

Programmazione Assembly 8086

Istituto Tecnico Superiore Antonio Scarpa
Anno Scolastico 2025-2026

15 novembre 2025

Indice

I	Fondamenti dell'Architettura 8086	11
	Prefazione	13
1	Architettura del Microprocessore 8086	17
1.1	Introduzione	17
1.2	Caratteristiche principali	17
1.2.1	Specifiche tecniche	17
1.2.2	Organizzazione interna	18
1.3	Modalità di funzionamento	18
1.3.1	Modalità minima	18
1.3.2	Modalità massima	18
1.4	Memoria segmentata	18
1.4.1	Calcolo indirizzo fisico	19
1.4.2	Rappresentazione grafica	19
1.5	Flag register	19
1.6	Pinout e segnali	20
1.7	Compatibilità e successori	20
1.8	Riepilogo	21
1.9	Esercizi	21
2	Registri e Organizzazione della Memoria	23
2.1	Introduzione	23
2.2	Registri General Purpose	23
2.2.1	Registro AX (Accumulatore)	23
2.2.2	Registro BX (Base)	24
2.2.3	Registro CX (Contatore)	24
2.2.4	Registro DX (Dati)	24
2.2.5	Tabella riassuntiva	25
2.3	Registri Puntatore e Indice	25
2.3.1	Stack Pointer (SP)	25
2.3.2	Base Pointer (BP)	25
2.3.3	Source Index (SI)	25
2.3.4	Destination Index (DI)	26
2.3.5	Instruction Pointer (IP)	26
2.4	Registri Segmento	26
2.4.1	Code Segment (CS)	26

2.4.2	Data Segment (DS)	26
2.4.3	Stack Segment (SS)	26
2.4.4	Extra Segment (ES)	27
2.5	Organizzazione della Memoria	27
2.5.1	Modello di memoria segmentata	27
2.5.2	Layout tipico di un programma .COM	28
2.5.3	Layout programma .EXE	28
2.6	Convenzioni di utilizzo	28
2.7	Esempi pratici	28
2.8	Riepilogo	29
2.9	Esercizi	30
3	Modalità di Indirizzamento	31
3.1	Introduzione	31
3.2	Modalità di indirizzamento	31
3.2.1	1. Immediate Addressing (Indirizzamento Immediato)	31
3.2.2	2. Register Addressing (Indirizzamento a Registro)	31
3.2.3	3. Direct Addressing (Indirizzamento Diretto)	32
3.2.4	4. Register Indirect Addressing (Indirizzamento Indiretto a Registro)	32
3.2.5	5. Based Addressing (Indirizzamento Base + Offset)	33
3.2.6	6. Indexed Addressing (Indirizzamento Indicizzato)	34
3.2.7	7. Based Indexed Addressing (Indirizzamento Base + Indice)	34
3.3	Tabella riassuntiva	35
3.4	Segmenti di default e override	35
3.5	Esempi pratici completi	36
3.6	Performance e ottimizzazione	37
3.7	Riepilogo	37
3.8	Esercizi	38
II	Set di Istruzioni 8086	39
4	Istruzioni di Trasferimento Dati	41
4.1	Introduzione	41
4.2	MOV — Move Data	41
4.3	XCHG — Exchange	42
4.4	LEA — Load Effective Address	42
4.5	PUSH e POP — Stack Operations	42
4.5.1	PUSH	42
4.5.2	POP	43
4.6	PUSHF e POPF — Flag Operations	44
4.7	Istruzioni di I/O	44
4.7.1	IN — Input from Port	44
4.7.2	OUT — Output to Port	44
4.8	XLAT — Translate	44
4.9	Riepilogo istruzioni	45
4.10	Esercizi	45

5 Istruzioni Aritmetiche	47
5.1 Introduzione	47
5.2 ADD e ADC — Addition	47
5.2.1 ADD	47
5.2.2 ADC — Add with Carry	47
5.3 SUB e SBB — Subtraction	48
5.3.1 SUB	48
5.3.2 SBB — Subtract with Borrow	48
5.4 INC e DEC	48
5.5 MUL e IMUL — Multiplication	48
5.5.1 MUL — Unsigned Multiplication	48
5.5.2 IMUL — Signed Multiplication	48
5.6 DIV e IDIV — Division	49
5.6.1 DIV — Unsigned Division	49
5.7 NEG e CMP	49
5.7.1 NEG — Negate (Complemento a due)	49
5.7.2 CMP — Compare	49
5.8 Esercizi	50
6 Istruzioni Logiche e Bit Manipulation	51
6.1 Istruzioni Logiche	51
6.1.1 AND	51
6.1.2 OR	51
6.1.3 XOR	51
6.1.4 NOT	51
6.1.5 TEST	52
6.2 Shift e Rotate	52
6.2.1 SHL/SHR — Shift Logico	52
6.2.2 SAL/SAR — Shift Aritmetico	52
6.2.3 ROL/ROR — Rotate	52
6.2.4 RCL/RCR — Rotate through Carry	52
6.3 Esercizi	52
7 Istruzioni di Controllo di Flusso	55
7.1 Jump Incondizionati	55
7.1.1 JMP — Unconditional Jump	55
7.2 Jump Condizionati	55
7.2.1 Jump basati su Zero Flag	55
7.2.2 Jump Unsigned	55
7.2.3 Jump Signed	55
7.2.4 Jump su singoli flag	56
7.3 Loop Instructions	56
7.3.1 LOOP	56
7.3.2 LOOPE/LOOPZ	56
7.3.3 LOOPNE/LOOPNZ	56
7.3.4 JCXZ	56
7.4 Procedure	57

7.4.1	CALL	57
7.4.2	RET	57
7.5	Interrupt	57
7.5.1	INT	57
7.5.2	IRET	58
7.6	Tabella Jump Condizionati	58
7.7	Esercizi	58

III Tecniche di Programmazione **59**

8 Procedure e Gestione dello Stack **61**

8.1	Lo Stack dell'8086	61
8.1.1	Operazioni fondamentali	61
8.2	Chiamate a Procedura	61
8.2.1	Convenzioni di chiamata	61
8.2.2	Passaggio parametri via stack	62
8.2.3	Stack frame layout	62
8.3	Variabili Locali	62
8.4	Ricorsione	63
8.5	Esercizi	63

9 Operazioni su Stringhe **65**

9.1	Introduzione	65
9.2	Registri per Stringhe	65
9.2.1	CLD e STD	65
9.3	Istruzioni Base	65
9.3.1	MOVS/MOVSW — Move String	65
9.3.2	LODSB/LODSW — Load String	66
9.3.3	STOSB/STOSW — Store String	66
9.3.4	CMPSB/CMPSW — Compare String	66
9.3.5	SCASB/SCASW — Scan String	66
9.4	Prefissi REP	66
9.4.1	REP	66
9.4.2	REPE/REPZ	66
9.4.3	REPNE/REPNZ	66
9.5	Esempi Pratici	67
9.6	Esercizi	68

10 Sistema di Interrupt **69**

10.1	Introduzione agli Interrupt	69
10.2	Tipi di Interrupt	69
10.2.1	Hardware Interrupts	69
10.2.2	Software Interrupts	69
10.2.3	Exceptions	69
10.3	Interrupt Vector Table (IVT)	69
10.4	Interrupt DOS (INT 21h)	69

10.4.1	Funzione 01h: Input carattere con echo	69
10.4.2	Funzione 02h: Output carattere	70
10.4.3	Funzione 09h: Stampa stringa	70
10.4.4	Funzione 4Ch: Termina programma	70
10.5	Interrupt BIOS	70
10.5.1	INT 10h: Video Services	70
10.5.2	INT 16h: Keyboard Services	71
10.5.3	INT 13h: Disk Services	71
10.6	Gestione Custom Interrupt	71
10.7	Esercizi	72
11	I/O e Interfacciamento Hardware	73
11.1	Introduzione	73
11.2	Istruzioni I/O	73
11.2.1	IN e OUT	73
11.2.2	Porte del PIC	73
11.3	Programmable Interval Timer (8253/8254)	74
11.3.1	Porte del PIT	74
11.4	Porta Parallela (LPT)	75
11.4.1	Porte LPT1	75
11.5	Porta Seriale (COM)	75
11.5.1	UART 8250/16550	75
11.6	Tastiera (8042)	77
11.6.1	Porte tastiera	77
11.7	VGA (Video Graphics Array)	77
11.7.1	Accesso diretto a memoria video	77
11.8	Esercizi	77
IV	Progetti e Applicazioni	79
12	Progetti Applicativi	81
12.1	Progetto 1: Calcolatrice a 16 bit	81
12.2	Progetto 2: Ordinamento Array	83
12.3	Progetto 3: Editor di Testo Semplice	84
12.4	Progetto 4: Gioco Snake	84
12.5	Progetto 5: Bootloader	84
12.6	Esercizi	84
13	Esercizi Progressivi	87
13.1	Livello Base	87
13.2	Livello Intermedio	87
13.3	Livello Avanzato	88
13.4	Progetti Integrati	89
13.5	Sfide Avanzate	89
13.6	Riepilogo Competenze	90

A	Soluzioni agli Esercizi	91
A.1	Soluzioni Capitolo 1 — Architettura	91
A.1.1	Esercizio 1.1	91
A.1.2	Esercizio 1.2	92
A.1.3	Esercizio 1.5	92
A.2	Soluzioni Capitolo 2 — Registri	92
A.2.1	Esercizio 2.1	92
A.2.2	Esercizio 2.2	92
A.2.3	Esercizio 2.3	92
A.2.4	Esercizio 2.4	93
A.3	Soluzioni Capitolo 4 — Trasferimento Dati	93
A.3.1	Esercizio 4.1	93
A.3.2	Esercizio 4.2	93
A.4	Soluzioni Capitolo 5 — Aritmetiche	93
A.4.1	Esercizio 5.1	93
A.4.2	Esercizio 5.2	94
A.4.3	Esercizio 5.3	94
A.4.4	Esercizio 5.5	94
A.5	Soluzioni Capitolo 7 — Controllo Flusso	94
A.5.1	Esercizio 7.1	94
A.5.2	Esercizio 7.2	95
A.5.3	Esercizio 7.3	95
A.6	Soluzioni Capitolo 9 — Stringhe	95
A.6.1	Esercizio 9.1	95
A.6.2	Esercizio 9.4	96
A.7	Note Finali	97
B	Quick Reference	99
B.1	Registri 8086	99
B.1.1	General Purpose (16 bit)	99
B.1.2	Puntatore e Indice	99
B.1.3	Segmento	99
B.2	Flag Register	100
B.3	Modalità Indirizzamento	100
B.4	Istruzioni per Categoria	100
B.4.1	Trasferimento Dati	100
B.4.2	Aritmetiche	100
B.4.3	Logiche	101
B.4.4	Controllo Flusso	101
B.4.5	Stringhe	101
B.5	Interrupt DOS (INT 21h)	102
B.6	Interrupt BIOS	102
B.6.1	INT 10h (Video)	102
B.6.2	INT 16h (Keyboard)	102
B.7	Struttura Programma .COM	102
B.8	Struttura Programma .EXE	103

B.9	Calcolo Indirizzo Fisico	103
B.10	Tabella ASCII (estratto)	104
B.11	Scan Code Tastiera (estratto)	104
B.12	Porte I/O Comuni	104
C	Bibliografia e Risorse	105
C.1	Manuali Ufficiali	105
C.2	Libri di Testo	105
C.3	Risorse Online	106
C.3.1	Tutorial e Guide	106
C.3.2	Emulatori e Tools	106
C.3.3	Simulatori Online	106
C.4	Comunità e Forum	107
C.5	Video Corsi	107
C.6	Progetti e Codice Esempio	107
C.7	Specifiche Hardware	108
C.8	Reverse Engineering	108
C.9	Sicurezza e Exploit Development	108
C.10	Standard e Riferimenti	109
C.11	Risorse in Italiano	109
C.12	Riviste e Pubblicazioni	109
C.13	Note Finali	109

Parte I

Fondamenti dell'Architettura 8086

Prefazione

A chi è rivolto questo manuale

Questo manuale di programmazione Assembly 8086 è pensato per studenti di istituti tecnici superiori che desiderano comprendere a fondo il funzionamento del microprocessore e acquisire competenze di programmazione a basso livello. Il corso fornisce una solida base teorica e pratica sulla programmazione Assembly, fondamentale per chi intende specializzarsi in sistemi embedded, reverse engineering, ottimizzazione del codice o sicurezza informatica.

Prerequisiti

Prima di affrontare questo corso è necessario avere:

- Conoscenza di base della programmazione (C o linguaggi simili)
- Comprensione dei sistemi di numerazione (binario, esadecimale)
- Nozioni di base di architettura dei computer
- Familiarità con i concetti di variabili, cicli e condizioni

Obiettivi del corso

Al termine del corso lo studente sarà in grado di:

1. **Comprendere l'architettura 8086:** registri, segmenti, modalità di indirizzamento
2. **Scrivere programmi Assembly:** utilizzando il set completo di istruzioni
3. **Gestire procedure e stack:** chiamate a funzione, passaggio parametri
4. **Programmare interrupt:** gestione I/O e servizi DOS
5. **Ottimizzare il codice:** tecniche per efficienza e performance
6. **Debugging:** identificare e correggere errori in Assembly

Strumenti utilizzati

Per seguire il corso è necessario disporre di:

- **Emulatore 8086:** EMU8086, DOSBox, NASM, MASM
- **Debugger:** Debug.exe, TD (Turbo Debugger), GDB
- **Editor di testo:** Notepad++, VS Code, TASM IDE
- **Sistema operativo:** Windows/Linux/macOS con emulatore DOS

EMU8086 (Consigliato)

EMU8086 è un ambiente integrato che combina assembler, disassembler ed emulatore. Permette di:

- Scrivere e assemblare codice Assembly 8086
- Eseguire programmi step-by-step
- Visualizzare registri e memoria in tempo reale
- Debuggare con breakpoint

Download: <https://emu8086-microprocessor-emulator.en.softonic.com/>

NASM (Netwide Assembler)

NASM è un assembler open-source multi-piattaforma:

```
1 # Installazione su Linux
2 sudo apt-get install nasm
3
4 # Assemblaggio e linking
5 nasm -f elf program.asm -o program.o
6 ld -m elf_i386 program.o -o program
7
8 # Esecuzione
9 ./program
```

DOSBox

Per eseguire programmi .COM e .EXE su sistemi moderni:

```
1 # Installazione
2 sudo apt-get install dosbox # Linux
3 brew install dosbox        # macOS
4
5 # Esecuzione
6 dosbox
```

```
7 mount c ~/assembly
8 c:
9 program.exe
```

Struttura del manuale

Il libro è organizzato in 4 parti:

Parte I — Fondamenti (Cap. 1-3) Architettura 8086, registri, memoria, modalità di indirizzamento

Parte II — Set di Istruzioni (Cap. 4-7) Istruzioni di trasferimento dati, aritmetiche, logiche, controllo di flusso

Parte III — Tecniche Avanzate (Cap. 8-11) Procedure, stack, stringhe, interrupt, I/O hardware

Parte IV — Applicazioni (Cap. 12-13) Progetti completi ed esercizi pratici

Convenzioni tipografiche

- `MOV AX, BX` — Codice Assembly inline
- **Registro AX** — Termini tecnici importanti
- *little-endian* — Concetti da memorizzare

Nota

I codici di esempio sono testati con EMU8086 e NASM. Alcune istruzioni potrebbero richiedere adattamenti per altri assembler.

Metodologia didattica

Ogni capitolo segue questa struttura:

1. **Introduzione teorica:** Spiegazione del concetto
2. **Esempi commentati:** Codice Assembly con spiegazioni dettagliate
3. **Esercizi guidati:** Problemi con soluzione passo-passo
4. **Esercizi proposti:** Sfide per consolidare l'apprendimento
5. **Riepilogo:** Punti chiave del capitolo

Supporto e risorse

- Manuale Intel 8086: <https://www.intel.com/>
- OSDev Wiki: <https://wiki.osdev.org/>
- Assembly Language Tutorial: https://www.tutorialspoint.com/assembly_programming/
- Stack Overflow: Tag [assembly] e [x86]

Ringraziamenti

Si ringraziano tutti gli studenti che hanno contribuito con feedback e suggerimenti al miglioramento di questo materiale didattico.

Buono studio!
Gli autori

Capitolo 1

Architettura del Microprocessore 8086

1.1 Introduzione

Il microprocessore Intel 8086, introdotto nel 1978, rappresenta una pietra miliare nella storia dell'informatica. È il primo processore della famiglia x86, architettura che ancora oggi domina il mercato dei personal computer. Comprendere l'8086 significa gettare le basi per la programmazione a basso livello su processori moderni.

Definizione

Il **microprocessore 8086** è una CPU a 16 bit con bus dati a 16 bit e bus indirizzi a 20 bit, capace di indirizzare fino a 1 MB di memoria ($2^{20} = 1.048.576$ byte).

1.2 Caratteristiche principali

1.2.1 Specifiche tecniche

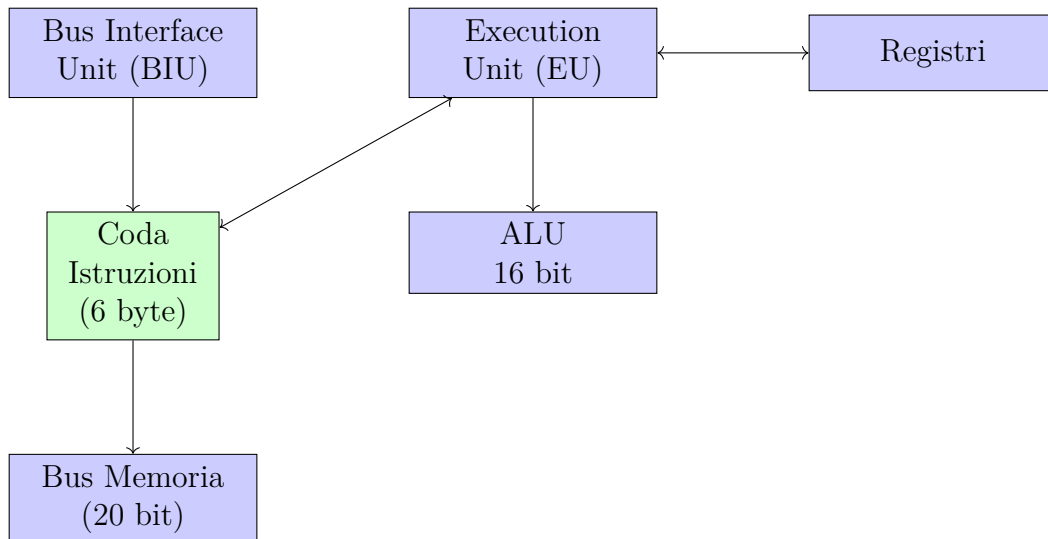
Caratteristica	Valore
Architettura	16 bit
Bus dati	16 bit
Bus indirizzi	20 bit
Memoria indirizzabile	1 MB (1.048.576 byte)
Frequenza clock	5-10 MHz
Registri general purpose	4 imes 16 bit
Registri segmento	4 imes 16 bit
Numero transistor	29.000
Tecnologia	NMOS 3 μ m

Tabella 1.1: Specifiche tecniche Intel 8086

1.2.2 Organizzazione interna

L'8086 utilizza un'architettura pipeline a due stadi:

- **Bus Interface Unit (BIU):** Gestisce il prelievo delle istruzioni dalla memoria
- **Execution Unit (EU):** Esegue le istruzioni decodificate



Nota

La **coda di istruzioni** (instruction queue) permette alla BIU di precaricare fino a 6 byte di istruzioni mentre la EU esegue l'istruzione corrente. Questo meccanismo di *prefetch* migliora le performance.

1.3 Modalità di funzionamento

L'8086 può operare in due modalità:

1.3.1 Modalità minima

Sistema monoprocessoire dove l'8086 genera autonomamente tutti i segnali di controllo.

1.3.2 Modalità massima

Sistema multiprocessore con coprocessore 8087 (FPU) e controller DMA 8237. Richiede circuiti esterni per la generazione dei segnali.

1.4 Memoria segmentata

L'8086 utilizza un modello di memoria **segmentata** per indirizzare 1 MB con registri a 16 bit.

Definizione

Un **segmento** è un blocco di memoria di massimo 64 KB (2^{16} byte). L'indirizzo fisico viene calcolato combinando un registro segmento e un offset.

1.4.1 Calcolo indirizzo fisico

L'indirizzo fisico a 20 bit si ottiene con la formula:

$$\text{Indirizzo Fisico} = (\text{Segmento} \times 16) + \text{Offset}$$

Equivalentemente:

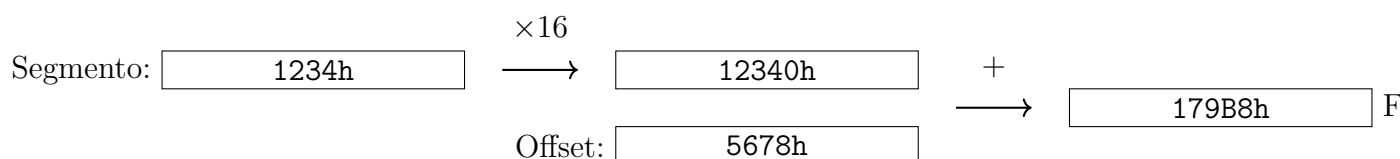
$$\text{Indirizzo Fisico} = (\text{Segmento} \ll 4) + \text{Offset}$$

Esempio

Calcolare l'indirizzo fisico per DS:SI = 1234h:5678h:

$$\begin{aligned}\text{Segmento} &= 1234h \\ \text{Offset} &= 5678h \\ \text{Indirizzo Fisico} &= (1234h \times 10h) + 5678h \\ &= 12340h + 5678h \\ &= 179B8h\end{aligned}$$

In decimale: $179B8h = 96.696$ byte.

1.4.2 Rappresentazione grafica**1.5 Flag register**

Il registro FLAGS (16 bit) contiene bit di stato e controllo:

Attenzione

I flag **TF**, **IF** e **DF** sono *flag di controllo* impostabili dal programmatore. Gli altri sono *flag di stato* modificati automaticamente dalle istruzioni.

Bit	Nome	Descrizione
0	CF	Carry Flag: riporto/prestito in operazioni aritmetiche
2	PF	Parity Flag: parità del byte meno significativo
4	AF	Auxiliary Carry: riporto bit 3 (BCD)
6	ZF	Zero Flag: risultato zero
7	SF	Sign Flag: bit di segno del risultato
8	TF	Trap Flag: modalità debug single-step
9	IF	Interrupt Flag: abilita interrupt mascherabili
10	DF	Direction Flag: direzione operazioni su stringhe
11	OF	Overflow Flag: overflow in aritmetica con segno

Tabella 1.2: Flag principali del registro FLAGS

1.6 Pinout e segnali

L'8086 ha 40 pin divisi in:

- **AD0-AD15:** Bus multiplexato indirizzo/dati (16 bit)
- **A16-A19:** Bit superiori indirizzo (4 bit)
- **Control signals:** ALE, RD, WR, IO/M, etc.
- **Interrupt:** INTR, NMI, INTA
- **Power:** VCC, GND
- **Clock:** CLK

Nota

Il bus AD0-AD15 è **multiplexato**: trasporta indirizzi durante T1 e dati durante T2-T4 del ciclo bus. Il segnale ALE (Address Latch Enable) indica quando il bus contiene un indirizzo valido.

1.7 Compatibilità e successori

L'8086 ha dato origine a una famiglia di processori:

- **8088:** Variante con bus dati a 8 bit (usato nell'IBM PC)
- **80186/80188:** Versioni migliorate con periferiche integrate
- **80286:** Modalità protetta, 16 MB indirizzabili
- **80386:** Prima CPU x86 a 32 bit

- **Pentium e successivi:** Evoluzione moderna dell'architettura x86

Definizione

La **retrocompatibilità** dell'architettura x86 permette ai processori moderni di eseguire codice Assembly 8086 in *modalità reale* (real mode).

1.8 Riepilogo

- L'8086 è un processore a 16 bit con bus indirizzi a 20 bit
- Può indirizzare 1 MB di memoria tramite segmentazione
- Utilizza pipeline BIU/EU per migliorare le performance
- Il registro FLAGS contiene 9 flag di stato e controllo
- È il capostipite dell'architettura x86 ancora in uso

1.9 Esercizi

Esercizio 1.1

Calcolare l'indirizzo fisico per i seguenti indirizzi segmentati:

- a) CS:IP = 2000h:1000h
- b) DS:SI = FFFFh:0010h
- c) SS:SP = 3000h:FFFEh

Esercizio 1.2

Quanti segmenti distinti di 64 KB possono coesistere nello spazio di indirizzamento dell'8086?

Esercizio 1.3

Spiegare perché l'8086 ha bisogno di 20 bit per l'indirizzo fisico ma utilizza registri a 16 bit.

Esercizio 1.4

Descrivere la differenza tra modalità minima e massima dell'8086.

Esercizio 1.5

Quali flag vengono influenzati dall'istruzione `ADD AX, BX`? Spiegare in quali condizioni ciascun flag viene impostato a 1.

Capitolo 2

Registri e Organizzazione della Memoria

2.1 Introduzione

I registri sono celle di memoria ultrarapide integrate nel processore. L'8086 dispone di 14 registri a 16 bit, suddivisi in 4 categorie: registri generali, registri puntatore, registri indice e registri segmento. La conoscenza approfondita dei registri è fondamentale per scrivere codice Assembly efficiente.

2.2 Registri General Purpose

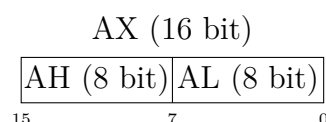
L'8086 ha 4 registri general purpose a 16 bit, ciascuno divisibile in due registri a 8 bit.

2.2.1 Registro AX (Accumulatore)

Definizione

AX è il registro *accumulatore*, utilizzato per operazioni aritmetiche, I/O e moltiplicazioni/divisioni.

- **AX** (16 bit): Accumulatore completo
- **AH** (8 bit): Byte alto (High)
- **AL** (8 bit): Byte basso (Low)



Usi tipici:

- Operazioni aritmetiche generiche

- Istruzioni I/O (`IN AL, port / OUT port, AL`)
- Moltiplicazione: risultato in `AX` (8 bit) o `DX:AX` (16 bit)
- Divisione: dividendo in `AX` (8 bit) o `DX:AX` (16 bit)

2.2.2 Registro **BX** (Base)

- **BX**: Registro base per indirizzamento indiretto
- **BH** / **BL**: Byte alto/basso

Usi tipici:

- Indirizzamento di array: `MOV AL, [BX]`
- Puntatore a strutture dati
- Calcoli generici

2.2.3 Registro **CX** (Contatore)

- **CX**: Registro contatore per loop
- **CH** / **CL**: Byte alto/basso

Usi tipici:

- Contatore in istruzioni `LOOP`, `LOOPE`, `LOOPNE`
- Contatore in operazioni su stringhe con prefisso `REP`
- Shift/rotate: `CL` specifica il numero di posizioni

2.2.4 Registro **DX** (Dati)

- **DX**: Registro dati per I/O e operazioni estese
- **DH** / **DL**: Byte alto/basso

Usi tipici:

- Indirizzamento porte I/O: `IN AL, DX / OUT DX, AL`
- Moltiplicazione 16 bit: risultato alto in `DX`, basso in `AX`
- Divisione 16 bit: dividendo in `DX:AX`

16 bit	8 bit H	8 bit L	Uso principale
AX	AH	AL	Accumulatore, I/O, moltiplicazioni
BX	BH	BL	Base per indirizzamento, puntatori
CX	CH	CL	Contatore loop, shift count
DX	DH	DL	Dati, porte I/O, divisioni

Tabella 2.1: Registri general purpose

2.2.5 Tabella riassuntiva

2.3 Registri Puntatore e Indice

Questi registri sono utilizzati per l'indirizzamento della memoria e non sono divisibili in byte.

2.3.1 Stack Pointer (SP)

Definizione

SP punta alla cima dello stack nel segmento SS. Lo stack cresce verso indirizzi decrescenti.

- Utilizzato con PUSH e POP
- Automaticamente decrementato da PUSH, incrementato da POP
- Indirizzo stack: SS:SP

2.3.2 Base Pointer (BP)

- **BP**: Puntatore base per accedere parametri e variabili locali
- Tipicamente usato con segmento SS: [BP+offset]
- Fondamentale nelle chiamate a procedure

2.3.3 Source Index (SI)

- **SI**: Indice sorgente per operazioni su stringhe
- Usato con segmento DS: DS:SI
- Auto-incremento/decremento con istruzioni LODS, MOVS, CMPS

2.3.4 Destination Index (DI)

- **DI**: Indice destinazione per operazioni su stringhe
- Usato con segmento ES: ES:DI
- Auto-incremento/decremento con istruzioni STOS, MOVS, SCAS

2.3.5 Instruction Pointer (IP)

Definizione

IP contiene l'offset della prossima istruzione da eseguire nel segmento CS. Non è direttamente modificabile, ma cambia con istruzioni di salto e chiamate.

- Indirizzo istruzione corrente: CS:IP
- Modificato da: JMP, CALL, RET, INT, IRET
- Automaticamente incrementato dopo ogni istruzione

2.4 Registri Segmento

I 4 registri segmento definiscono le basi dei segmenti di memoria.

2.4.1 Code Segment (CS)

- **CS**: Segmento codice, contiene le istruzioni del programma
- Indirizzo istruzione: CS:IP
- Modificabile solo con JMP FAR, CALL FAR, RET FAR

2.4.2 Data Segment (DS)

- **DS**: Segmento dati, contiene variabili globali
- Usato implicitamente da: [BX], [SI], [DI]
- Modificabile con MOV DS, AX

2.4.3 Stack Segment (SS)

- **SS**: Segmento stack, area per PUSH/POP
- Indirizzo cima stack: SS:SP
- Usato implicitamente da [BP]

2.4.4 Extra Segment (ES)

- **ES**: Segmento extra per operazioni su stringhe
- Usato implicitamente da DI in STOS, MOVS, SCAS
- Utile per copiare dati tra segmenti

Attenzione

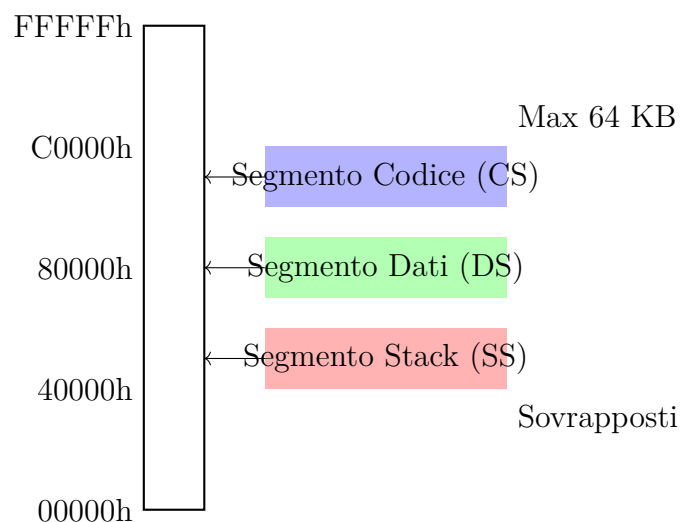
I registri segmento **non possono essere modificati direttamente**. Bisogna passare attraverso un registro general purpose:

```
1 ; CORRETTO
2 MOV AX, 1000h
3 MOV DS, AX
4
5 ; ERRORE - non consentito
6 MOV DS, 1000h
```

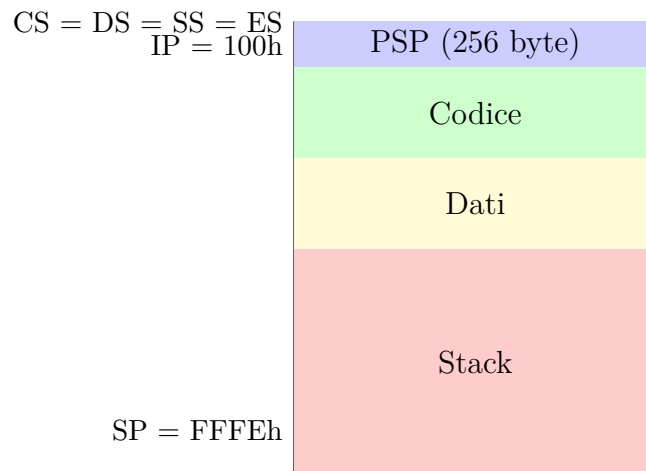
2.5 Organizzazione della Memoria

2.5.1 Modello di memoria segmentata

Lo spazio di indirizzamento dell'8086 (1 MB) è organizzato in segmenti sovrapposti:



2.5.2 Layout tipico di un programma .COM



Nota

Nei programmi **.COM**, tutti i registri segmento puntano allo stesso indirizzo. L'intera immagine del programma (codice, dati, stack) risiede in un unico segmento di 64 KB. Il Program Segment Prefix (PSP) occupa i primi 256 byte.

2.5.3 Layout programma .EXE

Nei file **.EXE** i segmenti sono separati:

- **CS**: Segmento codice (istruzioni)
- **DS**: Segmento dati (variabili globali)
- **SS**: Segmento stack (separato da codice e dati)
- **ES**: Tipicamente uguale a DS all'avvio

2.6 Convenzioni di utilizzo

2.7 Esempi pratici

Esempio

Sommare due numeri a 16 bit:

```

1 ; Somma: 1234h + 5678h = 68ACh
2 MOV AX, 1234h      ; AX = 1234h
3 MOV BX, 5678h      ; BX = 5678h
4 ADD AX, BX          ; AX = 68ACh, BX immutato

```

Operazione	Registri tipici
Moltiplicazione 8 bit	AL imes operando → AX
Moltiplicazione 16 bit	AX imes operando → DX:AX
Divisione 8 bit	AX ÷ operando → AL (quoziente), AH (resto)
Divisione 16 bit	DX:AX ÷ operando → AX (quoziente), DX (resto)
Loop counter	CX (decrementato automaticamente)
String source	DS:SI
String destination	ES:DI
Procedure stack frame	SS:BP
I/O a 8 bit fisso	AL + numero porta immediato
I/O variabile	AL/AX + DX (numero porta)

Tabella 2.2: Convenzioni d'uso dei registri

Esempio

Accesso a byte alto e basso:

```

1  MOV AX, 1234h      ; AX = 1234h
2  MOV BL, AL         ; BL = 34h (byte basso)
3  MOV BH, AH         ; BH = 12h (byte alto)
4  ; Ora BX = 1234h

```

Esempio

Indirizzamento indiretto con BX:

```

1  .DATA
2  array DB 10h, 20h, 30h, 40h
3
4  .CODE
5  MOV BX, OFFSET array ; BX punta all'array
6  MOV AL, [BX]         ; AL = 10h (primo elemento)
7  INC BX               ; BX punta al secondo elemento
8  MOV AL, [BX]         ; AL = 20h

```

2.8 Riepilogo

- L'8086 ha 14 registri a 16 bit
- I 4 registri general purpose (AX, BX, CX, DX) sono divisibili in byte
- SP punta allo stack, BP ai parametri, SI/DI alle stringhe
- I 4 registri segmento (CS, DS, SS, ES) definiscono le basi dei segmenti
- IP contiene l'offset della prossima istruzione
- I programmi .COM usano un unico segmento, i .EXE hanno segmenti separati

2.9 Esercizi

Esercizio 2.1

Dato $AX = 1234h$, scrivere le istruzioni per:

- a) Azzerare solo AL mantenendo AH invariato
- b) Azzerare solo AH mantenendo AL invariato
- c) Scambiare AH e AL

Esercizio 2.2

Spiegare la differenza tra `MOV AX, BX` e `MOV AX, [BX]`.

Esercizio 2.3

Perché non è possibile eseguire `MOV DS, 1000h`? Come si risolve?

Esercizio 2.4

Calcolare l'indirizzo fisico di $SS:SP = 2000h:FFFEh$. Se si esegue `PUSH AX`, qual è il nuovo valore di SP e l'indirizzo fisico corrispondente?

Esercizio 2.5

In un programma .COM, se tutti i segmenti valgono 1000h e $IP = 0100h$, qual è l'indirizzo fisico della prima istruzione?

Capitolo 3

Modalità di Indirizzamento

3.1 Introduzione

Le modalità di indirizzamento determinano come il processore calcola l'indirizzo effettivo degli operandi. L'8086 supporta 7 modalità di indirizzamento, ciascuna adatta a scenari specifici. La comprensione delle modalità di indirizzamento è cruciale per accedere efficientemente a variabili, array e strutture dati.

Definizione

Una **modalità di indirizzamento** specifica il metodo con cui il processore determina l'indirizzo di memoria (effective address) o il valore di un operando.

3.2 Modalità di indirizzamento

3.2.1 1. Immediate Addressing (Indirizzamento Immediato)

L'operando è una costante inclusa nell'istruzione stessa.

Sintassi: `MOV dest, costante`

```
1 MOV AX, 1234h      ; AX = 1234h (costante immediata)
2 MOV BL, 'A'        ; BL = 41h (codice ASCII di 'A')
3 ADD CX, 10         ; CX = CX + 10
```

Nota

I valori immediati sono codificati direttamente nell'istruzione, occupando 1 o 2 byte aggiuntivi. Sono i più veloci da accedere (nessun accesso a memoria).

3.2.2 2. Register Addressing (Indirizzamento a Registro)

L'operando risiede in un registro.

Sintassi: `MOV dest_reg, src_reg`

```
1 MOV AX, BX         ; AX = BX
2 ADD SI, DI         ; SI = SI + DI
```

```
3 MOV DL, CH ; DL = CH (byte alto di CX)
```

Vantaggi:

- Velocissime (nessun accesso a memoria)
- Istruzioni compatte (1-2 byte)

3.2.3 3. Direct Addressing (Indirizzamento Diretto)

L'indirizzo effettivo è specificato direttamente nell'istruzione.

Sintassi: MOV dest, [indirizzo]

```
1 .DATA
2 var1 DW 1234h
3
4 .CODE
5 MOV AX, [0200h] ; AX = contenuto di DS:0200h
6 MOV BX, var1    ; BX = contenuto di var1
7 MOV [1000h], CX ; Memoria DS:1000h = CX
```

Indirizzo effettivo:

EA = offset specificato nell'istruzione

Indirizzo fisico:

Fisico = DS × 16 + EA

Esempio

Se DS = 1000h e si esegue MOV AX, [0200h]:

$$\begin{aligned}
 \text{EA} &= 0200h \\
 \text{Fisico} &= 1000h \times 10h + 0200h \\
 &= 10000h + 0200h \\
 &= 10200h
 \end{aligned}$$

Il contenuto della locazione fisica 10200h viene caricato in AX.

3.2.4 4. Register Indirect Addressing (Indirizzamento Indiretto a Registro)

L'indirizzo è contenuto in un registro (BX, SI, DI, BP).

Sintassi: MOV dest, [registro]

Registri utilizzabili:

- [BX] segmento DS (default)
- [SI] segmento DS

- [DI] segmento DS
- [BP] segmento SS (eccezione!)

```

1 MOV BX, 1000h
2 MOV AL, [BX]      ; AL = byte da DS:1000h
3
4 MOV SI, 2000h
5 MOV AX, [SI]      ; AX = word da DS:2000h
6
7 MOV BP, 0FFFEh
8 MOV CX, [BP]      ; CX = word da SS:FFFEh (stack)

```

Attenzione

[BP] usa il segmento **SS** (stack), non DS! Per usare DS con BP:

```

1 MOV AX, DS:[BP]    ; Forza segmento DS

```

Uso tipico: Accesso ad array e puntatori

```

1 .DATA
2 array DB 10, 20, 30, 40, 50
3
4 .CODE
5 MOV BX, OFFSET array ; BX punta all'array
6 MOV AL, [BX]         ; AL = 10 (primo elemento)
7 INC BX
8 MOV AL, [BX]         ; AL = 20 (secondo elemento)

```

3.2.5 5. Based Addressing (Indirizzamento Base + Offset)

Combina un registro base (BX o BP) con un offset costante.

Sintassi: [BX+offset] o [BP+offset]

```

1 MOV AX, [BX+4]      ; AX = contenuto di DS:(BX+4)
2 MOV CX, [BP+6]      ; CX = contenuto di SS:(BP+6)
3 MOV [BX+10], DX     ; Memoria DS:(BX+10) = DX

```

Indirizzo effettivo:

EA = BX + offset (segmento DS)

EA = BP + offset (segmento SS)

Uso tipico: Accesso a campi di struct

```

1 ; Struttura: [nome 20 byte][età 1 byte][salario 2 byte]
2 MOV BX, OFFSET personal
3 MOV AL, [BX+20]      ; AL = età (offset 20)
4 MOV AX, [BX+21]      ; AX = salario (offset 21)

```

3.2.6 6. Indexed Addressing (Indirizzamento Indicizzato)

Combina un registro indice (SI o DI) con un offset costante.

Sintassi: [SI+offset] o [DI+offset]

```
1 MOV AX, [SI+2]      ; AX = contenuto di DS:(SI+2)
2 MOV BL, [DI+5]      ; BL = byte da DS:(DI+5)
```

Uso tipico: Iterazione su array multi-dimensionali

```
1 .DATA
2 matrix DW 1, 2, 3, 4, 5, 6      ; Array 2 imes3
3
4 .CODE
5 MOV SI, 0
6 MOV AX, matrix[SI+0]           ; Elemento (0,0)
7 MOV BX, matrix[SI+2]           ; Elemento (0,1)
8 MOV CX, matrix[SI+4]           ; Elemento (0,2)
9 ADD SI, 6                       ; Passa alla seconda riga
10 MOV AX, matrix[SI+0]           ; Elemento (1,0)
```

3.2.7 7. Based Indexed Addressing (Indirizzamento Base + Indice)

Combina un registro base (BX o BP) con un registro indice (SI o DI) e opzionalmente un offset.

Sintassi:

- [BX+SI]
- [BX+DI]
- [BP+SI]
- [BP+DI]
- [BX+SI+offset]

```
1 MOV AX, [BX+SI]      ; AX = DS:(BX+SI)
2 MOV CL, [BP+DI]      ; CL = SS:(BP+DI)
3 MOV DX, [BX+SI+10]    ; DX = DS:(BX+SI+10)
```

Indirizzo effettivo:

$$EA = BX/BP + SI/DI + \text{offset}$$

Uso tipico: Array bidimensionali, array di struct

```
1 ; Matrice 4 imes3 (word): accesso a elemento [riga][colonna]
2 ; BX = riga * 6 (3 word      imes 2 byte)
3 ; SI = colonna * 2
4
```

```

5  .DATA
6  matrix DW 1,2,3, 4,5,6, 7,8,9, 10,11,12 ; 4 righe imes 3
      colonne
7
8  .CODE
9  MOV BX, 6 ; Riga 1 (offset 6 byte)
10 MOV SI, 4 ; Colonna 2 (offset 4 byte)
11 MOV AX, matrix[BX+SI] ; AX = 6 (elemento [1][2])

```

3.3 Tabella riassuntiva

Modalità	Sintassi	EA	Esempio
Immediate	MOV AX, 10		Costante
Register	MOV AX, BX		Registro
Direct	[1234h]	1234h	Variabili globali
Reg. Indirect	[BX]	BX	Array, puntatori
Based	[BX+4]	BX+4	Struct
Indexed	[SI+4]	SI+4	Array offset
Based Indexed	[BX+SI+4]	BX+SI+4	Matrici, array struct

Tabella 3.1: Modalità di indirizzamento 8086

3.4 Segmenti di default e override

Registro	Segmento Default	Override
BX	DS	ES: [BX]
SI	DS	SS: [SI]
DI	DS (tranne stringhe)	CS: [DI]
BP	SS	DS: [BP]

Tabella 3.2: Segmenti di default

Esempio

Override di segmento:

```

1  MOV AX, DS:[BP+4] ; Forza DS invece di SS
2  MOV BX, ES:[BX+SI] ; Usa ES invece di DS
3  MOV CL, CS:[DI] ; Legge dal segmento codice

```

3.5 Esempi pratici completi

Esempio

Array di interi a 16 bit:

```

1  .DATA
2  numbers DW 100, 200, 300, 400, 500
3
4  .CODE
5  MOV SI, 0                ; Indice iniziale
6  MOV CX, 5                ; Numero elementi
7
8  loop_start:
9      MOV AX, numbers[SI]  ; Carica elemento corrente
10     ADD AX, 10            ; Incrementa di 10
11     MOV numbers[SI], AX  ; Salva risultato
12     ADD SI, 2            ; Prossimo elemento (word = 2
                           ; byte)
13     LOOP loop_start

```

Esempio

Struct con campi multipli:

```

1  ; Struct Studente: [matricola 2 byte][voto 1 byte][età 1
   ; byte]
2  .DATA
3  studente1 DW 12345
4             DB 28, 20
5
6  .CODE
7  MOV BX, OFFSET studente1
8  MOV AX, [BX+0]          ; AX = matricola (12345)
9  MOV AL, [BX+2]          ; AL = voto (28)
10 MOV AH, [BX+3]          ; AH = età (20)

```

Esempio

Matrice bidimensionale 3 imes4 (byte):

```

1  .DATA
2  ; Matrice 3 righe    imes 4 colonne
3  matrix DB 1,2,3,4, 5,6,7,8, 9,10,11,12
4
5  .CODE
6  ; Accesso a elemento [riga][colonna]
7  ; BX = riga * 4 (numero colonne)
8  ; SI = colonna
9
10 MOV BX, 4           ; Riga 1 (offset 4)
11 MOV SI, 2           ; Colonna 2
12 MOV AL, matrix[BX+SI] ; AL = 7 (elemento [1][2])

```

3.6 Performance e ottimizzazione

Modalità	Clock Cycles	Byte Istruzione
Register	2	2
Immediate	4	3-4
Direct	8-10	4
Reg. Indirect	8-10	2-3
Based/Indexed	11-12	3-4
Based Indexed	12-13	3-5

Tabella 3.3: Performance delle modalità di indirizzamento (approssimativa)

Nota

Le operazioni tra registri sono le più veloci. Minimizzare gli accessi a memoria migliora significativamente le performance.

3.7 Riepilogo

- L'8086 supporta 7 modalità di indirizzamento
- Immediate e Register sono le più veloci (nessun accesso a memoria)
- BX, SI, DI usano segmento DS; BP usa SS
- Based Indexed permette array multidimensionali e struct complesse
- Override di segmento con prefissi ES:, DS:, CS:, SS:

3.8 Esercizi

Esercizio 3.1

Classificare le seguenti istruzioni secondo la modalità di indirizzamento:

- a) `MOV AX, 1234h`
- b) `MOV BX, [SI]`
- c) `MOV CL, [BP+4]`
- d) `MOV DX, [BX+SI+2]`

Esercizio 3.2

Dato `DS = 2000h`, `BX = 0100h`, `SI = 0050h`, calcolare l'indirizzo fisico acceduto da:

```
1  MOV AX, [BX+SI+10]
```

Esercizio 3.3

Scrivere codice per accedere al terzo elemento (indice 2) di un array di word:

```
1  array DW 10, 20, 30, 40, 50
```

Esercizio 3.4

Spiegare perché `MOV AX, [BP]` usa il segmento `SS` e non `DS`. Come forzare l'uso di `DS`?

Esercizio 3.5

Scrivere codice per sommare tutti gli elementi di un array di 5 byte:

```
1  data DB 10, 20, 30, 40, 50
```

Usare indirizzamento indiretto e registro contatore.

Parte II

Set di Istruzioni 8086

Capitolo 4

Istruzioni di Trasferimento Dati

4.1 Introduzione

Le istruzioni di trasferimento dati spostano informazioni tra registri, memoria e porte I/O senza modificarne il contenuto. Sono le istruzioni più utilizzate in Assembly e costituiscono la base di ogni programma.

4.2 MOV — Move Data

Definizione

`MOV dest, src` copia il valore di `src` in `dest`. L'operando sorgente rimane invariato.

Sintassi:

```
1 MOV registro, registro
2 MOV registro, memoria
3 MOV memoria, registro
4 MOV registro, immediato
5 MOV memoria, immediato
```

Restrizioni:

- Non è consentito `MOV memoria, memoria`
- Non è consentito `MOV segmento, immediato`
- Gli operandi devono avere la stessa dimensione

Esempio

```

1 MOV AX, 1234h      ; AX = 1234h
2 MOV BX, AX         ; BX = AX = 1234h
3 MOV CL, 5          ; CL = 5
4 MOV [1000h], AL     ; Memoria DS:1000h = AL
5
6 ; ERRORI:
7 ; MOV AX, BL        ; Dimensioni diverse!
8 ; MOV DS, 1000h     ; No immediato in segmento!
9 ; MOV [SI], [DI]    ; No mem-to-mem!

```

4.3 XCHG — Exchange

Scambia il contenuto di due operandi.

```

1 XCHG AX, BX        ; Scambia AX con BX
2 XCHG AL, [SI]       ; Scambia AL con byte in DS:SI
3 XCHG CX, DX         ; Scambia CX con DX

```

Nota

XCHG è più veloce di tre MOV separate per lo scambio. Non può essere usato con valori immediati.

4.4 LEA — Load Effective Address

Carica l'indirizzo effettivo (offset) di un operando in un registro.

```

1 .DATA
2 buffer DB 100 DUP(0)
3
4 .CODE
5 LEA BX, buffer      ; BX = offset di buffer (equivalente a
6                      ; OFFSET)
7 LEA SI, [BX+10]     ; SI = BX + 10 (calcolo offset)

```

Differenza tra LEA e OFFSET:

```

1 MOV BX, OFFSET var  ; BX = indirizzo di var (compile-time)
2 LEA BX, var         ; BX = indirizzo di var (run-time)
3 LEA SI, [BX+DI+4]   ; OFFSET non può fare calcoli run-time!

```

4.5 PUSH e POP — Stack Operations

4.5.1 PUSH

Inserisce un valore nello stack.

```
1  PUSH  AX                ; Decrementa SP di 2, salva AX
2  PUSH  BX
3  PUSH  DS
4  PUSH  1234h             ; (solo 80186+)
```

Operazione:

1. $SP = SP - 2$
2. Memoria[SS:SP] = operando

4.5.2 POP

Estrae un valore dallo stack.

```
1  POP  AX                ; Carica AX da stack, incrementa SP di 2
2  POP  DS
3  POP  word ptr [BX]
```

Operazione:

1. Operando = Memoria[SS:SP]
2. $SP = SP + 2$

Attenzione

Lo stack cresce verso indirizzi decrescenti! PUSH decrementa SP, POP incrementa SP.

Esempio

Salvare e ripristinare registri:

```
1  ; Salva contesto
2  PUSH AX
3  PUSH BX
4  PUSH CX
5
6  ; Usa registri per operazioni
7  MOV AX, 100
8  ADD BX, AX
9
10 ; Ripristina contesto (ordine inverso!)
11 POP CX
12 POP BX
13 POP AX
```

4.6 PUSHF e POPF — Flag Operations

```
1 PUSHF          ; Salva registro FLAGS nello stack
2 POPF           ; Ripristina FLAGS dallo stack
```

Utile per salvare lo stato dei flag prima di operazioni che li modificano.

4.7 Istruzioni di I/O

4.7.1 IN — Input from Port

```
1 IN AL, porta    ; Legge byte da porta (0-255)
2 IN AX, porta    ; Legge word da porta
3 IN AL, DX       ; Legge da porta in DX (0-65535)
4 IN AX, DX
```

4.7.2 OUT — Output to Port

```
1 OUT porta, AL   ; Scrive byte su porta
2 OUT porta, AX   ; Scrive word su porta
3 OUT DX, AL      ; Scrive su porta in DX
4 OUT DX, AX
```

Esempio

Lettura e scrittura porta I/O:

```
1 ; Legge da porta 60h (tastiera)
2 IN AL, 60h      ; AL = scan code
3
4 ; Scrive su porta 3F8h (COM1)
5 MOV DX, 3F8h
6 MOV AL, 'A'
7 OUT DX, AL      ; Invia 'A' alla seriale
```

4.8 XLAT — Translate

Traduce un byte usando una tabella di lookup.

```
1 XLAT            ; AL = DS:[BX+AL]
```

Esempio

Conversione da esadecimale a ASCII:

```

1  .DATA
2  hex_table DB '0123456789ABCDEF'
3
4  .CODE
5  MOV BX, OFFSET hex_table
6  MOV AL, 0Ah          ; Valore 10
7  XLAT                 ; AL = 'A' (hex_table[10])

```

4.9 Riepilogo istruzioni

Istruzione	Sintassi	Descrizione
MOV	MOV dest, src	Copia dati
XCHG	XCHG op1, op2	Scambia operandi
LEA	LEA reg, mem	Carica indirizzo effettivo
PUSH	PUSH operando	Inserisce nello stack
POP	POP operando	Estrae dallo stack
PUSHF	PUSHF	Salva FLAGS
POPF	POPF	Ripristina FLAGS
IN	IN AL/AX, porta	Input da porta
OUT	OUT porta, AL/AX	Output su porta
XLAT	XLAT	Traduce byte con tabella

Tabella 4.1: Istruzioni di trasferimento dati

4.10 Esercizi

Esercizio 4.1

Scrivere codice per scambiare i valori di AX e BX senza usare XCHG.

Esercizio 4.2

Dato lo stack inizialmente vuoto e SP = FFFEh, tracciare il valore di SP dopo:

```

1  PUSH AX
2  PUSH BX
3  POP CX
4  PUSH DX

```

Esercizio 4.3

Spiegare la differenza tra:

```
1 MOV BX, OFFSET array
2 LEA BX, array
3 LEA SI, [BX+DI+4]
```

Esercizio 4.4

Scrivere codice per convertire le cifre 0-15 in caratteri esadecimali '0'-'F' usando XLAT.

Esercizio 4.5

Perché non è consentito `MOV DS, 1000h`? Come si risolve?

Capitolo 5

Istruzioni Aritmetiche

5.1 Introduzione

Le istruzioni aritmetiche eseguono operazioni matematiche su operandi interi. L'8086 supporta addizione, sottrazione, moltiplicazione, divisione e operazioni BCD.

5.2 ADD e ADC — Addition

5.2.1 ADD

```
1 ADD dest, src      ; dest = dest + src
2 ADD AX, BX         ; AX = AX + BX
3 ADD AL, 5          ; AL = AL + 5
4 ADD word ptr [BX], 100
```

Flag modificati: CF, ZF, SF, OF, PF, AF

5.2.2 ADC — Add with Carry

```
1 ADC dest, src      ; dest = dest + src + CF
```

Usato per addizioni multi-precisione (numeri > 16 bit).

Esempio

Somma a 32 bit (DX:AX + CX:BX):

```
1 ; DX:AX = 12345678h, CX:BX = 9ABCDEF0h
2 ADD AX, BX      ; Somma parte bassa
3 ADC DX, CX      ; Somma parte alta + carry
4 ; Risultato in DX:AX = ACF13568h
```

5.3 SUB e SBB — Subtraction

5.3.1 SUB

```
1 SUB dest, src           ; dest = dest - src
2 SUB AX, 10
3 SUB CX, BX
```

5.3.2 SBB — Subtract with Borrow

```
1 SBB dest, src           ; dest = dest - src - CF
```

5.4 INC e DEC

```
1 INC operando           ; operando = operando + 1
2 DEC operando           ; operando = operando - 1
```

Nota

INC e DEC **non modificano** il Carry Flag! Più veloci di ADD/SUB con 1.

5.5 MUL e IMUL — Multiplication

5.5.1 MUL — Unsigned Multiplication

Operando	Operazione	Risultato
8 bit	AL imes operando	AX
16 bit	AX imes operando	DX:AX

```
1 MOV AL, 10
2 MOV BL, 20
3 MUL BL           ; AX = AL      imes BL = 200 (C8h)
4
5 MOV AX, 1000
6 MOV BX, 50
7 MUL BX           ; DX:AX = AX    imes BX = 50000
```

5.5.2 IMUL — Signed Multiplication

Stessa sintassi di MUL, ma interpreta gli operandi come numeri con segno.

5.6 DIV e IDIV — Division

5.6.1 DIV — Unsigned Division

Dividendo	Divisore	Quoziente	Resto
AX	8 bit	AL	AH
DX:AX	16 bit	AX	DX

```

1 MOV AX, 100
2 MOV BL, 7
3 DIV BL                ; AL = 14 (quoziente), AH = 2 (resto)
4
5 MOV DX, 0
6 MOV AX, 50000
7 MOV BX, 100
8 DIV BX                ; AX = 500, DX = 0

```

Attenzione

Prima di DIV a 16 bit, azzerare DX se il dividendo è solo in AX! Divisione per zero causa interrupt 0 (errore).

5.7 NEG e CMP

5.7.1 NEG — Negate (Complemento a due)

```

1 NEG operando          ; operando = -operando
2 NEG AL                 ; AL = -AL

```

5.7.2 CMP — Compare

```

1 CMP op1, op2          ; Esegue op1 - op2 e imposta flag (senza
    salvare)

```

Usato con salti condizionali:

```

1 CMP AX, 10
2 JE equal               ; Salta se AX == 10
3 JG greater            ; Salta se AX > 10 (signed)
4 JA above              ; Salta se AX > 10 (unsigned)

```

5.8 Esercizi

Esercizio 5.1

Calcolare 123×45 usando MUL.

Esercizio 5.2

Dividere 1000 per 7 e trovare quoziente e resto.

Esercizio 5.3

Sommare due numeri a 32 bit: $12345678h + ABCDEF01h$.

Esercizio 5.4

Spiegare perché INC AX è preferibile ad ADD AX, 1.

Esercizio 5.5

Scrivere codice per calcolare il valore assoluto di un numero con segno in AX.

Capitolo 6

Istruzioni Logiche e Bit Manipulation

6.1 Istruzioni Logiche

6.1.1 AND

```
1 AND dest, src ; dest = dest AND src (bit a bit)
```

Uso: mascherare bit, testare bit, azzerare bit.

Esempio

```
1 MOV AL, 11001101b
2 AND AL, 00001111b ; AL = 00001101b (maschera nibble basso)
```

6.1.2 OR

```
1 OR dest, src ; dest = dest OR src
```

Uso: impostare bit a 1.

6.1.3 XOR

```
1 XOR dest, src ; dest = dest XOR src
```

Uso: toggle bit, azzerare registro.

```
1 XOR AX, AX ; Azzeramento veloce (AX = 0)
```

6.1.4 NOT

```
1 NOT operando ; Complemento a uno (inverte tutti i bit)
```

6.1.5 TEST

```
1  TEST op1, op2      ; Esegue AND e imposta flag (senza salvare)
```

6.2 Shift e Rotate

6.2.1 SHL/SHR — Shift Logico

```
1  SHL dest, count    ; Shift left (moltiplicazione per 2^count)
2  SHR dest, count    ; Shift right (divisione per 2^count)
```

6.2.2 SAL/SAR — Shift Aritmetico

```
1  SAL dest, count    ; Come SHL
2  SAR dest, count    ; Preserva bit di segno
```

6.2.3 ROL/ROR — Rotate

```
1  ROL dest, count    ; Rotate left (bit escono e rientrano)
2  ROR dest, count    ; Rotate right
```

6.2.4 RCL/RCR — Rotate through Carry

```
1  RCL dest, count    ; Rotate left includendo CF
2  RCR dest, count    ; Rotate right includendo CF
```

Esempio

Moltiplicazione veloce per 8:

```
1  MOV AX, 5
2  SHL AX, 3      ; AX = 5 * 8 = 40
```

6.3 Esercizi

Esercizio 6.1

Azzerare i bit 0, 2, 4 di AL usando AND.

Esercizio 6.2

Impostare i bit 3, 5, 7 di BL usando OR.

Esercizio 6.3

Testare se il bit 6 di AL è 1 usando TEST.

Esercizio 6.4

Moltiplicare AX per 16 usando shift.

Esercizio 6.5

Spiegare la differenza tra SHR e SAR con numeri negativi.

Capitolo 7

Istruzioni di Controllo di Flusso

7.1 Jump Incondizionati

7.1.1 JMP — Unconditional Jump

```
1  JMP label           ; Salta a label
2  JMP SHORT label     ; Salto corto (-128/+127 byte)
3  JMP NEAR label      ; Salto near (stesso segmento)
4  JMP FAR label       ; Salto far (cambia CS:IP)
```

7.2 Jump Condizionati

Basati su flag del registro FLAGS.

7.2.1 Jump basati su Zero Flag

```
1  JE/JZ label         ; Jump if Equal/Zero (ZF=1)
2  JNE/JNZ label       ; Jump if Not Equal/Not Zero (ZF=0)
```

7.2.2 Jump Unsigned

```
1  JA/JNBE label       ; Jump if Above (CF=0 AND ZF=0)
2  JAE/JNB label       ; Jump if Above or Equal (CF=0)
3  JB/JNAE label       ; Jump if Below (CF=1)
4  JBE/JNA label       ; Jump if Below or Equal (CF=1 OR ZF=1)
```

7.2.3 Jump Signed

```
1  JG/JNLE label       ; Jump if Greater (ZF=0 AND SF=0F)
2  JGE/JNL label       ; Jump if Greater or Equal (SF=0F)
3  JL/JNGE label       ; Jump if Less (SF!=0F)
4  JLE/JNG label       ; Jump if Less or Equal (ZF=1 OR SF!=0F)
```

7.2.4 Jump su singoli flag

```

1 JC label           ; Jump if Carry (CF=1)
2 JNC label          ; Jump if No Carry (CF=0)
3 JS label           ; Jump if Sign (SF=1, negativo)
4 JNS label          ; Jump if No Sign (SF=0, positivo)
5 JO label           ; Jump if Overflow (OF=1)
6 JNO label          ; Jump if No Overflow (OF=0)
7 JP/JPE label       ; Jump if Parity Even (PF=1)
8 JNP/JPO label      ; Jump if Parity Odd (PF=0)

```

7.3 Loop Instructions

7.3.1 LOOP

```

1 LOOP label         ; Decrementa CX, salta se CX!=0

```

Esempio

Loop per 10 iterazioni:

```

1 MOV CX, 10
2 loop_start:
3     ; Corpo del loop
4     DEC BX
5     ; ...
6 LOOP loop_start    ; Ripete finche' CX!=0

```

7.3.2 LOOPE/LOOPZ

```

1 LOOPE label        ; Loop while Equal (decrementa CX, salta se
                      ZF=1 AND CX!=0)

```

7.3.3 LOOPNE/LOOPNZ

```

1 LOOPNE label       ; Loop while Not Equal (decrementa CX, salta
                      se ZF=0 AND CX!=0)

```

7.3.4 JCXZ

```

1 JCXZ label         ; Jump if CX=0 (senza modificare CX)

```

7.4 Procedure

7.4.1 CALL

```
1 CALL procedura      ; Chiama procedura (PUSH IP, JMP)
2 CALL NEAR proc       ; Chiamata near
3 CALL FAR proc        ; Chiamata far (PUSH CS, PUSH IP)
```

7.4.2 RET

```
1 RET                  ; Ritorna (POP IP)
2 RET n                ; Ritorna e rimuove n byte dallo stack
3 RETF                 ; Return far (POP IP, POP CS)
```

Esempio

Procedura semplice:

```
1 somma PROC
2     ADD AX, BX      ; AX = AX + BX
3     RET
4 somma ENDP
5
6 ; Chiamata:
7 MOV AX, 10
8 MOV BX, 20
9 CALL somma          ; AX = 30
```

7.5 Interrupt

7.5.1 INT

```
1 INT numero          ; Invoca interrupt software
```

Esempio

DOS interrupt 21h (servizi DOS):

```
1 MOV AH, 9           ; Funzione 9: stampa stringa
2 MOV DX, OFFSET msg
3 INT 21h
4
5 MOV AH, 4Ch          ; Funzione 4Ch: termina programma
6 INT 21h
```

7.5.2 IRET

```
1 IRET ; Return from interrupt (POP IP, POP CS, POP
    FLAGS)
```

7.6 Tabella Jump Condizionati

Istruzione	Condizione	Uso
JE/JZ	ZF=1	Uguale/Zero
JNE/JNZ	ZF=0	Diverso/Non zero
JG/JNLE	ZF=0 AND SF=OF	Greater (signed)
JGE/JNL	SF=OF	Greater or Equal (signed)
JL/JNGE	SF≠OF	Less (signed)
JLE/JNG	ZF=1 OR SF≠OF	Less or Equal (signed)
JA/JNBE	CF=0 AND ZF=0	Above (unsigned)
JAЕ/JNB	CF=0	Above or Equal (unsigned)
JB/JNAE	CF=1	Below (unsigned)
JBE/JNA	CF=1 OR ZF=1	Below or Equal (unsigned)

Tabella 7.1: Jump condizionati principali

7.7 Esercizi

Esercizio 7.1

Scrivere un loop per sommare i numeri da 1 a 100.

Esercizio 7.2

Implementare una procedura che calcola il massimo tra AX e BX (risultato in AX).

Esercizio 7.3

Spiegare la differenza tra JG e JA. Quando si usa ciascuno?

Esercizio 7.4

Scrivere codice per cercare un byte in un array di 10 elementi.

Esercizio 7.5

Cosa succede se si chiama una procedura senza RET?

Parte III

Tecniche di Programmazione

Capitolo 8

Procedure e Gestione dello Stack

8.1 Lo Stack dell'8086

Lo stack è un'area di memoria LIFO (Last In, First Out) gestita dai registri SS:SP.

8.1.1 Operazioni fondamentali

1	PUSH AX	; $SP = SP - 2, [SS:SP] = AX$
2	POP BX	; $BX = [SS:SP], SP = SP + 2$

8.2 Chiamate a Procedura

8.2.1 Convenzioni di chiamata

Caller (chiamante):

1. Salva registri da preservare
2. Passa parametri (stack o registri)
3. Esegue CALL
4. Recupera risultato
5. Ripristina registri

Callee (procedura chiamata):

1. Salva registri usati
2. Esegue operazioni
3. Ripristina registri
4. Esegue RET

8.2.2 Passaggio parametri via stack

```

1  ; Procedura: somma(a, b) -> risultato in AX
2  somma PROC
3      PUSH BP
4      MOV BP, SP          ; Stack frame
5
6      MOV AX, [BP+4]      ; Parametro a
7      ADD AX, [BP+6]      ; Parametro b
8
9      POP BP
10     RET 4               ; Rimuove 2 parametri (4 byte)
11 somma ENDP
12
13 ; Chiamata:
14 PUSH 20                 ; Parametro b
15 PUSH 10                 ; Parametro a
16 CALL somma              ; AX = 30

```

8.2.3 Stack frame layout

[BP+6] Parametro b
 [BP+4] Parametro a
 [BP+2] Return address (IP)
 [BP+0] Old BP <-- BP
 [BP-2] Variabile locale 1
 [BP-4] Variabile locale 2 <-- SP

8.3 Variabili Locali

```

1  myproc PROC
2      PUSH BP
3      MOV BP, SP
4      SUB SP, 4           ; Alloca 4 byte per variabili locali
5
6      MOV word ptr [BP-2], 100 ; Variabile locale 1
7      MOV word ptr [BP-4], 200 ; Variabile locale 2
8
9      MOV SP, BP          ; Dealloca variabili
10     POP BP
11     RET
12 myproc ENDP

```

8.4 Ricorsione

Esempio

Fattoriale ricorsivo:

```

1  ; Fattoriale(n) -> AX
2  factorial PROC
3      PUSH BP
4      MOV BP, SP
5
6      MOV AX, [BP+4]      ; n
7      CMP AX, 1
8      JLE base_case
9
10     DEC AX
11     PUSH AX
12     CALL factorial      ; Chiamata ricorsiva
13     ADD SP, 2
14
15     MUL word ptr [BP+4]  ; AX = risultato * n
16     JMP done
17
18 base_case:
19     MOV AX, 1
20
21 done:
22     POP BP
23     RET 2
24 factorial ENDP

```

8.5 Esercizi

Esercizio 8.1

Scrivere una procedura che calcola x^n (potenza).

Esercizio 8.2

Implementare Fibonacci ricorsivo.

Esercizio 8.3

Spiegare perché BP è preferibile a SP per accedere a parametri e variabili locali.

Esercizio 8.4

Disegnare lo stack frame per una procedura con 3 parametri e 2 variabili locali.

Esercizio 8.5

Cosa succede se si dimentica POP BP prima di RET?

Capitolo 9

Operazioni su Stringhe

9.1 Introduzione

Le istruzioni su stringhe dell'8086 operano su sequenze di byte o word in memoria, usando SI (source) e DI (destination).

9.2 Registri per Stringhe

- **SI**: Source Index (DS:SI per default)
- **DI**: Destination Index (ES:DI sempre)
- **CX**: Contatore per prefissi REP
- **DF**: Direction Flag (0=incrementa, 1=decrementa)

9.2.1 CLD e STD

```
1 CLD                ; Clear Direction Flag (incremento)
2 STD                ; Set Direction Flag (decremento)
```

9.3 Istruzioni Base

9.3.1 MOVSb/MOVSr — Move String

```
1 MOVSb              ; byte[ES:DI] = byte[DS:SI], aggiorna SI e
  DI
2 MOVSr              ; word[ES:DI] = word[DS:SI], aggiorna SI e
  DI di 2
```

9.3.2 LODSB/LODSW — Load String

```

1 LODSB          ; AL = byte[DS:SI], aggiorna SI
2 LODSW          ; AX = word[DS:SI], aggiorna SI di 2

```

9.3.3 STOSB/STOSW — Store String

```

1 STOSB          ; byte[ES:DI] = AL, aggiorna DI
2 STOSW          ; word[ES:DI] = AX, aggiorna DI di 2

```

9.3.4 CMPSB/CMPSW — Compare String

```

1 CMPSB          ; Confronta byte[DS:SI] con byte[ES:DI],
    imposta flag
2 CMPSW          ; Confronta word

```

9.3.5 SCASB/SCASW — Scan String

```

1 SCASB          ; Confronta AL con byte[ES:DI], aggiorna DI
2 SCASW          ; Confronta AX con word[ES:DI]

```

9.4 Prefissi REP

9.4.1 REP

```

1 REP MOVSB      ; Ripete MOVSB per CX volte
2 REP STOSB      ; Ripete STOSB per CX volte

```

9.4.2 REPE/REPZ

```

1 REPE CMPSB     ; Ripete finche' uguale e CX!=0

```

9.4.3 REPNE/REPNZ

```

1 REPNE SCASB    ; Ripete finche' diverso e CX!=0

```

9.5 Esempi Pratici

Esempio

Copiare 100 byte da src a dest:

```
1  .DATA
2  src DB 100 DUP(?)
3  dest DB 100 DUP(?)
4
5  .CODE
6  CLD
7  MOV SI, OFFSET src
8  MOV DI, OFFSET dest
9  MOV CX, 100
10 REP MOVSB
```

Esempio

Riempire buffer con zero:

```
1  CLD
2  MOV DI, OFFSET buffer
3  MOV CX, 256
4  MOV AL, 0
5  REP STOSB           ; