- Addressing Mode
  - A form for specifying one or more operands
- Effective Address
  - The address of the operand (in memory)



Opcode	Mode	Address or operand

Addressing Modes

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

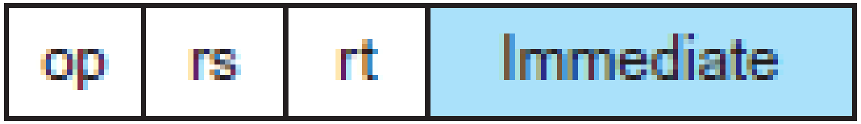
Implied Addressing

- The operand is specified implicitly in the instruction, such as:
  - Register-reference instructions that use an accumulator register
  - Zero-address instructions in a stack-organized computer
    - 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
```

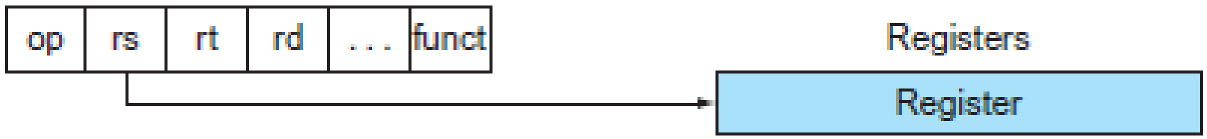


op	rs	rt	Immediate

Register (Direct) Addressing

- Operand is a register
- MIPS-32 example:

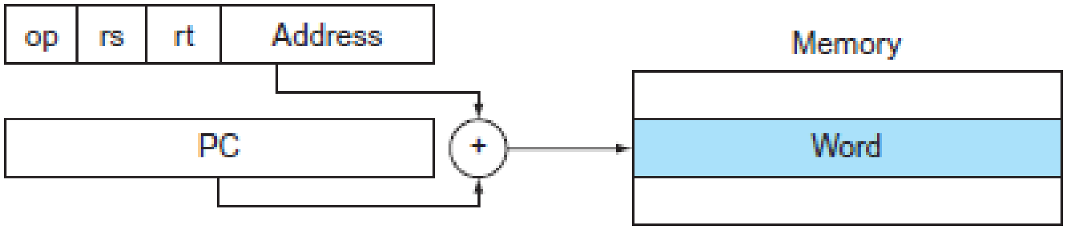
```
add $s0,$s1,$s2    # $s0 ← $s1 + $s2
```



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
```

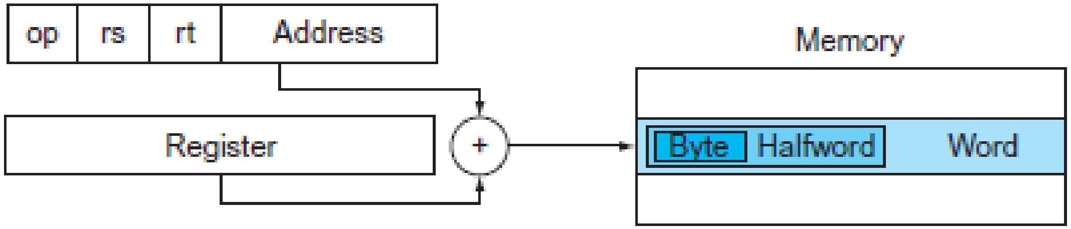


d

Base or Displacement Addressing

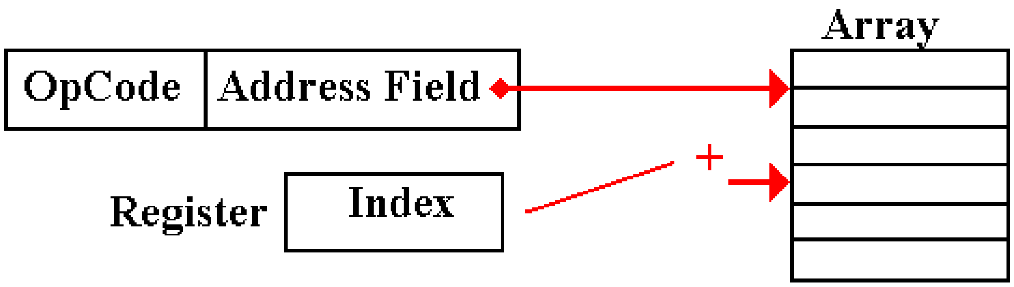
- Address of operand (in memory) is sum of a base register and a constant displacement within instruction
- MIPS-32 example:

