Types of Instructions

Types of Instructions

- Data Transfer Instructions

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

Data value is not modified

Data Transfer Instructions

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

Data Manipulation Instructions

- Arithmetic
- Logical & Bit Manipulation
- Shift

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

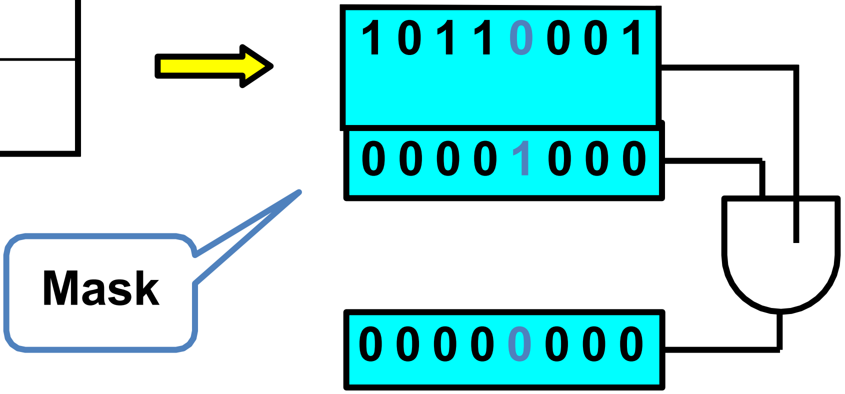
Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-OR	XOR
Clear carry	CLRC
Set carry	SETC
Complement carry	COMC
Enable interrupt	EI
Disable interrupt	DI

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

Program Control Instructions

Name	Mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RET
Compare (Subtract)	CMP
Test (AND)	TST

Subtract A – B but don’t store the result



Conditional Branch Instructions

Mnemonic	Branch Condition	Tested Condition
BZ	Branch if zero	Z = 1
BNZ	Branch if not zero	Z = 0
BC	Branch if carry	C = 1
BNC	Branch if no carry	C = 0
BP	Branch if plus	N = 0
BM	Branch if minus	N = 1
BV	Branch if overflow	V = 1
BNV	Branch if no overflow	V = 0

Basic Input/Output Operations

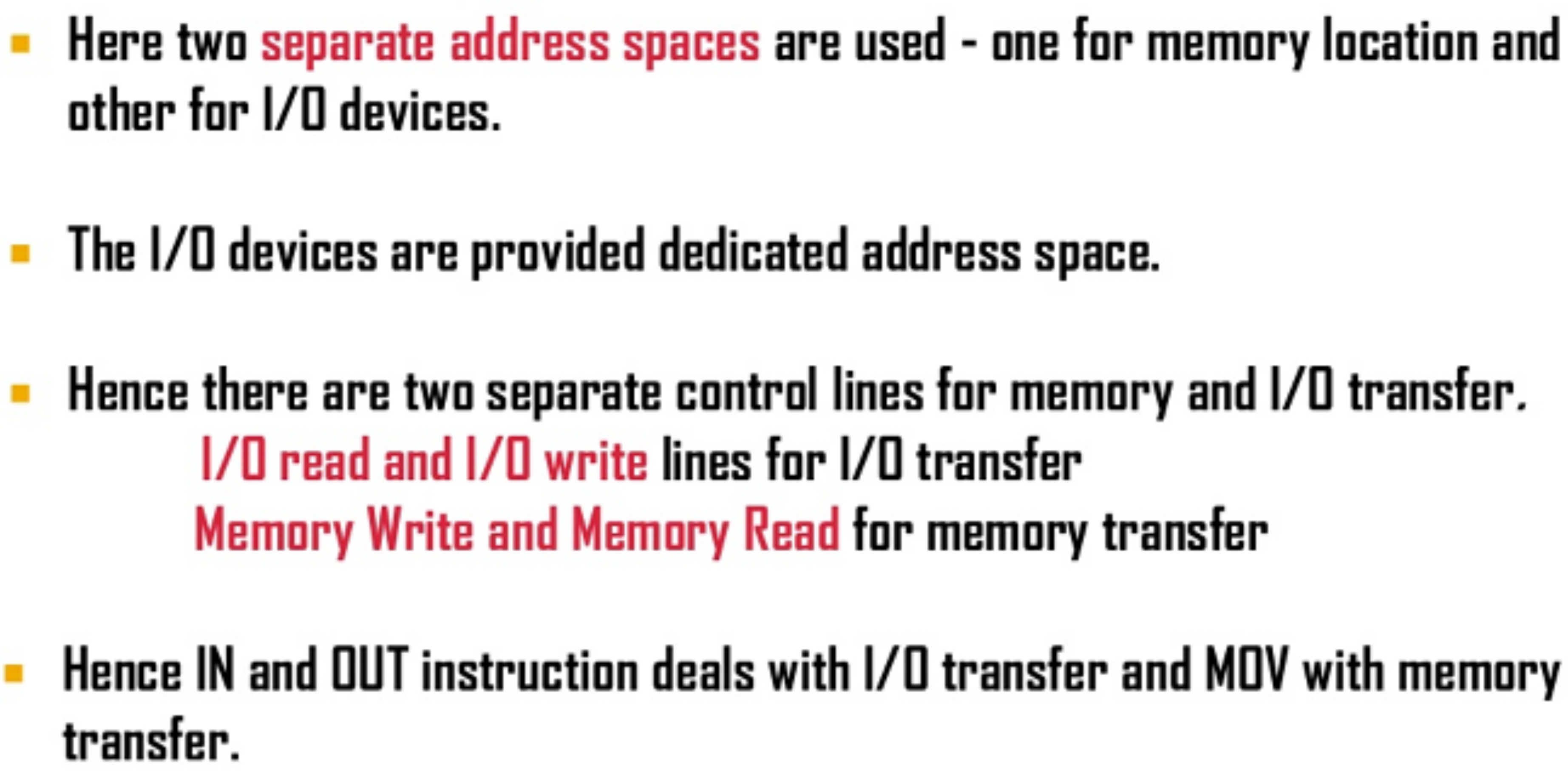
I/O

- The data on which the instructions operate are not necessarily already stored in memory.
- Data need to be transferred between processor and outside world (disk, keyboard, etc.)
- I/O operations are essential, the way they are performed can have a significant effect on the performance of the computer.

There are two techniques for addressing an I/O device by CPU:

- Memory mapped I/O
- I/O mapped I/O (Standard I/O or Isolated I/O or port I/O)

Standard IO

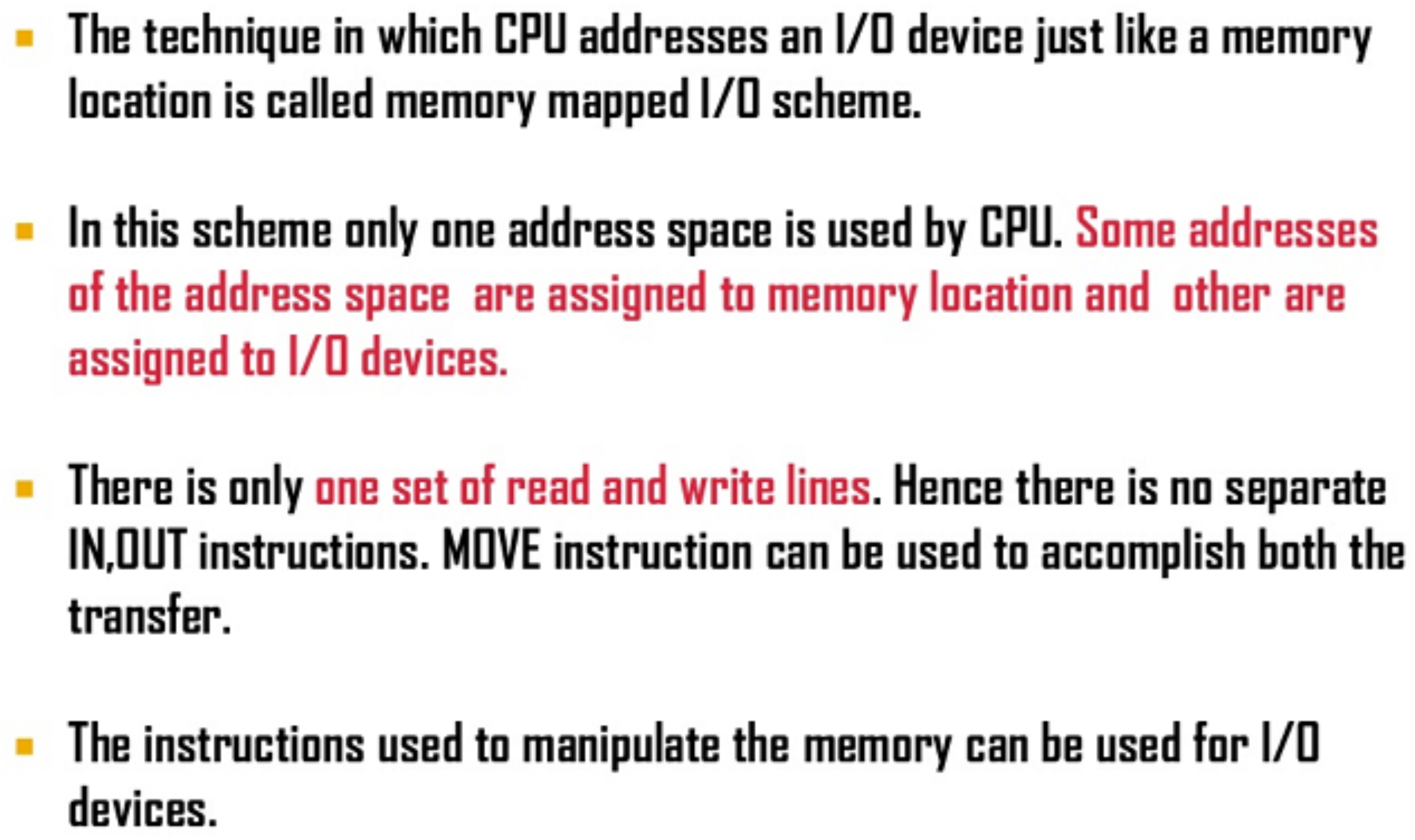


- Here two separate address spaces are used - one for memory location and other for I/O devices.
- The I/O devices are provided dedicated address space.
- Hence there are two separate control lines for memory and I/O transfer.

  I/O read and I/O write lines for I/O transfer

  Memory Write and Memory Read for memory transfer
- Hence IN and OUT instruction deals with I/O transfer and MOV with memory transfer.

Memory-Mapped I/O



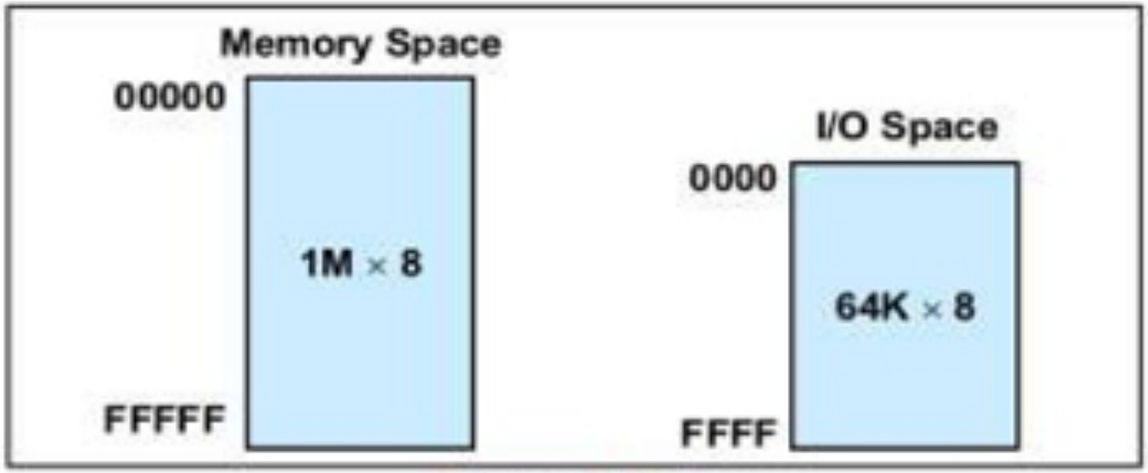
- The technique in which CPU addresses an I/O device just like a memory location is called memory mapped I/O scheme.

- In this scheme only one address space is used by CPU. Some addresses of the address space are assigned to memory location and other are assigned to I/O devices.

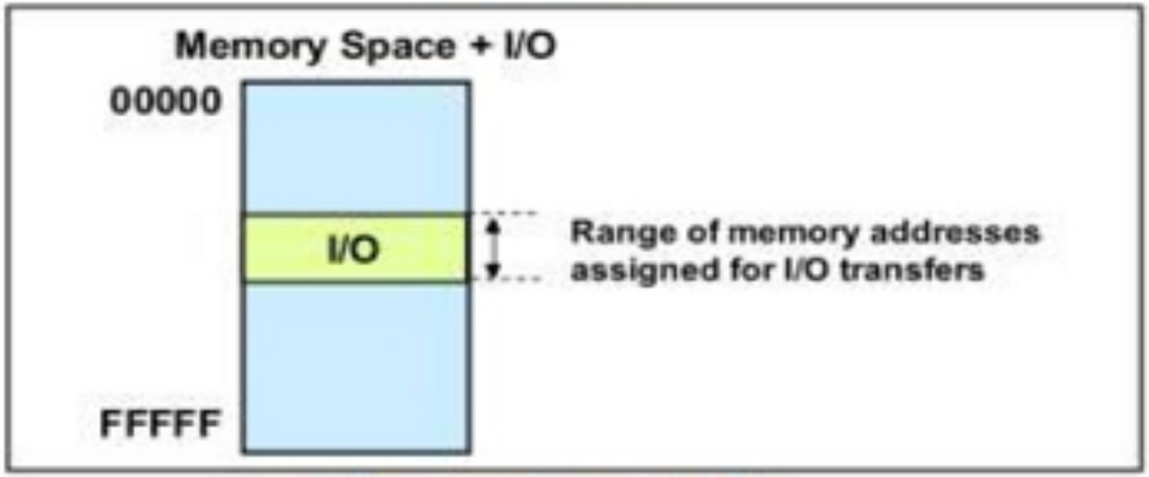
- There is only one set of read and write lines. Hence there is no separate IN,OUT instructions. MOVE instruction can be used to accomplish both the transfer.

- The instructions used to manipulate the memory can be used for I/O devices.

