8086 Instruction Set

Data Movement — Abbreviations

src = source

dest = destination

acc = accumulator

PTR = pointer

DWORD = double word (32 bits)

Data Movement — Conventions

AX $\leftarrow$ **BX** $\Rightarrow$ Copy **BX** to **AX**

AL $\leftarrow$ **[1000]** $\Rightarrow$ Copy memory byte at address **1000** to **AL**

AL $\leftarrow$ **[BX]** $\Rightarrow$ Copy memory byte at address stored in **BX** to **AL**

Data Movement — Scope

BYTE PTR[1000] = memory byte at address **1000**

WORD PTR[1000] = memory bytes at addresses **1000** and **1001**

DWORD PTR[1000] = memory bytes at addresses **1000** to **1003**

AX $\leftarrow$ **[BX]** $\Rightarrow$ **AL** $\leftarrow$ **[BX]** and
AH $\leftarrow$ **[BX+1]**

AX $\leftarrow$ **[BX]** $\Rightarrow$ **AX** $\leftarrow$ **WORD PTR[BX]**

Data Movement Instructions — 1

MOV dest ,src	**MOV AX, BX**	**AX ← BX**

PUSH src	**PUSH CX**	**SP ← SP-2;**
		[SP+1] ← CH;
		[SP] ← CL

POP dest	**POP CX**	**CL ← [SP];**
		CH ← [SP+1];
		SP ← SP+2

XCHG dest, src	**XCHG AX, BX** **XCHG AL, BH** **XCHG [SI], DX**	**AX ↔ BX** **AL ↔ BH** **[SI] ↔ DL** **[SI+1] ↔ DH**

Data Movement Instructions — 2

LAHF	**LAHF**	**AH ← FLAGS 1**
SAHF	**SAHF**	**FLAGS 1 ← AH**
LEA dest, src	**LEA BX, [BP+SI+4]**	**BX ← BP+SI+4**

LDS dest, src	**LDS BX, [SI]**	**BL ← [SI];**
		BH ← [SI+1];
		DS ← [SI+3:SI+2]

LES dest, src	**LES BX, [SI]**	**BL ← [SI];**
		BH ← [SI+1];
		ES ← [SI+3:SI+2]

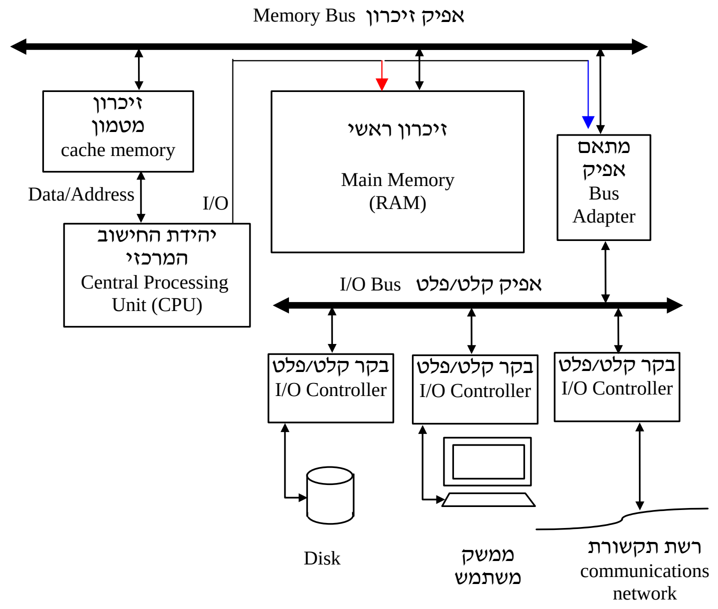
Data Movement — I/O Operations — 1

80x86 processors control an I/O signal on the memory bus
- I/O signal is **off** to select processor access to **RAM**
- I/O signal is **on** to select processor access to **I/O** bus
- **MOV** selects **RAM** access
- **IN** and **OUT** select **I/O** access

AL or AX are always src/dest for **I/O** instructions

I/O address is called a **port**
- can range from **0000 H** to **FFFF H**
- direct mode — 1 immediate byte address
- indirect mode — 2 address bytes in **DX**

Data Movement — I/O Operations — 2


Memory Bus אפיק זיכרון
זיכרון מטמון cache memory
זיכרון ראשי Main Memory (RAM)
מתאם אפיק Bus Adapter
Data/Address
I/O
יחידת החישוב המרכזי Central Processing Unit (CPU)
I/O Bus אפיק קלט/פלט
בקר קלט/פלט I/O Controller
בקר קלט/פלט I/O Controller
בקר קלט/פלט I/O Controller
Disk
ממשק משתמש
רשת תקשורת communications network

Data Movement — I/O Instructions — 3

IN acc, port	**IN AL,26H**	**AL ← port 26H**	**input byte from port 0 —— 255**
	IN AX,26H	**AL ← port 26H; AH ← port 27H**	
	IN AL,DX	**AL ← port DX**	**input byte from port 0 —— 65,535 (address in DX)**
	IN AX,DX	**AL ← port DX AH ← port DX+1**	
OUT port, acc	**OUT DX,AX**	**port DX ← AL port DX+1 ← AH**	**output byte to port 0 —— 65,535 (address in DX)**

XLAT

Replaces **AL** with **(AL+1)**st value in a table

Uses **AL** as index into 256 byte table beginning at **[BX]**

XLAT	**AL ← [BX+AL]**	**Replace byte in AL with byte from 256 byte table beginning at [BX], using AL as table offset**

Segment Override

CS:	**MOV CS:[BP],CX**	**CS:[BP] ← CX**
ES:	**MOV ES:[BP],CX**	**ES:[BP] ← CX**
DS:	**MOV DS:[BP],CX**	**DS:[BP] ← CX**
SS:	**MOV SS:[BP],CX**	**SS:[BP] ← CX**

Logical Operations

NOT dest	**NOT BX**	**BX ← not BX**
AND dest, src	**AND CX,DX**	**CX ← CX · DX**
OR dest, src	**OR CX,DX**	**CX ← CX + DX**
XOR dest, src	**XOR CX,DX**	**CX ← CX ⊕ DX**
TEST dest, src	**TEST CX,DX**	**CX · DX; update flags**

Unsigned Integers — 1

n-bit number

Represents value from **0** to $\mathbf{2^n-1}$

Integers determined modulo $\mathbf{2^n}$

Overflow

- $\mathbf{a+b>2^n-1}$
- Carry Flag is set

n=3	
7	111
6	110
5	101
4	100
3	011
2	010
1	001
0	000

Unsigned Integers — 2

CF	3-Bit Integer
0	111 + 001
1	000

CF	3-Bit Integer
0	000 - 001
1	111

Signed Numbers — 1

n-bit number (2’s complement)

Represents value from $\mathbf{-2^{n-1}}$ **to** $\mathbf{+2^{n-1}-1}$

Integers determined modulo $\mathbf{2^n}$

Overflow

- Carry-in not equal to carry-out at highest order
- Overflow Flag is set

Signed Numbers — 2

Upper bit = 0 for positive numbers

Upper bit = 1 for negative numbers

n=3	
+3	011
+2	010
+1	001
0	000
-1	111
-2	110
-3	101
-4	100

n=4	
+7	0111
…	…
+3	0011
+2	0010
+1	0001
0	0000
-1	1111
-2	1110
-3	1101
-4	1100
…	…
-8	1000

Signed Numbers — 3

OF	CO	CI	3-Bit Integer	Decimal
			111 + 001	-1 + 1
0	1	1	000	0

OF	CO	CI	3-Bit Integer	Decimal
			111 - 001	-1 - 1
0	0	0	110	-2

OF	CO	CI	3-Bit Integer	Decimal
			011 + 001	3 + 1
1	0	1	100	4

Data Conversion

CBW — convert byte to word with sign extension

CWD — convert word to double with sign extension

CBW	If AL < 80H, then AH ← 0 00000000 0xxxxxxx ← xxxxxxxx 0xxxxxxx
	If AL > 7F, then AH ← FFH 11111111 1xxxxxxx ← xxxxxxxx 1xxxxxxx
CWD	If AX < 8000H, then DX ← 0
	If AX > 7FFFH. then DX ← FFFFH

Add/Subtract

ADD dest, src	ADD BYTE PTR[BX],CH	[BX] ← [BX] + CH
ADC dest, src	ADC SI,DX	SI ← SI+DX+CF
SUB dest ,src	SUB SI,DX	SI ← SI-DX
SBB dest ,src	SBB SI,DX	SI ← SI-DX-CF
INC dest	INC BL	BL ← BL+1
DEC dest	DEC BL	BL ← BL-1
NEG dest	NEG BL	BL ← 0- BL
CMP dest ,src	CMP AL,AH	AL - AH; update flags

Multiplication / Division

MUL source	MUL BL	AX ← AL*BL
	MUL CX	DX:AX ← AX*CX
IMUL source	IMUL BL	AX ← AL*BL
	IMUL CX	DX:AX ← AX*CX

DIV source	DIV BL	AL ← BL / AX
		AH ← BL % AX

	DIV CX	AX ← CX / DX:AX
		AX ← CX % DX:AX

IDIV source	IDIV BL	AL ← BL / AX
		AH ← BL % AX

	IDIV CX	AX ← CX / DX:AX
		AX ← CX % DX:AX

Jump Instructions

Near — target within same code segment

Short — like near, but offset is one byte long

Far

Target is outside segment

Pointer is double word

JMP near target	JMP 1024	IP ← 1024
JMP short target	JMP 1024	IP ← 1024 (-128 < change in IP < 127)

JMP far target	JMP FAR PTR [1024]	IP ← [1025:1024];
		CS ← [1027:1026]

Conditional Jumps

ALU operations set flags in the status word

Conditional jumps test the flags and jump if flag is set

Mnemonic	Condition	Test
	Signed or Unsigned	
JC	Carry	CF = 1
JE/JZ	Equal/zero	ZF = 1
JP/JPE	Parity / parity even	PF = 1
JNC	Not carry	CF = 0
JNE/JNZ	Not equal/not zero	ZF = 0
JNP/JPO	Not Parity / parity odd	PF = 0

Unsigned Compare — 1

COMP	ZF	CF
A < B	0	1
A = B	1	0
A > B	0	0

$A < B \Rightarrow CF = 1$

$A \leq B \Rightarrow (ZF \oplus CF) = 1$

$A = B \Rightarrow ZF = 1$

$A \geq B \Rightarrow CF = 0$

$A > B \Rightarrow (ZF \oplus CF) = 0$

(ZF = CF = 1 is impossible)

Unsigned Compare — 2

Mnemonic	Condition	Test
	Unsigned Operations	
JA/JNBE	Above/not below nor equal	$(CF \oplus ZF) = 0$
JAE/JNB	Above or equal/not below	CF = 0
JB/JNAE	Below/not above nor equal	CF = 1
JBE/JNA	Below or equal/not above	$CF \oplus ZF = 1$

Signed Compare

Mnemonic	Condition	Test
	Signed Operations	
JG/JNLE	Greater/not less nor equal	(SF ⊕ OF) + ZF) = 0
JGE/JNL	Greater or equal/not less	(SF ⊕ OF) = 0
JLIJNGE	Less/not greater nor equal	(SF ⊕ OF) = 1
JLE/JNG	Less or equal/not greater	((SF ⊕ OF) + ZF) = 1
JO	Overflow	OF = 1
JS	Sign	SF = 1
JNO	Not overflow	OF = 0
JNS	Not sign	SF = 0

Loop Instructions

LOOP short target	LOOP 1024	CX ← CX - 1 If CX ≠ 0, then IP ← 1024 -128 < 1024 - IP < 127
LOOPZ short target	LOOPZ 1024	CX ← CX - 1 If (CX ≠ 0) AND (ZF = 1), then IP ← 1024 -128 < 1024 - IP < 127
LOOPNZ short target	LOOPNZ 1024	CX ← CX - 1 If (CX ≠ 0) AND (ZF = 0), then IP ← 1024 -128 < 1024 - IP < 127

Call and Return

CALL near target	CALL 1024	SP ← SP-2;
		[SP+1:SP] ← IP;
		IP ← 1024

CALL far target	CALL FAR PTR [1024]	SP ← SP-2;
		[SP+1:SP] ← CS;
		CS ← [1025:1024];
		SP ← SP-2;
		[SP+1:SP] ← IP;
		IP ← [1027:1026]

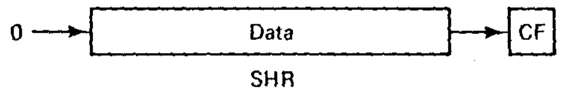
RET n (near)	RET
	
	RET 8
		IP ← [SP+1:SP];
		SP ← SP+2
		IP ← [SP+1:SP] ;
		SP ← SP+2+8

RET n (far)	RET
	
	RET 8
		IP ← [SP+1:SP];
		SP ← SP+2;
		CS ← [SP+1:SP];
		SP ← SP+2
		IP ← [SP+1:SP];
		SP ← SP+2;
		CS ← [SP+1:SP];
		SP ← SP+2+8

Shift And Rotate Instructions — 1

SHR

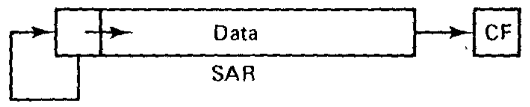
Shift Right



SAR

Shift Arithmetic Right

Shift bits right with sign preservation

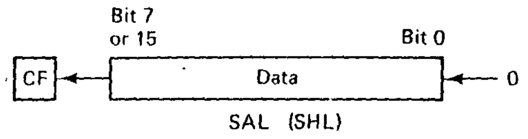


Shift And Rotate Instructions — 2

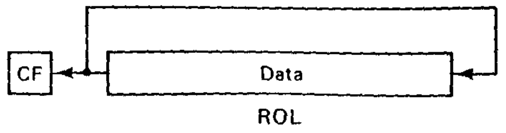
SAL

Shift Arithmetic Left

Shift left, sign bit to CF (OF = 1 if sign bit changes)

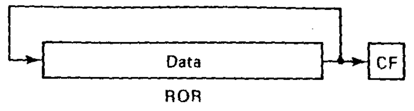


ROL — Rotate Left

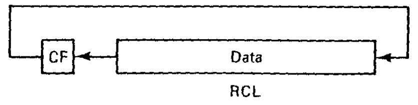


Shift And Rotate Instructions — 3

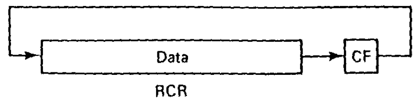
ROR — rotate right



RCL — rotate carry left



RCR — rotate carry right



Software Interrupts

Interrupts Program Flow

Transfers control to Interrupt Service Routine (ISR)

ISR can be stored anywhere in memory

Pointer to ISR stored in Interrupt Vector Table

Table starts at **00000**

Each Vector is 4 bytes long (2×**CS**+2×**IP**)

Vector 0 is at physical address 00000

Vector 1 is at physical address 00004

Vector 2 is at physical address 00008

ISR vector address is: **INT_type**×**4**

Vector is: **IP(L), IP(H), CS(L), CS(H)**

Software Interrupt Instructions

`INT type`	`INT 20H`	`SP ← SP-2; [SP+1:SP] ← flags`
		`IF ← 0; TF ← 0;`
		`SP ← SP-2; [SP+l:SP] ← CS;`
		`CS ← [00083H:00082H];`
		`SP ← SP-2; [SP+1:SP] ← IP;`
		`IP ← [00081H:00080H]`

`IRET none`	`IRET`	`IP ← [SP+1:SP];`
		`SP ← SP+2;`
		`CS ← [SP+1:SP];`
		`SP ← SP+2;`
		`flags ← [SP+1:SP];`
		`SP ← SP+2`

Hardware Interrupts

INTR line on chip causes interrupt

- INT_type is on address lines **`A0 to A7`**

NMI line on chip causes non-maskable interrupt

- Cannot be masked (turned off)
- Always points to **`INT type 4`**

Processor Control Instructions

STC	F9	STC	Within CPU	CF ← I	Set carry flag
CLC	F8	CLC	Within CPU	CF ← 0	Clear carry flag
CMC	FS	CMC	Within CPU	CF ← not(CF)	Complement carry flag
STD	FD	STD	Within CPU	DF ← 1	Set direction flag
CLD	FC	CLD	Within CPU	DF ← 0	Clear direction flag
STI	FB	STI	Within CPU	IF ← I	Set interrupt flag
CLI	FA	CLI	Within CPU	IF ← 0	Clear interrupt flag
HLT	F4	HLT	Within CPU	None	Halt
WAIT	9B	WAIT	Within CPU	None	Enter wait state
NOP	90	NOP	Within CPU	None	No operation

String Instructions — 1

STOSB	**ES:[DI] ← AL** **If DF = 0, DI ← DI+1** **If DF = 1, DI ← DI-1**
STOSW	**ES:[DI] ← AL; ES:[DI+1] ← AH** **If DF = 0, DI ← DI+2** **If DF = 1, DI ← DI-2**
LODSB	**AL ← DS:[SI]** **If DF = 0, SI ← SI+1** **If DF = 1, SI ← SI-1.**
LODSW	**AL ← DS:[SI]: AH ← DS:[SI+1]** **If DF = 0, SI ← SI+2;** **If DF = 1, SI ← SI-2**

String Instructions — 2

MOVSB	**ES:[DI] ← DS:[SI]** **If DF = 0, DI ← DI+1** **SI ← SI+1** **If DF = 1, DI ← DI-1** **SI ← SI-1**
MOVSW	**ES:[DI] ← DS:[SI]** **ES:[DI+1] ← DS:[SI+1]** **If DF = 0, DI ← DI+2** **SI ← Sl+2** **If DF = 1, DI ← DI-2** **SI ← SI-2**
SCASB	**AL-ES:[DI]; update flags** **If DF = 0, DI ← DI+1** **If DF = 1, DI ← DI-1**

String Instructions — 3

SCASW	AX-ES:[DI+l:DI]; update flags If DF = 0, DI ← DI+2 If DF = I, DI ← DI-2
CMPSB	DS:[SI]-ES:[DI]; update flags If DF = 0, DI ← DI+I SI ← SI+1 If DF = 1 , DI ← DI-1 SI ← SI-1
CMPSW	DS:[SI+I:SI]-ES:[DI+1:DI]; update flags If DF = 0, DI ← DI+2 SI ← SI+2 If DF = 1, DI ← DI-2 SI ← SI-2

String Instructions — 4

REP STOSB	STOSB; CX ← CX-1 Repeat until CX = 0
REP STOSW	STOSW; CX ←CX-1 Repeat until CX = 0
REP MOVSB	MOVSB; CX ← CX-1 Repeat until CX = 0
REP MOVSW	MOVSW; CX ← CX-1 Repeat until CX = 0