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



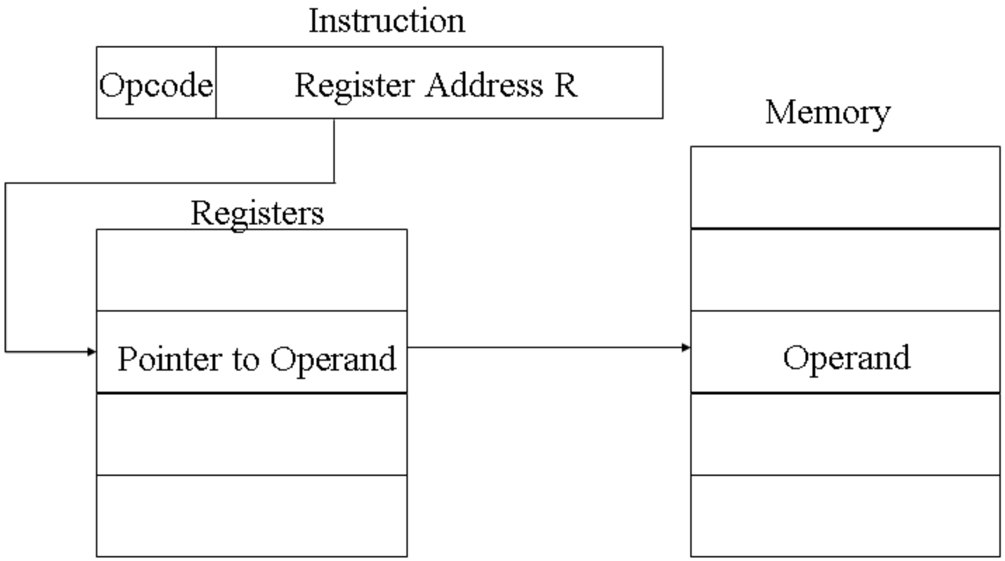
Indexed Addressing

- 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

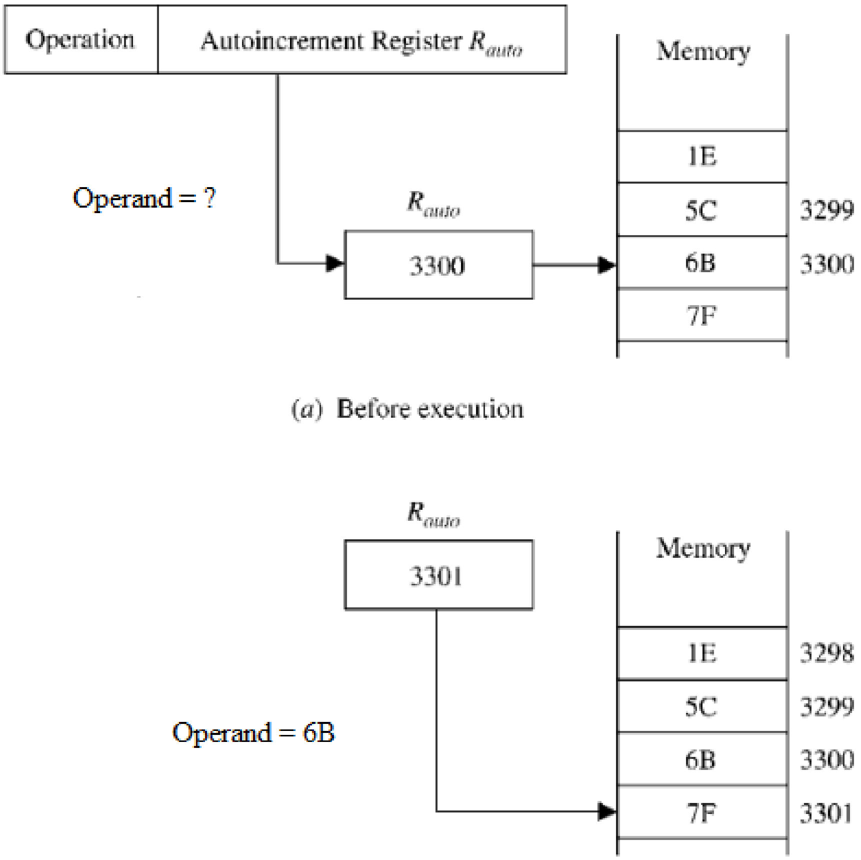


Auto-increment/Auto-decrement

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

Instruction Set Architecture

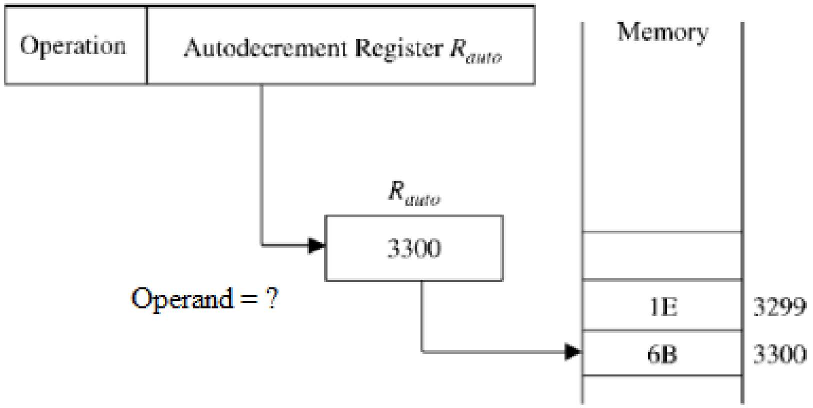
Auto-increment Addressing Mode



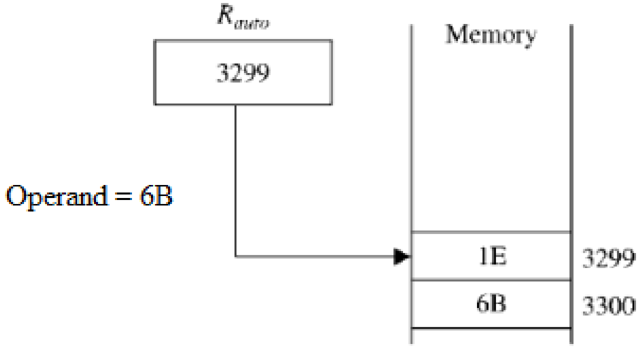
(a) Before execution

(b) After execution

Auto-decrement Addressing Mode



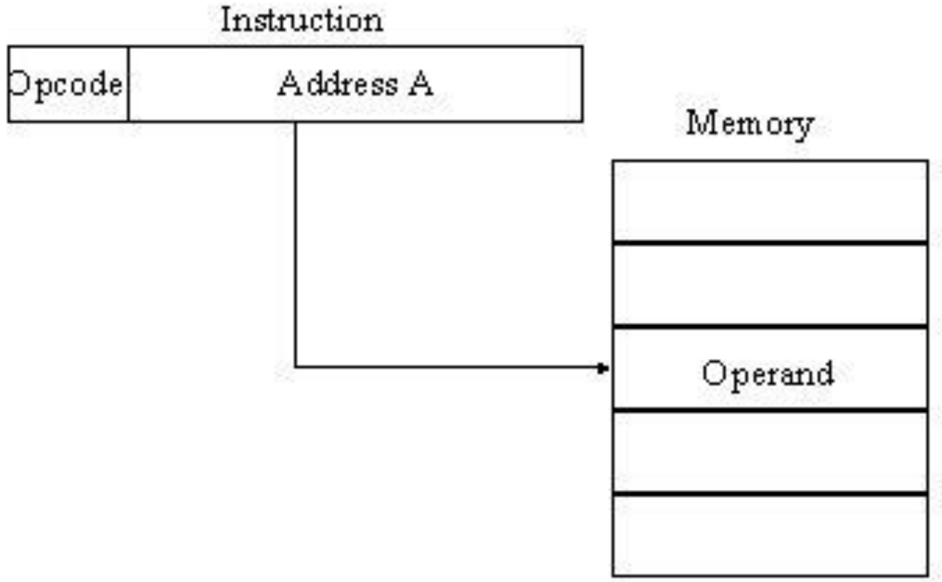