Isolated vs. Memory Mapped I/O

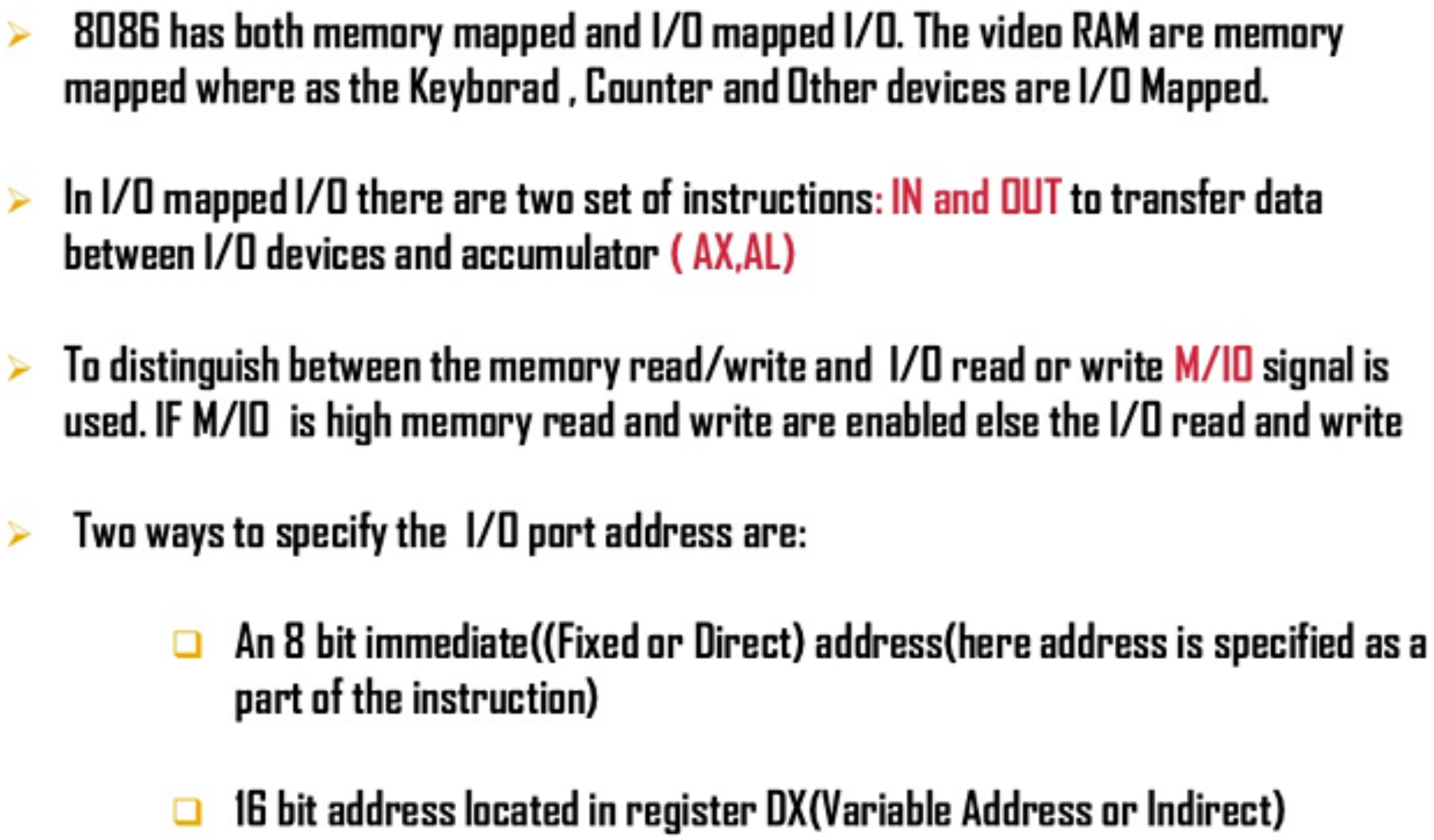


Isolated I/O



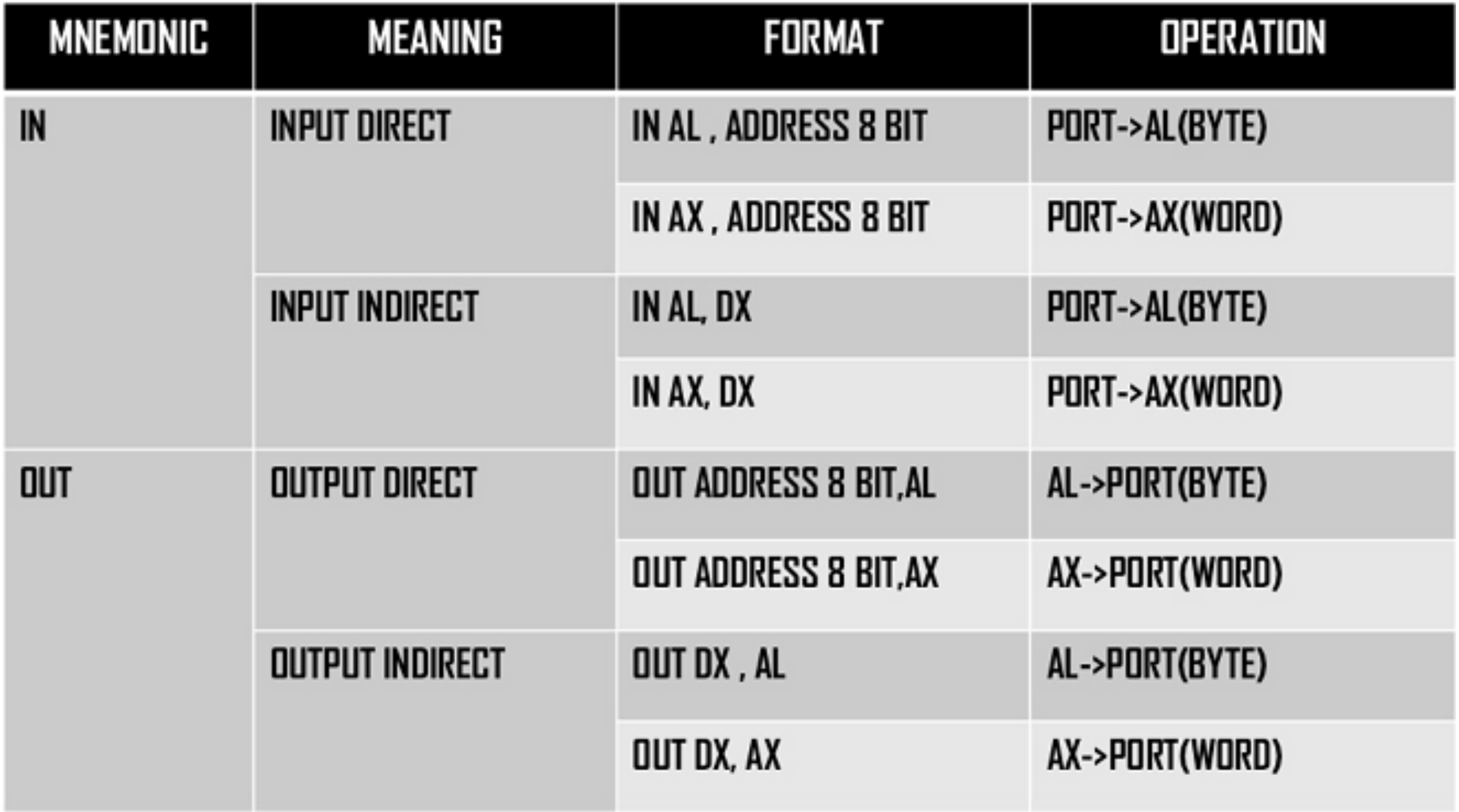
Memory-Mapped I/O

For Intel x86 Series



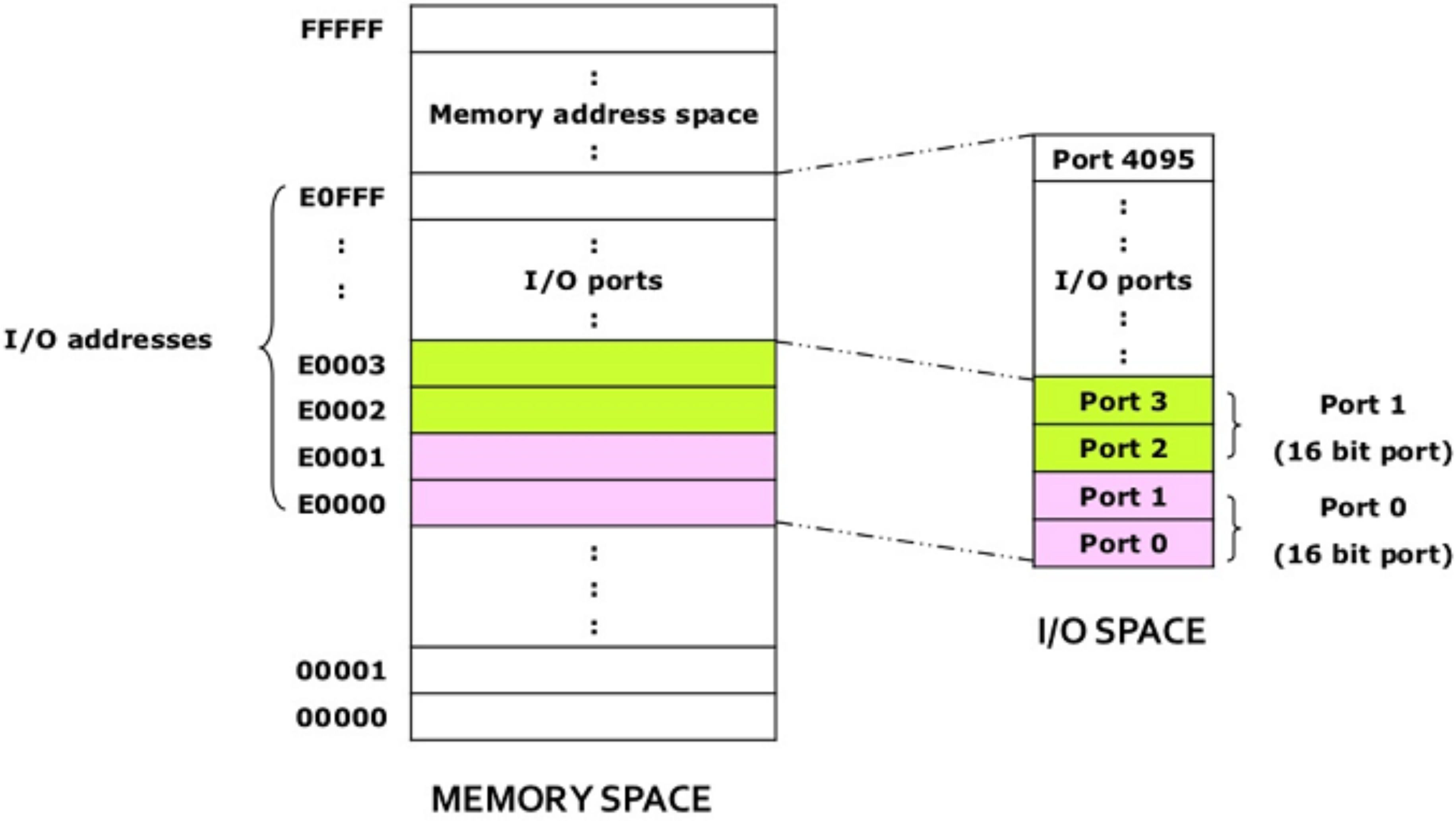
- 8086 has both memory mapped and I/O mapped I/O. The video RAM are memory mapped where as the Keyborad , Counter and Other devices are I/O Mapped.
- In I/O mapped I/O there are two set of instructions: IN and OUT to transfer data between I/O devices and accumulator (AX,AL)
- To distinguish between the memory read/write and I/O read or write M/IO signal is used. IF M/IO is high memory read and write are enabled else the I/O read and write
- Two ways to specify the I/O port address are:
    - An 8 bit immediate((Fixed or Direct) address(here address is specified as a part of the instruction)
    - 16 bit address located in register DX(Variable Address or Indirect)

I/O For Intel x86 Series

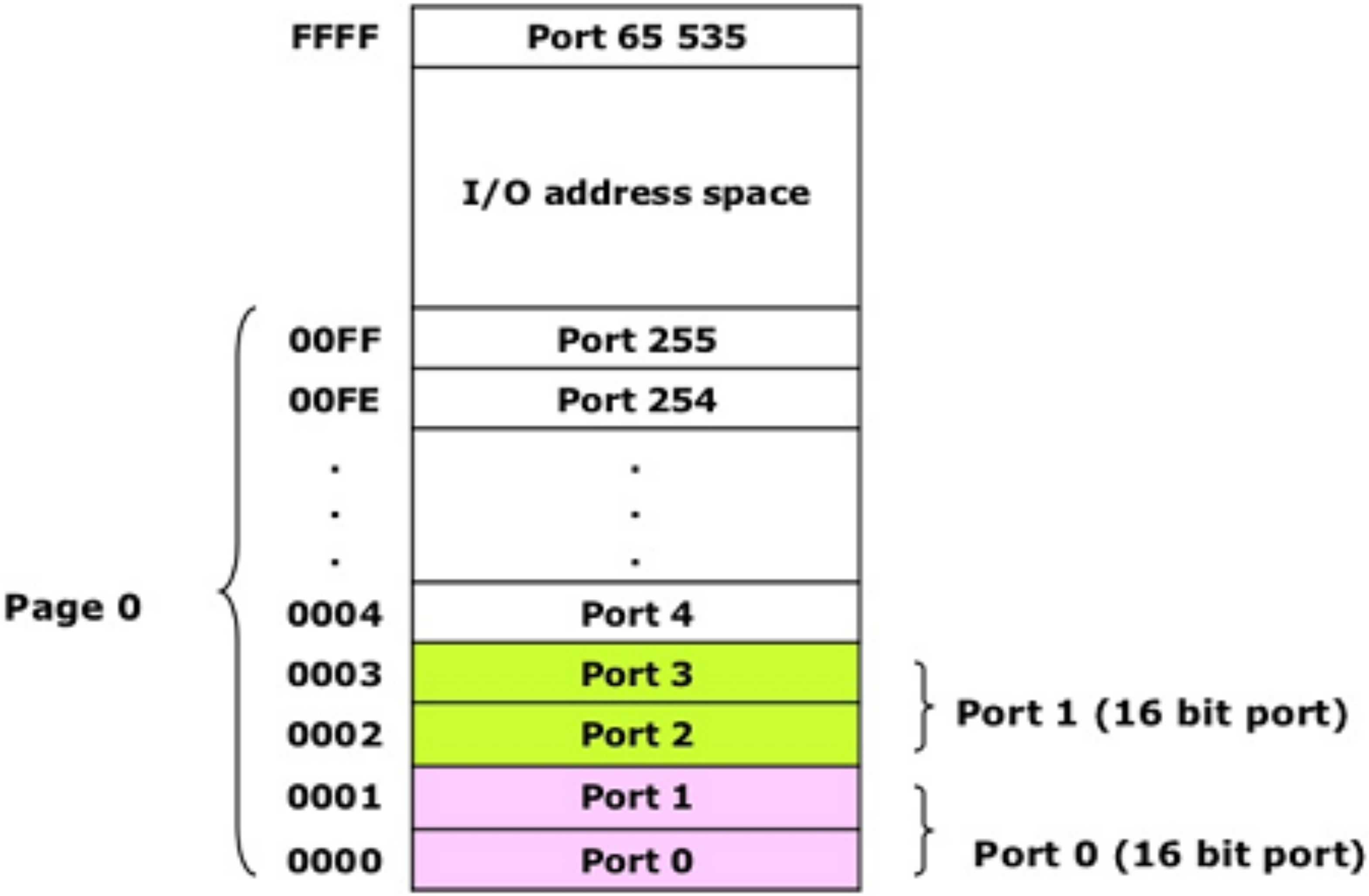


MNEMONIC	MEANING	FORMAT	OPERATION
IN	INPUT DIRECT	IN AL , ADDRESS 8 BIT	PORT->AL(BYTE)
		IN AX , ADDRESS 8 BIT	PORT->AX(WORD)
	INPUT INDIRECT	IN AL, DX	PORT->AL(BYTE)
		IN AX, DX	PORT->AX(WORD)
OUT	OUTPUT DIRECT	OUT ADDRESS 8 BIT,AL	AL->PORT(BYTE)
		OUT ADDRESS 8 BIT,AX	AX->PORT(WORD)
	OUTPUT INDIRECT	OUT DX , AL	AL->PORT(BYTE)
		OUT DX, AX	AX->PORT(WORD)

Memory Mapped I/O



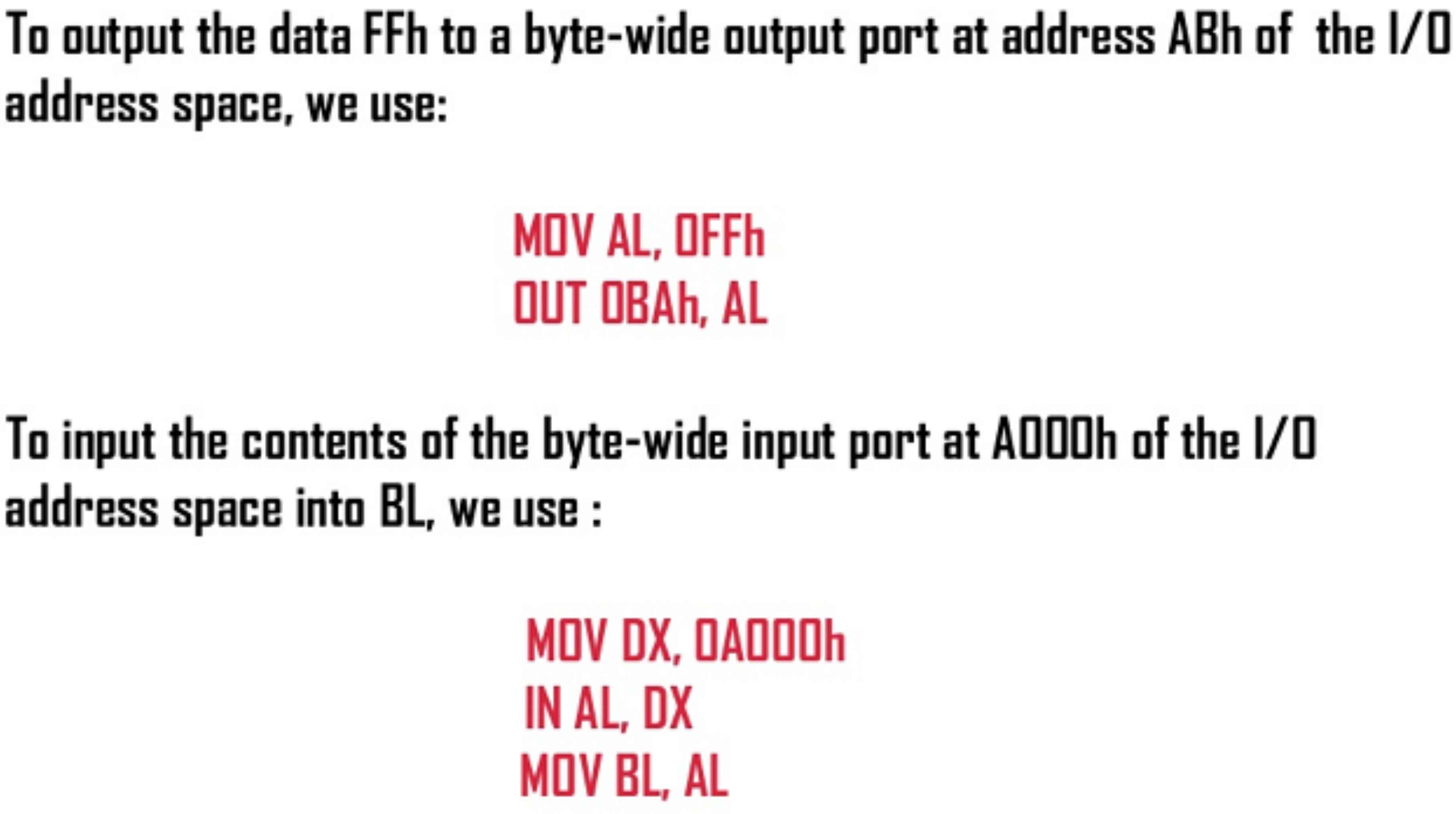
STANDARD I/O



FFFF	Port 65 535
	I/O address space
00FF	Port 255
00FE	Port 254
. . .	. . .
0004	Port 4
0003	Port 3
0002	Port 2
0001	Port 1
0000	Port 0

I/O SPACE

STANDARD I/O Example



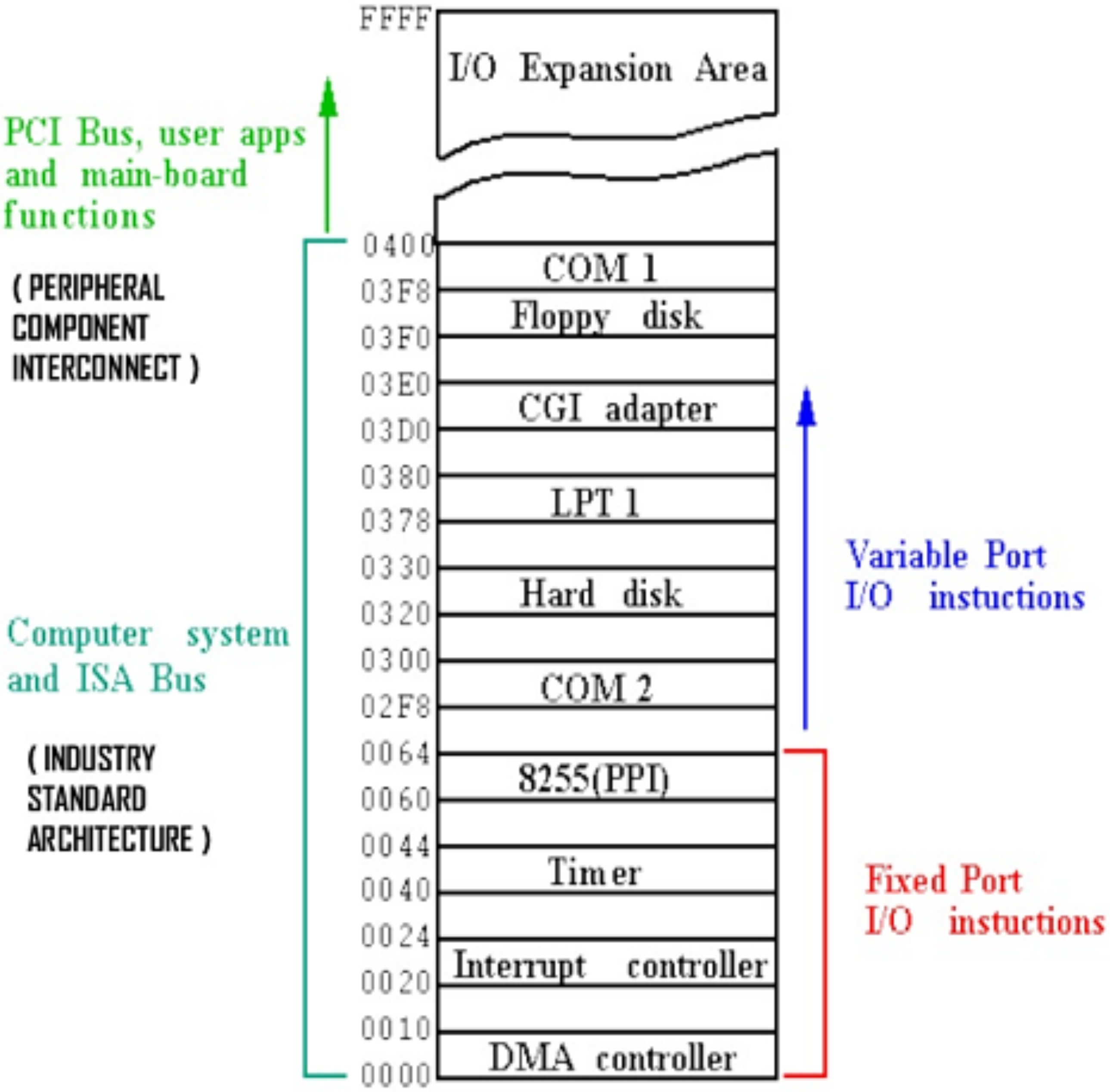
To output the data FFh to a byte-wide output port at address ABh of the I/O address space, we use:

```
MOV AL, 0FFh
OUT 0BAh, AL
```

To input the contents of the byte-wide input port at A000h of the I/O address space into BL, we use :

```
MOV DX, 0A000h
IN AL, DX
MOV BL, AL
```

PC I/O MAP



PCI Bus, user apps and main-board functions

(PERIPHERAL COMPONENT INTERCONNECT)

Computer system and ISA Bus

(INDUSTRY STANDARD ARCHITECTURE)

Variable Port I/O instuctions

Fixed Port I/O instuctions

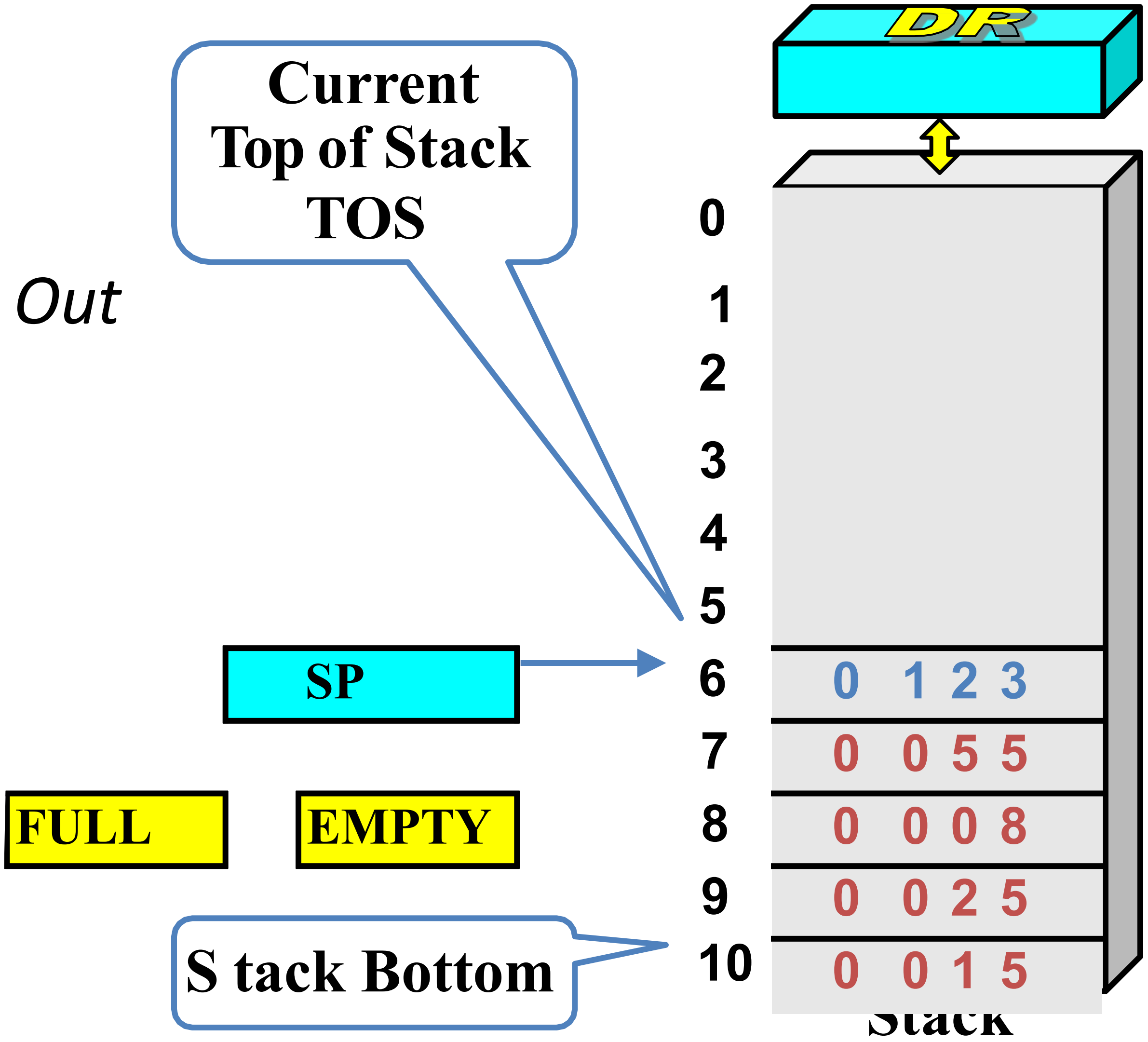
FFFF	
	I/O Expansion Area
0400	
	COM 1
03F8	
	Floppy disk
03F0	
03E0	
	CGI adapter
03D0	
0380	
	LPT 1
0378	
0330	
	Hard disk
0320	
0300	
	COM 2
02F8	
0064	
	8255(PPI)
0060	
0044	
	Timer
0040	
0024	
	Interrupt controller
0020	
0010	
	DMA controller
0000	

Stacks

Stack Organization

- LIFO

Last In First Out



DR

Current Top of Stack TOS

SP

FULL

EMPTY

S tack Bottom

0

1

2

3

4

5

6 0 1 2 3

7 0 0 5 5

8 0 0 0 8

9 0 0 2 5

10 0 0 1 5

Stack

Stack Organization

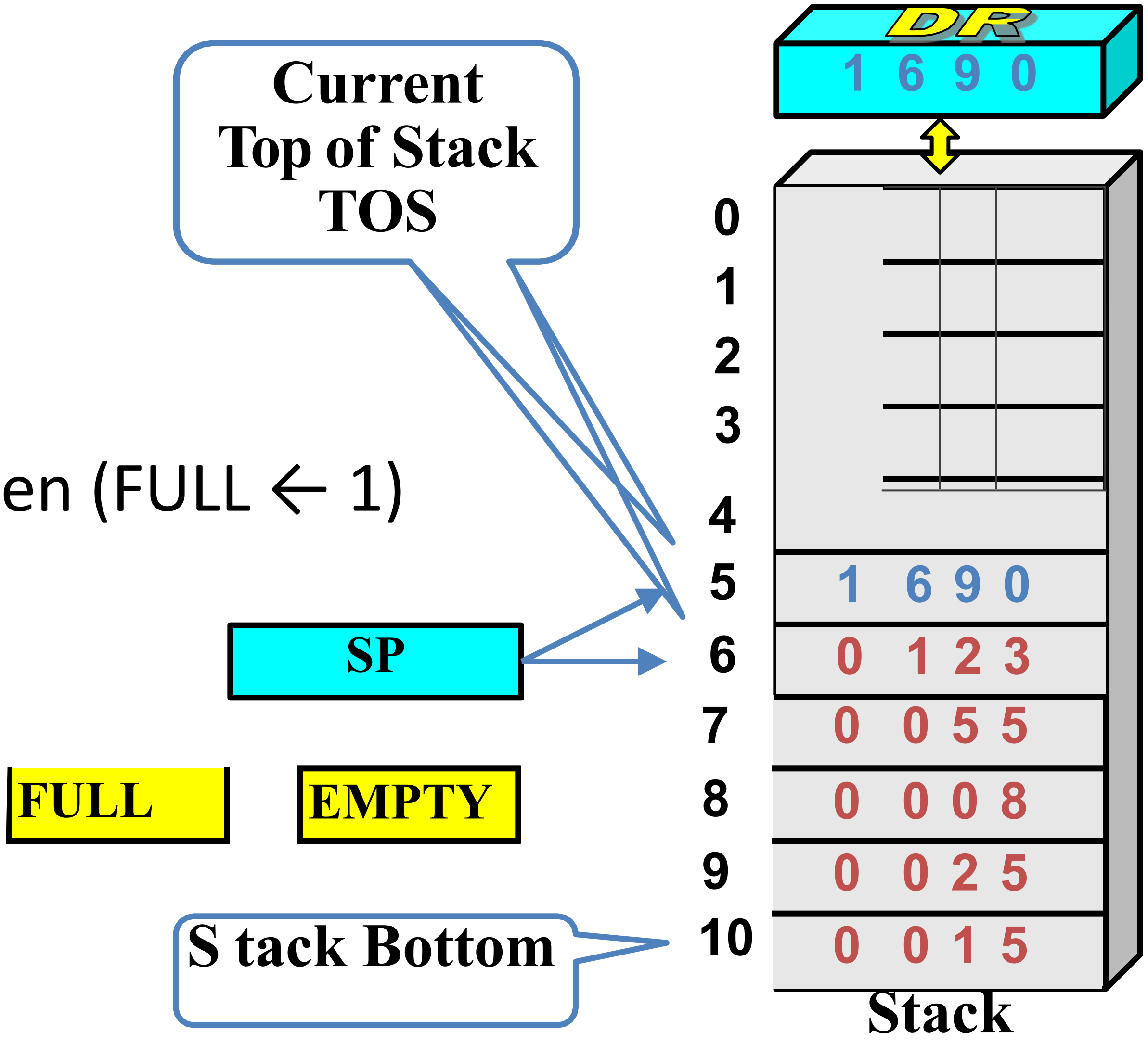
- PUSH

  SP ← SP – 1

  M[SP] ← DR

  If (SP = 0) then (FULL ← 1)

  EMPTY ← 0



Stack Organization

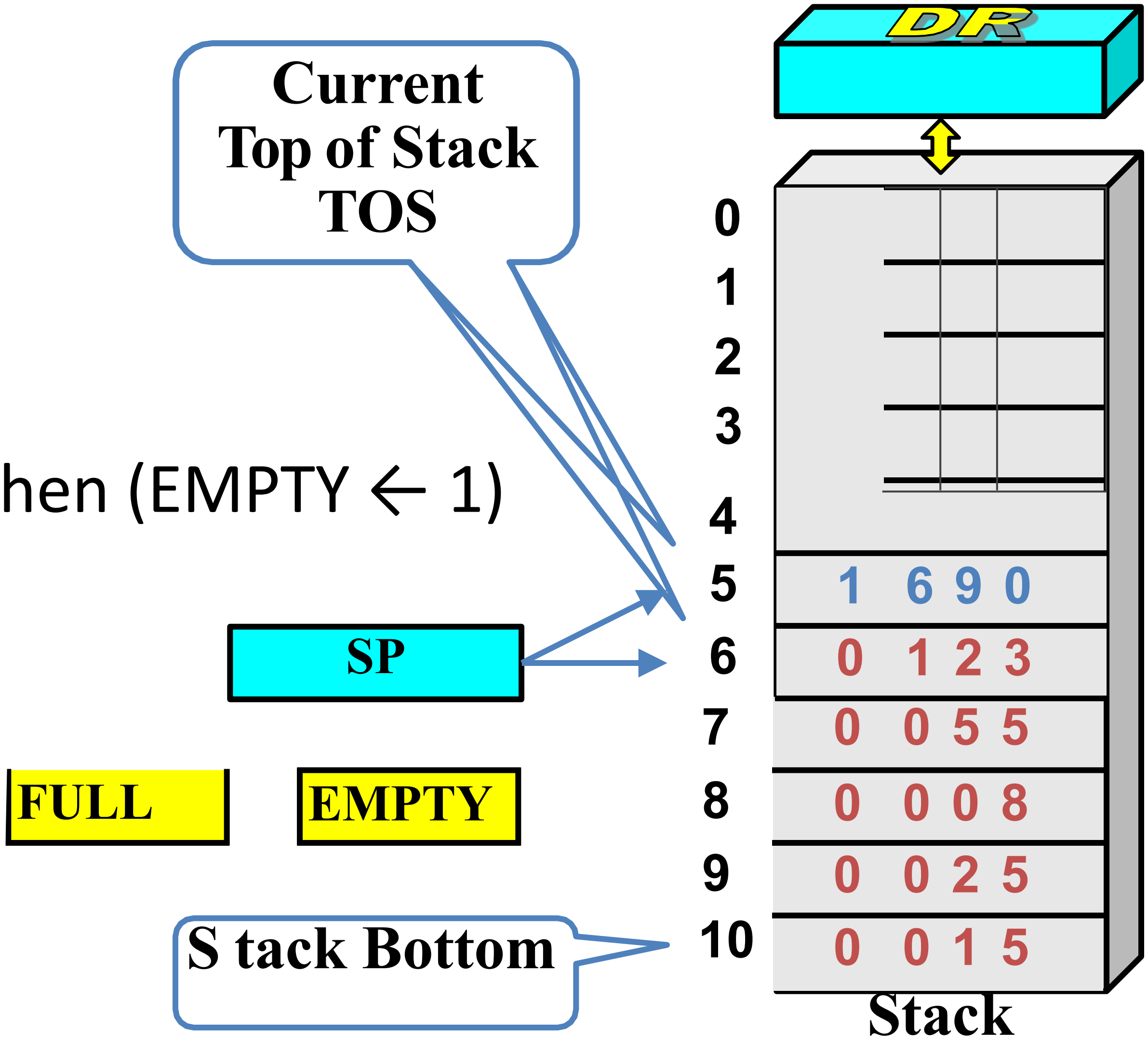
- POP

  DR ← M[SP]

  SP ← SP + 1

  If (SP = 11) then (EMPTY ← 1)

  FULL ← 0



Stack Organization

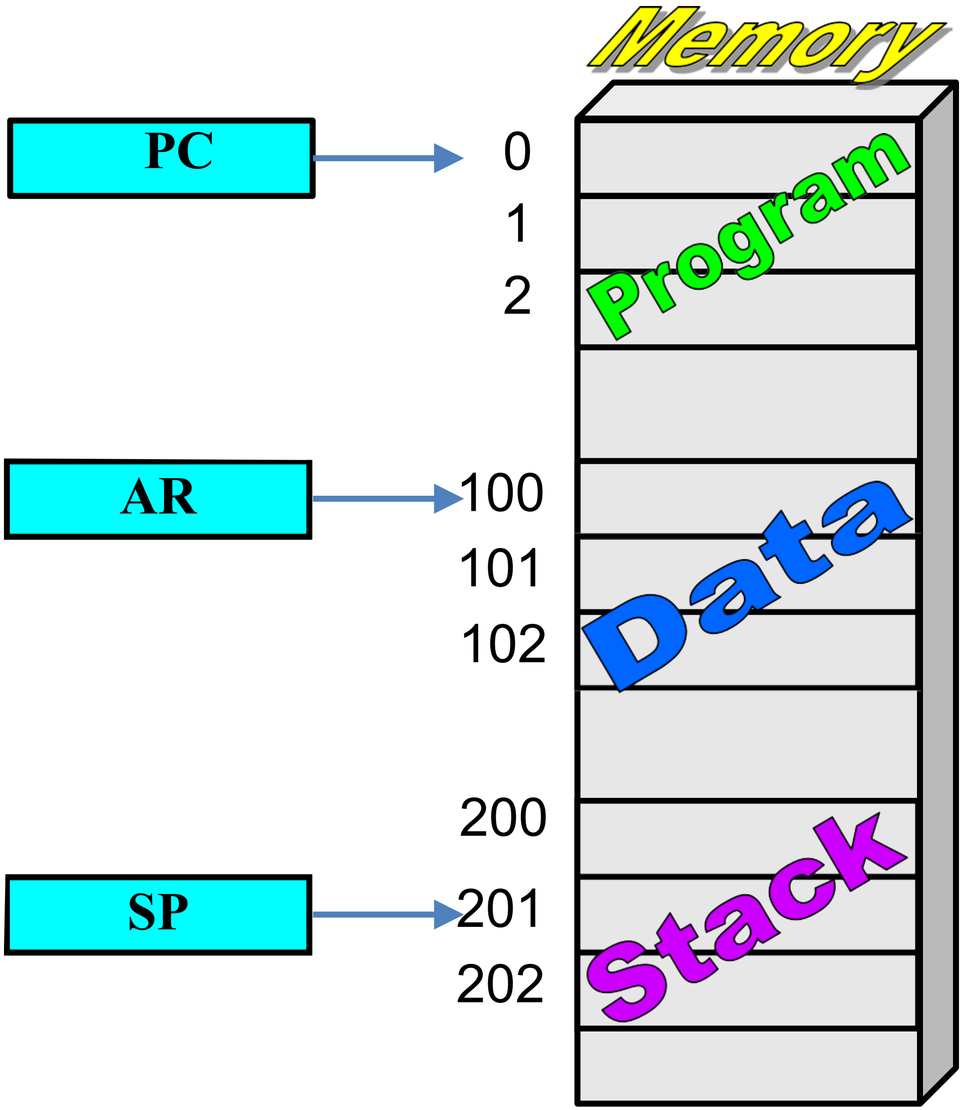
- Memory Stack
  - PUSH

    SP ← SP – 1

    M[SP] ← DR
  - POP

    DR ← M[SP]

    SP ← SP + 1



Reverse Polish Notation

- Infix Notation

  $A + B$

- Prefix or Polish Notation

  $+ A B$

- Postfix or Reverse Polish Notation (RPN)

  $A B +$

$$A * B + C * D \xrightarrow{\text{RPN}} A B * C D * +$$

(2) (4) * (3) (3) * +

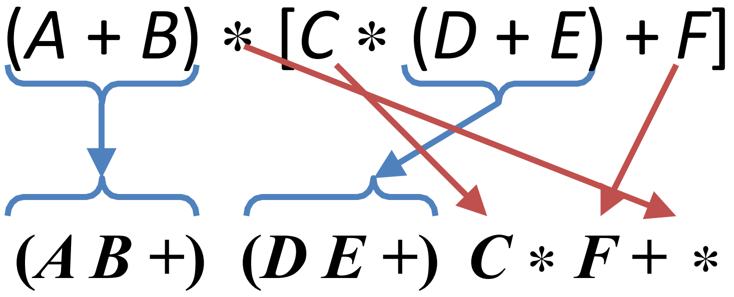
(8) (3) (3) * +

(8) (9) +

17

Reverse Polish Notation

- Example



$(A + B) * [C * (D + E) + F]$

$(A\;B\;+)\;\;(D\;E\;+)\;\;C * F + *$

Reverse Polish Notation

- Stack Operation

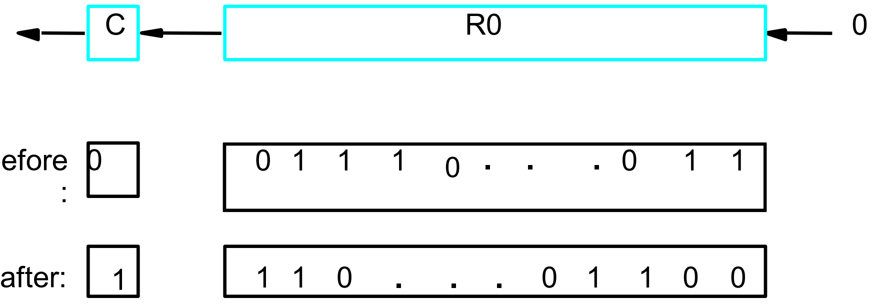
(3) (4) * (5) (6) * +

```
PUSH    3
PUSH    4
MULT
PUSH    5
PUSH    6
MULT
ADD
```

Additional Instructions

Logical Shifts

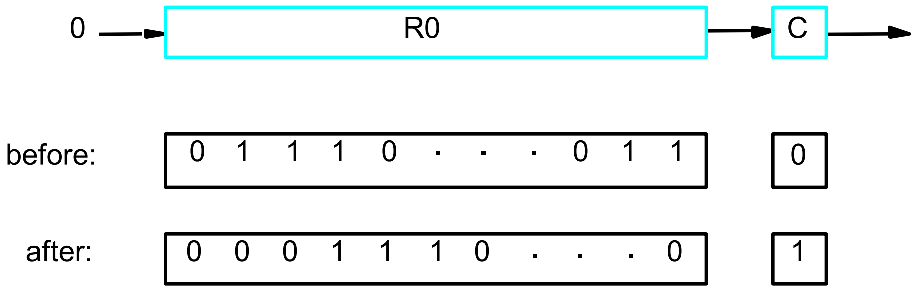
- Logical shift – shifting left (LShiftL) and shifting right (LShiftR)



before: 0 | 0 1 1 1 0 . . . 0 1 1

after: 1 | 1 1 0 . . . 0 1 1 0 0

(a) Logical shift left LShiftL #2,R0

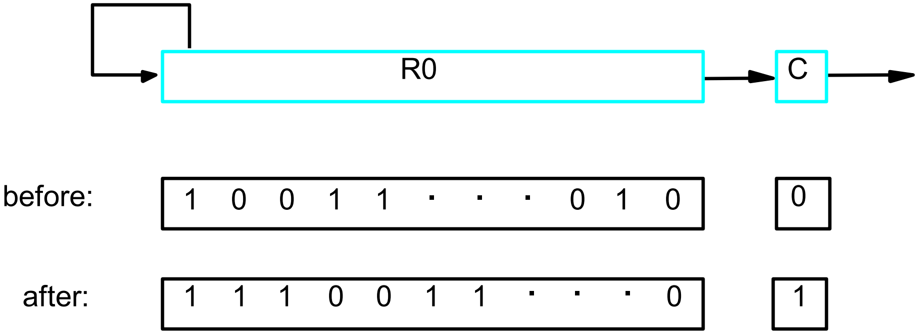


before: 0 1 1 1 0 . . . 0 1 1 | 0

after: 0 0 0 1 1 1 0 . . . 0 | 1

(b) Logical shift right LShiftR #2,R0

Arithmetic Shifts

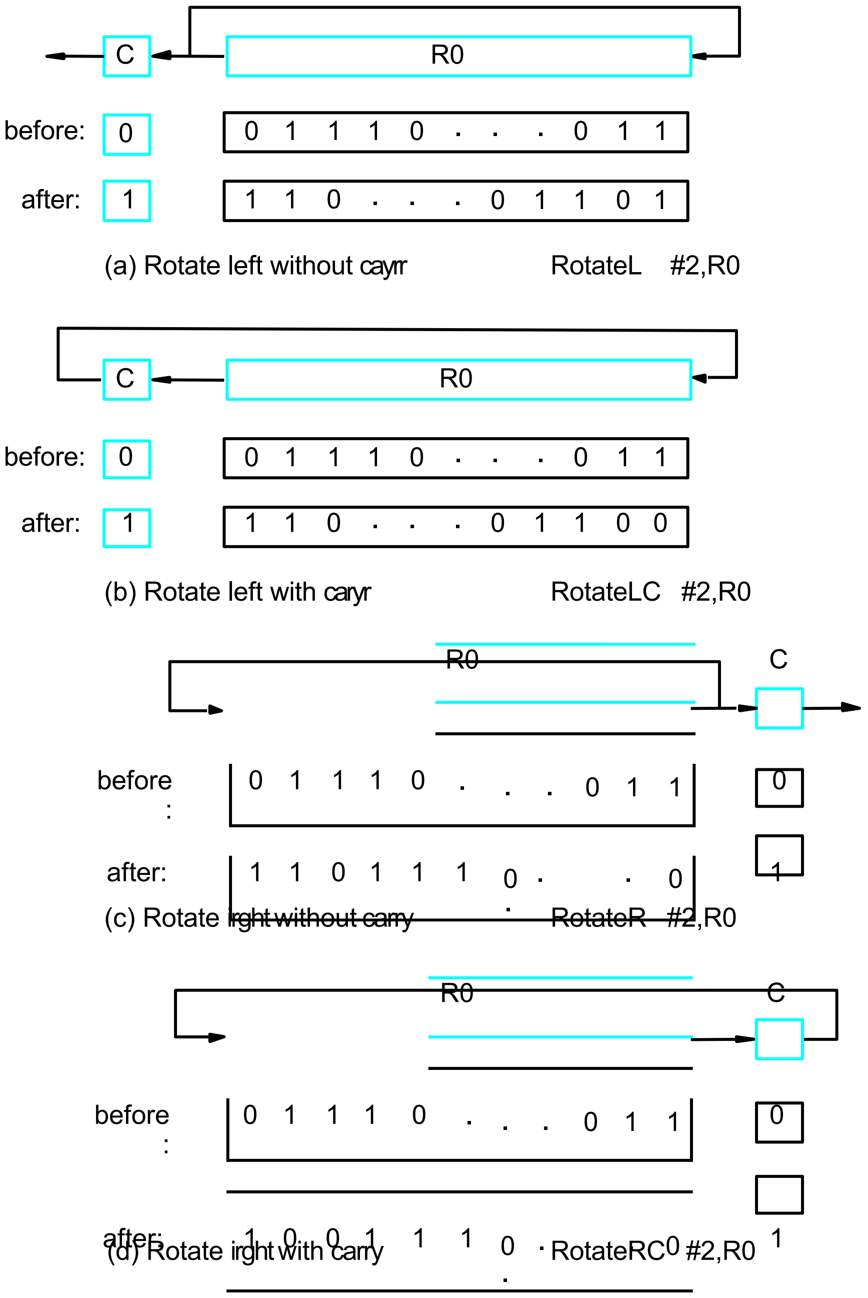


(c) Arithmetic shift right

AShiftR #2,R0

Arithmetic Left Shift?

Rotate



C R0

before: 0 | 0 1 1 1 0 . . . 0 1 1

after: 1 | 1 1 0 . . . 0 1 1 0 1

(a) Rotate left without cayrr RotateL #2,R0

C R0

before: 0 | 0 1 1 1 0 . . . 0 1 1

after: 1 | 1 1 0 . . . 0 1 1 0 0

(b) Rotate left with caryr RotateLC #2,R0

R0 C

before: 0 1 1 1 0 . . . 0 1 1 | 0

after: 1 1 0 1 1 1 0 . . . 0 | 1

(c) Rotate irght without carry RotateR #2,R0

R0 C

before: 0 1 1 1 0 . . . 0 1 1 | 0

after: 1 0 0 1 1 1 0 . . . 0 | 1

(d) Rotate irght with carry RotateRC #2,R0

Figure 2.32. Rotate instructions.

Multiplication and Division

- Not very popular (especially division)
- Multiply R_i, R_j

  $R_j \leftarrow [R_i] \times [R_j]$
- 2n-bit product case: high-order half in R(j+1)
- Divide R_i, R_j

  $R_j \leftarrow [R_i] / [R_j]$

  Quotient is in Rj, remainder may be placed in R(j+1)

Encoding of Machine Instructions

Encoding of Machine Instructions

- Assembly language program needs to be converted into machine instructions. (ADD = 0100 in ARM instruction set)
- In the previous section, an assumption was made that all instructions are one word in length.
- OP code: the type of operation to be performed and the type of operands used may be specified using an encoded binary pattern
- Suppose 32-bit word length, 8-bit OP code (how many instructions can we have?), 16 registers in total (how many bits?), 3-bit addressing mode indicator.

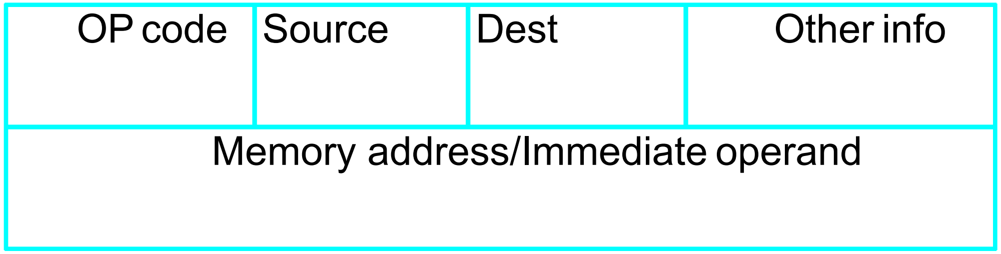
- Add R1, R2
- Move 24(R0), R5
- LshiftR #2, R0
- Move #$3A, R1
- Branch>0 LOOP

8	7	7	10
OP code	Source	Dest	Other info

(a) One-word instruction

Encoding of Machine Instructions

- What happens if we want to specify a memory operand using the Absolute addressing mode?
- Move R2, LOC
- 14-bit for LOC – insufficient
- Solution – use two words



OP code | Source | Dest | Other info

Memory address/Immediate operand

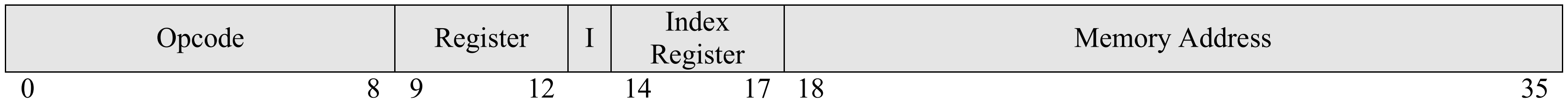
(b) Two-word instruction

Encoding of Machine Instructions

- Then what if an instruction in which two operands can be specified using the Absolute addressing mode?
- Move LOC1, LOC2
- Solution – use two additional words
- This approach results in instructions of variable length. Complex instructions can be implemented, closely resembling operations in high-level programming languages – Complex Instruction Set Computer (CISC)

Encoding of Machine Instructions

- If we insist that all instructions must fit into a single 32-bit word, it is not possible to provide a 32-bit address or a 32-bit immediate operand within the instruction.
- It is still possible to define a highly functional instruction set, which makes extensive use of the processor registers.
- Add R1, R2 ----- yes
- Add LOC, R2 ----- no
- Add (R3), R2 ----- yes



I = indirect bit

Figure 13.6 PDP-10 Instruction Format

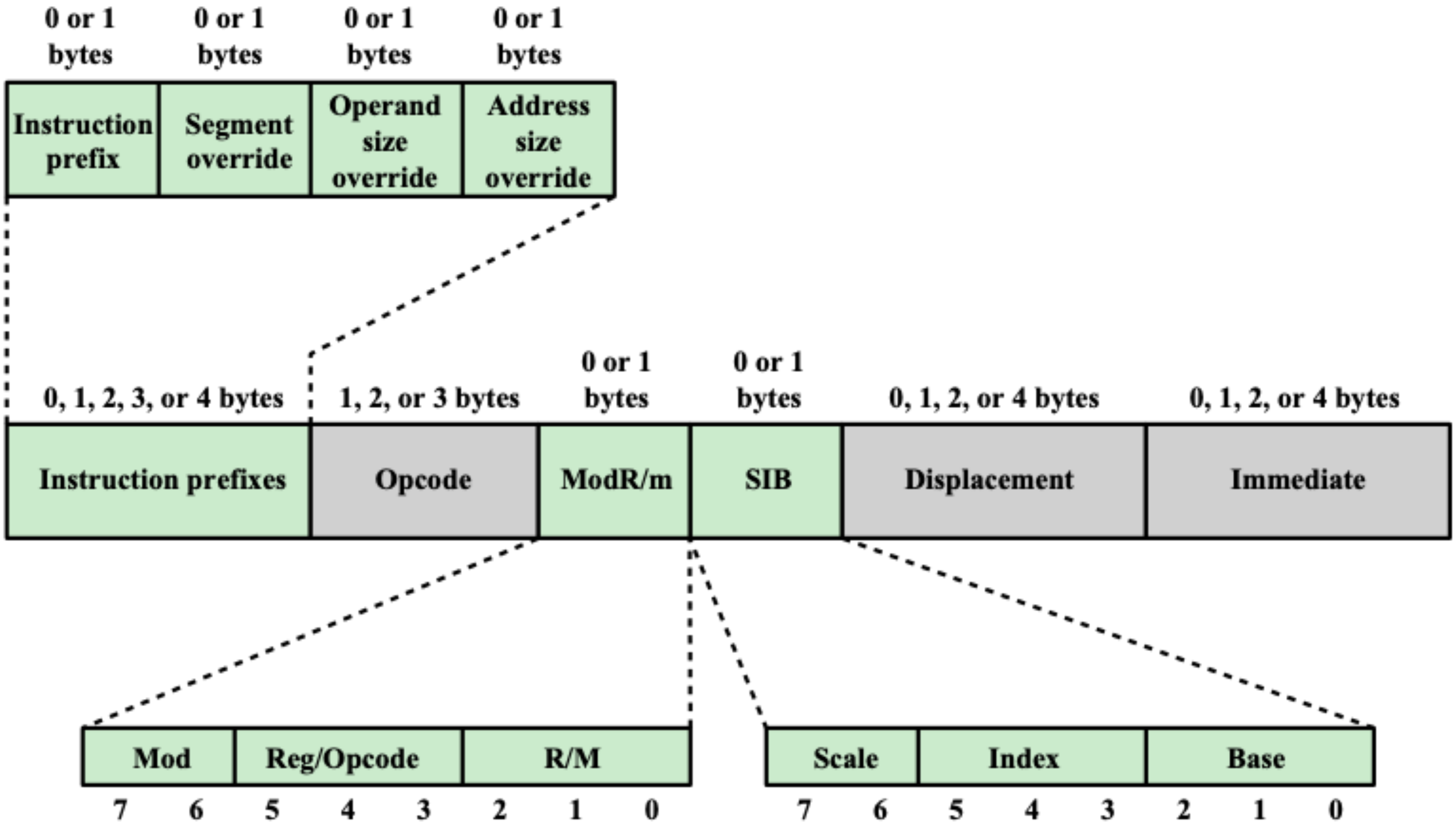
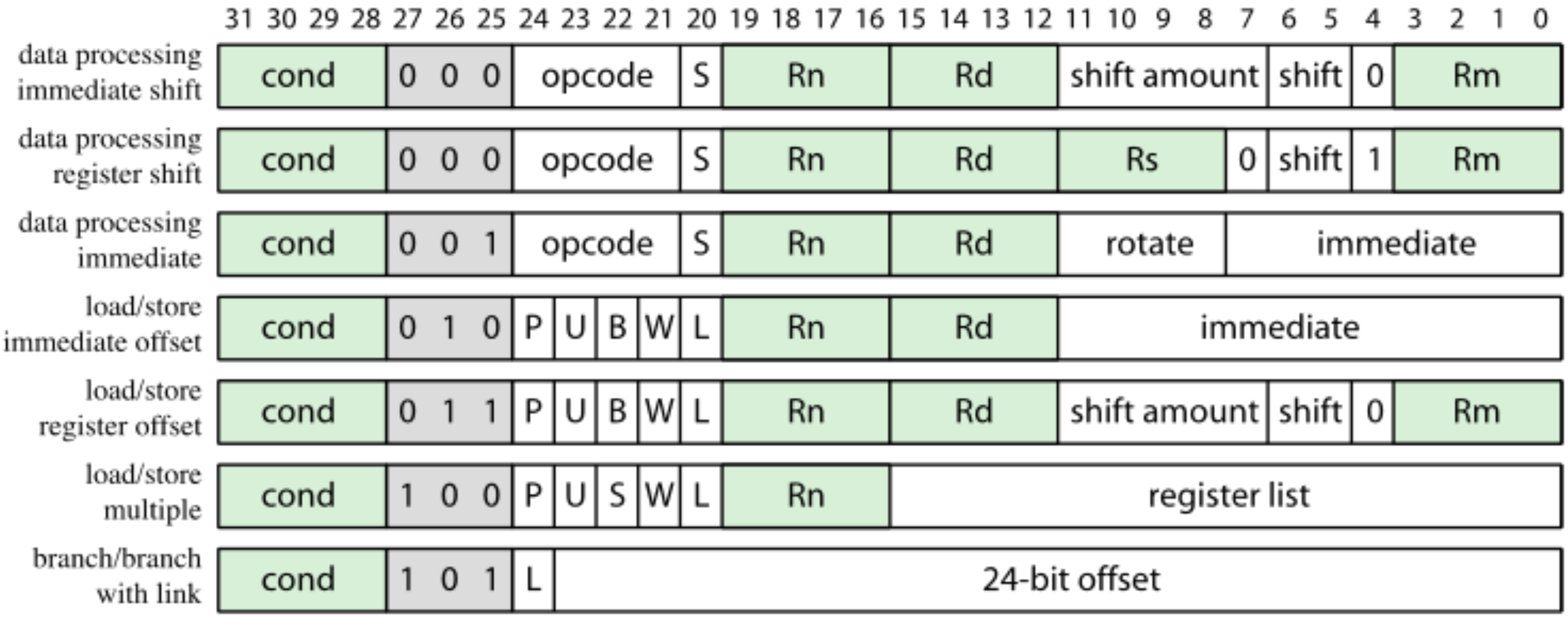


Figure 13.9 x86 Instruction Format



	31 30 29 28	27 26 25	24	23	22	21	20	19 18 17 16	15 14 13 12	11 10 9 8	7	6 5	4	3 2 1 0
data processing immediate shift	cond	0 0 0	opcode				S	Rn	Rd	shift amount		shift	0	Rm
data processing register shift	cond	0 0 0	opcode				S	Rn	Rd	Rs	0	shift	1	Rm
data processing immediate	cond	0 0 1	opcode				S	Rn	Rd	rotate	immediate			
load/store immediate offset	cond	0 1 0	P	U	B	W	L	Rn	Rd	immediate				
load/store register offset	cond	0 1 1	P	U	B	W	L	Rn	Rd	shift amount		shift	0	Rm
load/store multiple	cond	1 0 0	P	U	S	W	L	Rn	register list					
branch/branch with link	cond	1 0 1	L	24-bit offset										

S = For data processing instructions, signifies that the instruction updates the condition codes

S = For load/store multiple instructions, signifies whether instruction execution is restricted to supervisor mode

P, U, W = bits that distinguish among different types of addressing_mode

B = Distinguishes between an unsigned byte (B==1) and a word (B==0) access

L = For load/store instructions, distinguishes between a Load (L==1) and a Store (L==0)

L = For branch instructions, determines whether a return address is stored in the link register

Figure 13.10 ARM Instruction Formats