(*a*) Before execution



(*b*) After execution

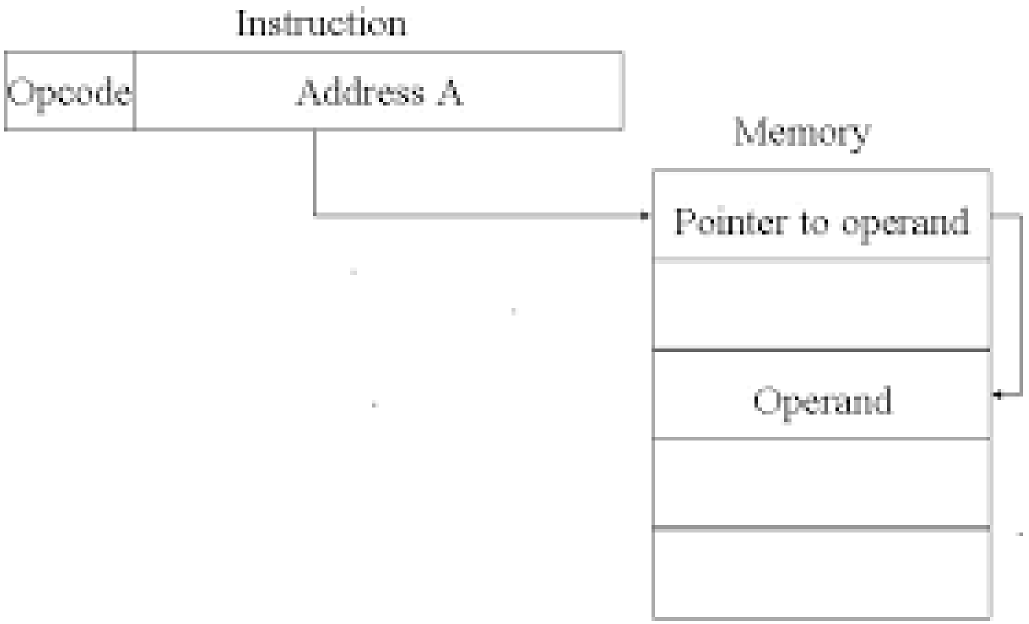
Memory Direct Addressing

- 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, 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 Convention

PC = 250

R1 = 400

ACC

Memory

250	Opcode	Mode
251	ADRS or NBR = 500	
252	Next instruction	
400	700	
500	800	
752	600	
800	300	
900	200	

			Refers to Figure 9-6	
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

- Introduction
- Operand Addressing
- Instruction Sets
  - CISC/RISC Instruction Sets
  - Computer Instruction Classification

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

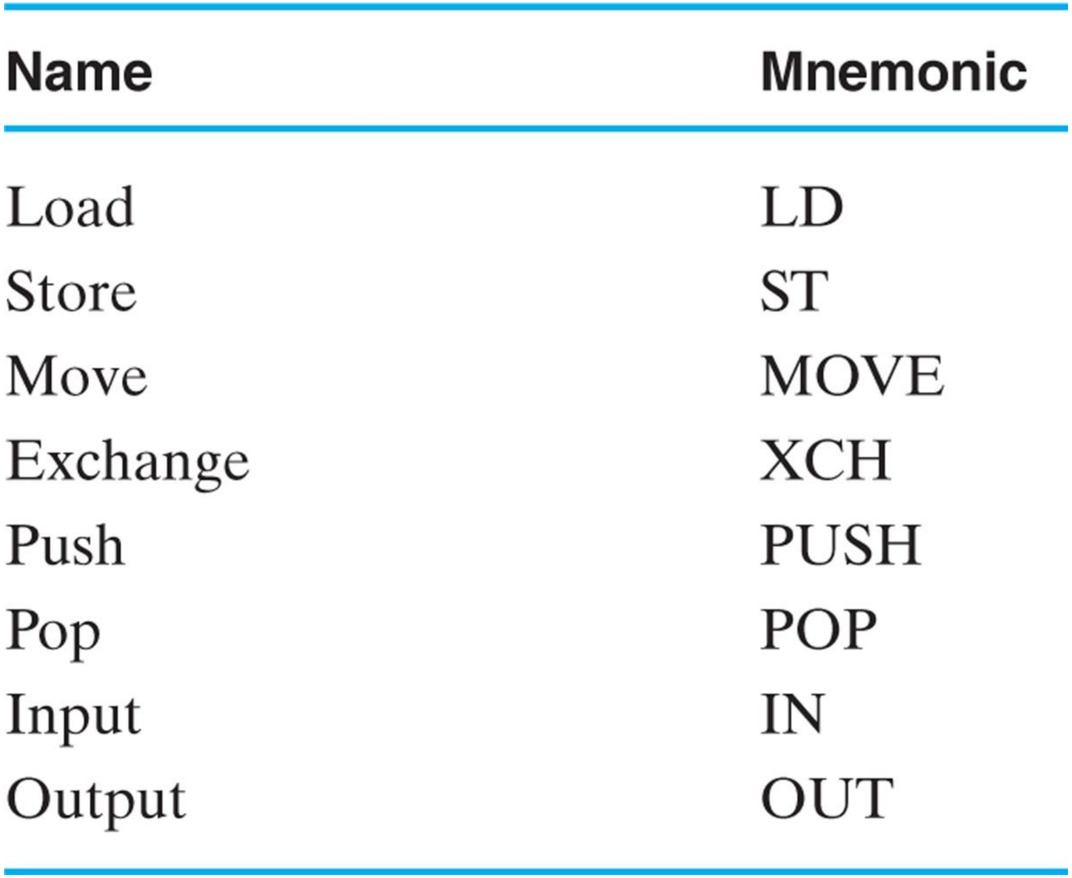
Contents

- Introduction
- Operand Addressing
- Instruction Sets
  - CISC/RISC Instruction Sets
  - Computer Instruction Classification

Computer Instruction Classification

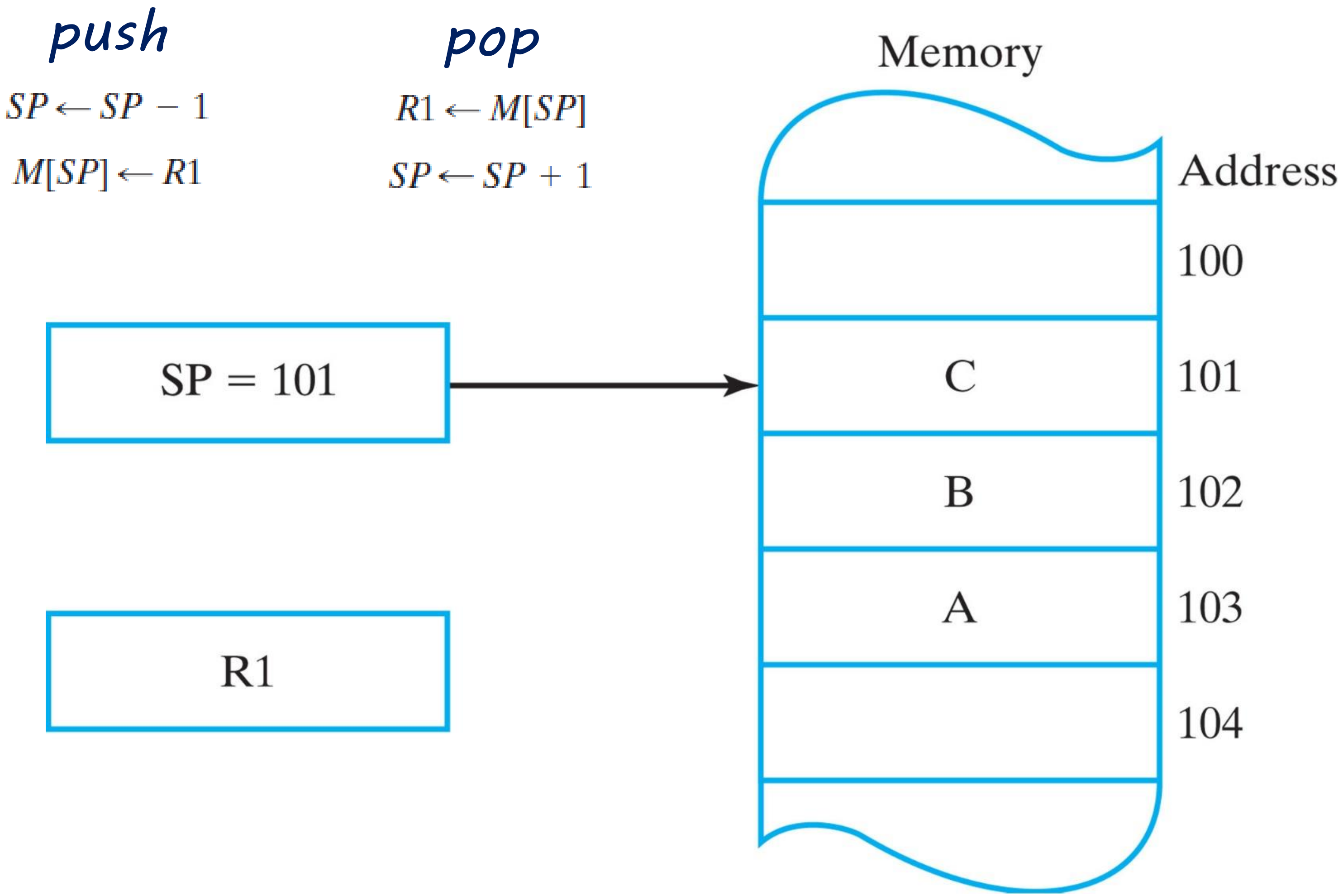
- Data Transfer instructions
  - cause transfer of data from one location to another without changing the binary information content
- Data manipulation instructions
  - perform arithmetic, logic, and shift operations
- Program control instructions
  - provide decision-making capabilities and change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

Table 9.2: Typical Data Transfer Instructions



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

Figure 9-7: Memory Stack



push

$SP \leftarrow SP - 1$

$M[SP] \leftarrow R1$

pop

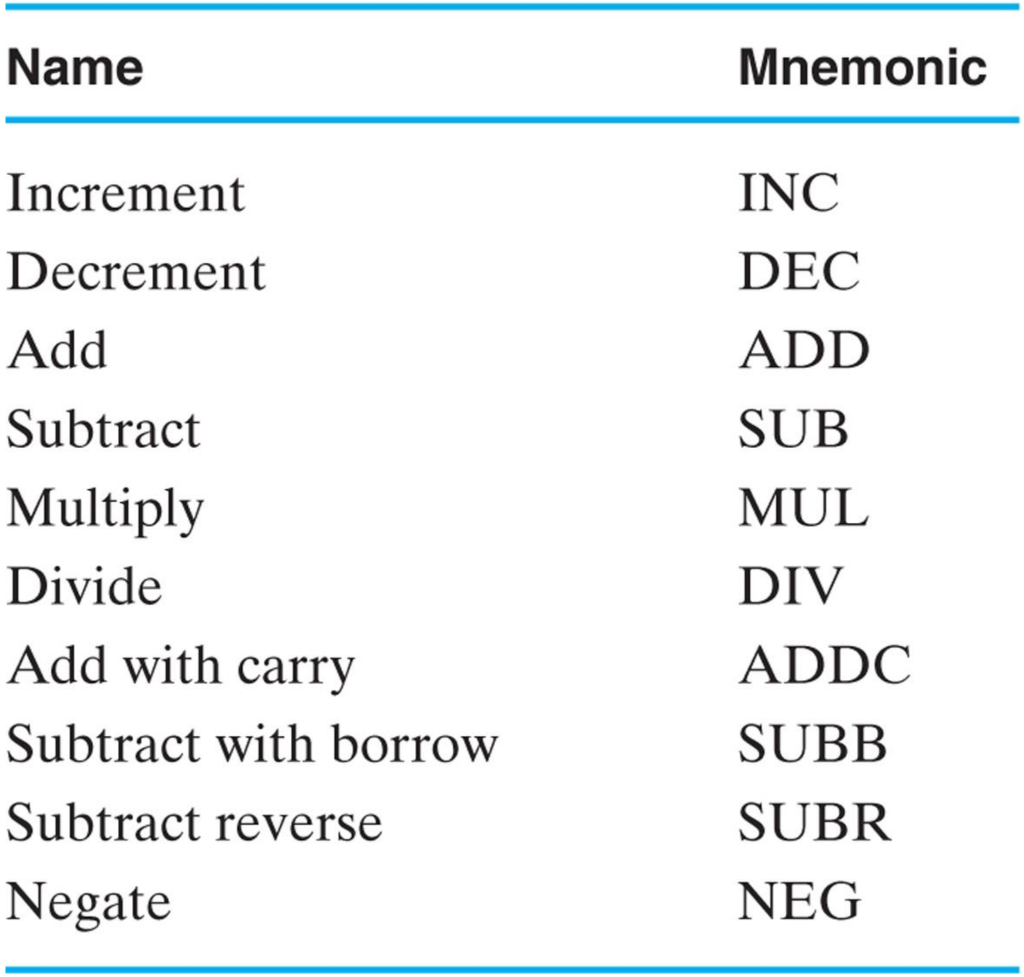
$R1 \leftarrow M[SP]$

$SP \leftarrow SP + 1$

Data Manipulation Instructions

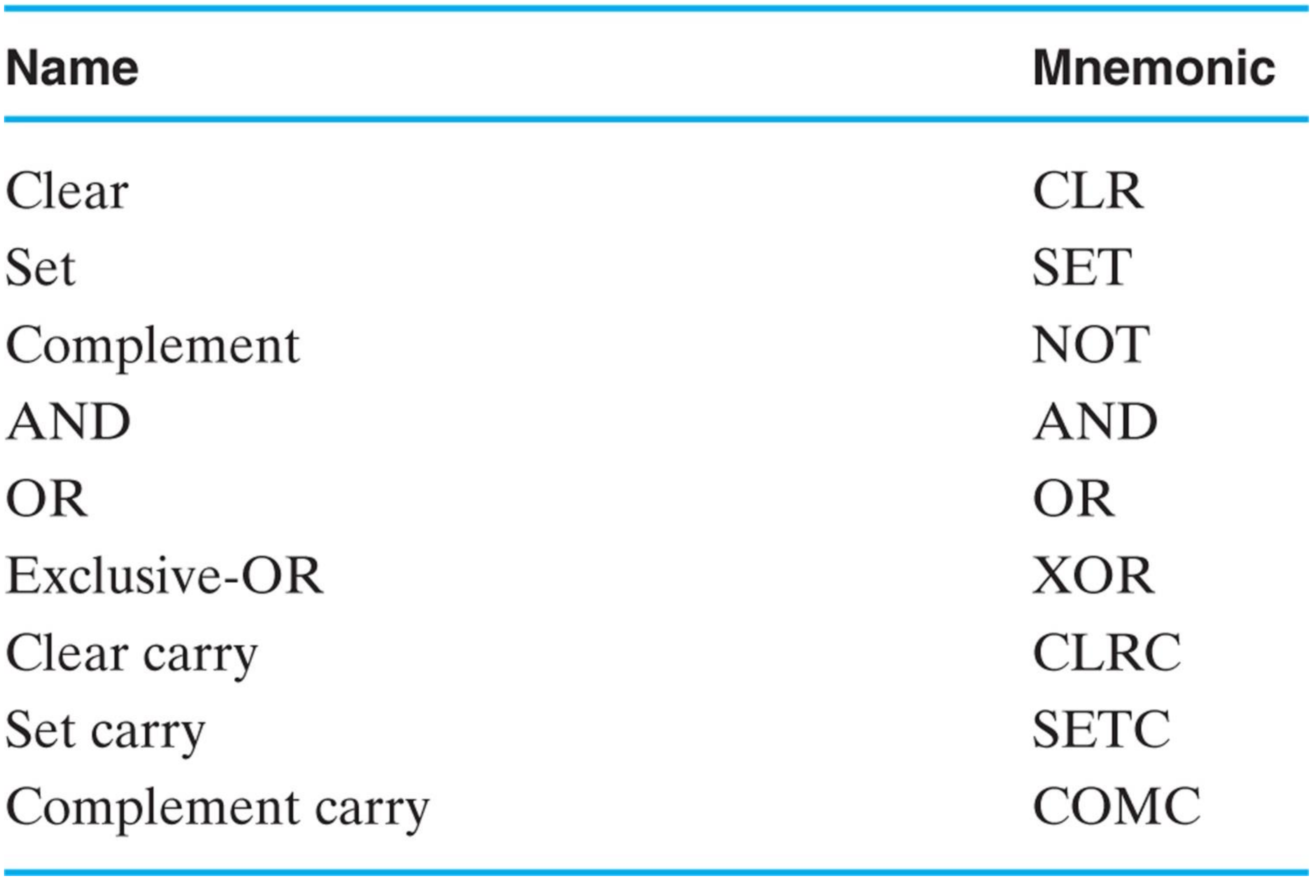
- Arithmetic instructions
- Logic and bit-manipulation instructions
- Shift instructions

Table 9.3: 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

Table 9.4: Typical Logical and 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

Table 9.5: Typical Shift Instructions

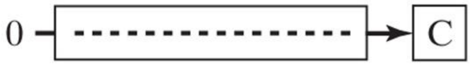
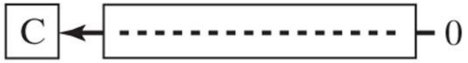
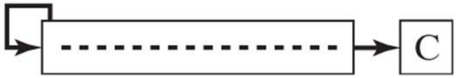
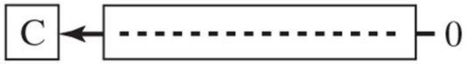
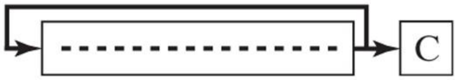
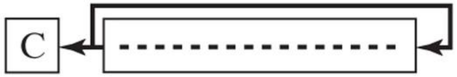
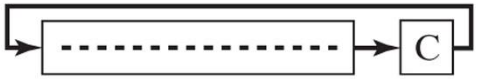
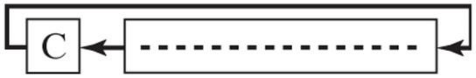
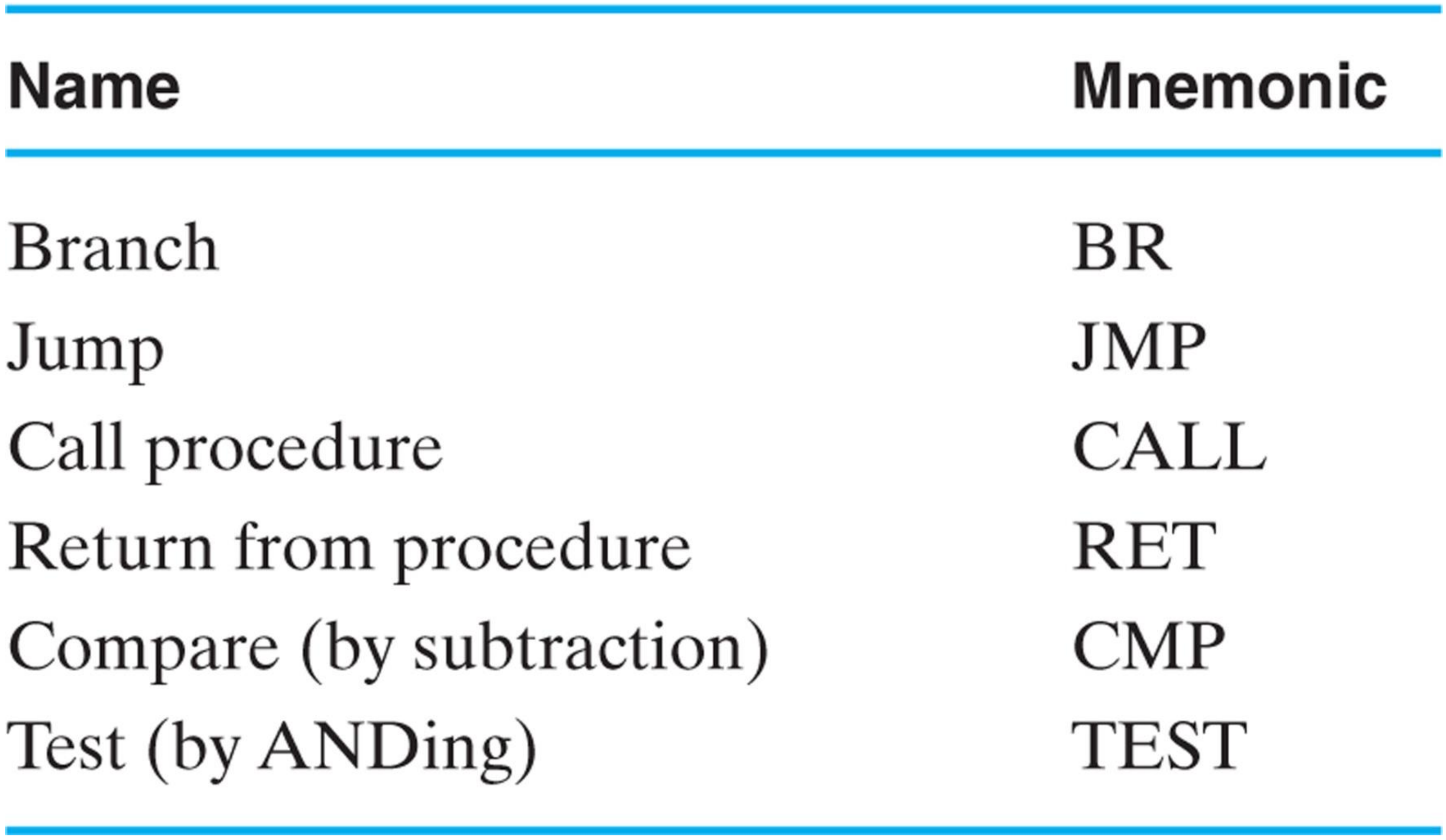
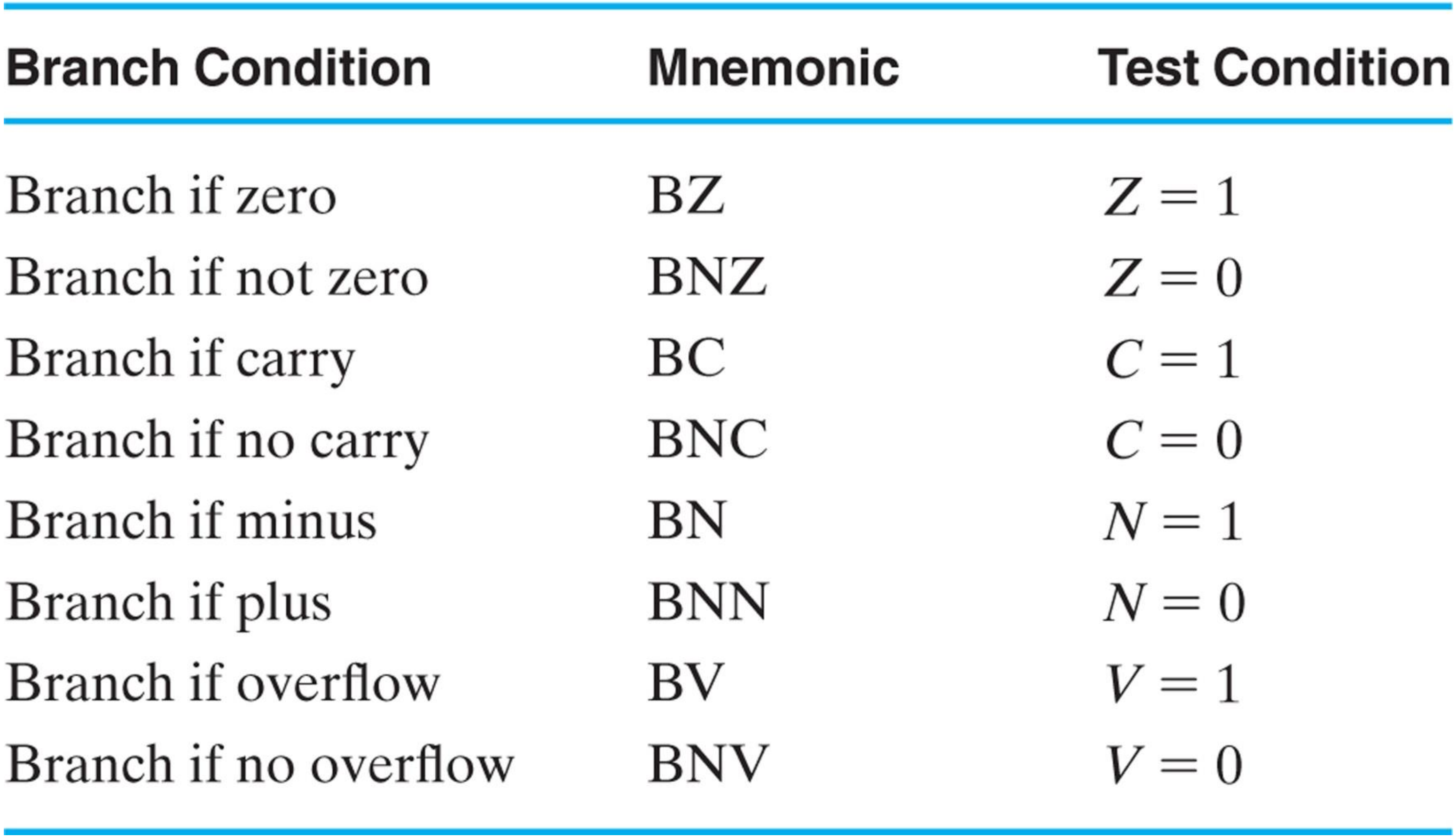
Name	Mnemonic	Diagram
Logical shift right	SHR	
Logical shift left	SHL	
Arithmetic shift right	SHRA	
Arithmetic shift left	SHLA	
Rotate right	ROR	
Rotate left	ROL	
Rotate right with carry	RORC	
Rotate left with carry	ROLC	

Table 9.7: 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

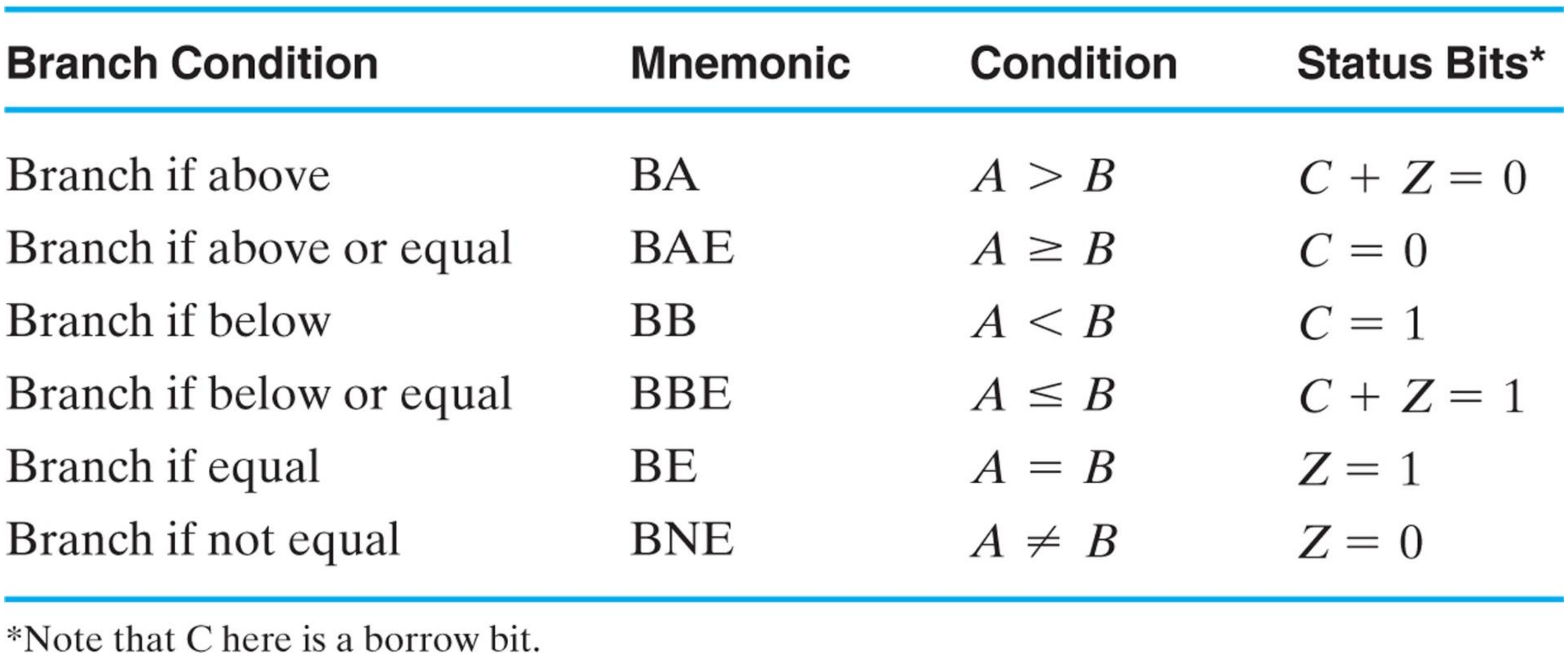
Table 9.8: Conditional Branch Instructions Relating to Status Bits in the PSR



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	$V = 0$

Pearson

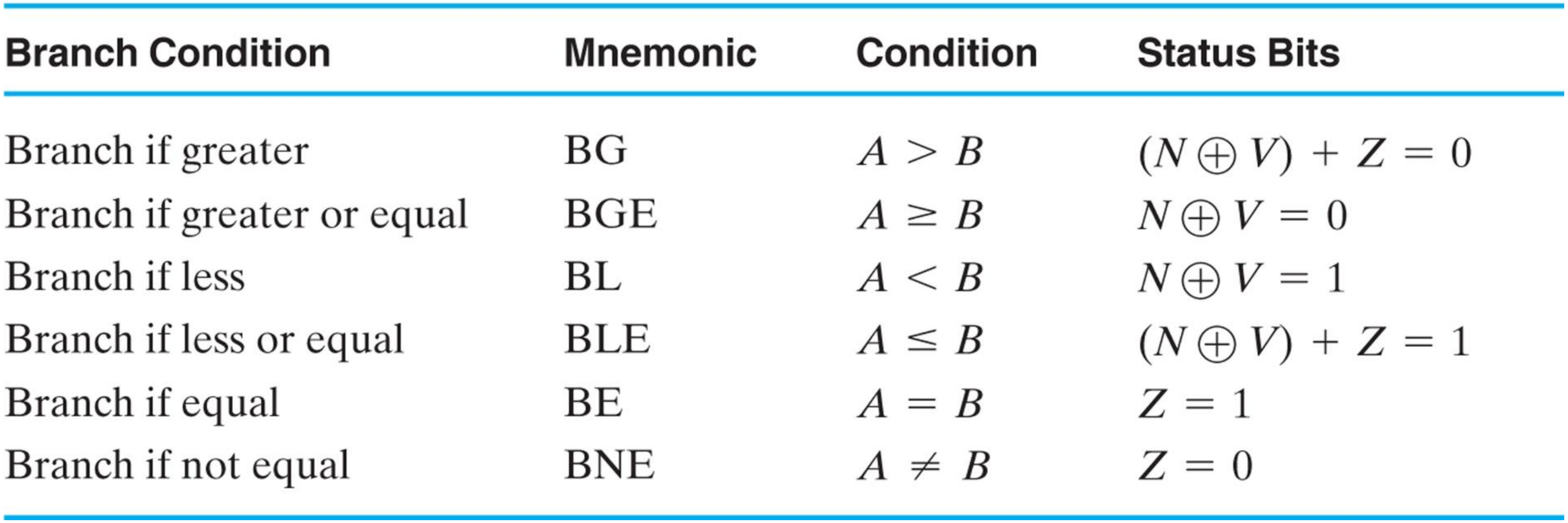
Table 9.9: Conditional Branch Instructions for Unsigned Numbers



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.

Table 9.10: Conditional Branch Instructions for 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
- CISC/RISC Instruction Sets
- Computer Instruction Classification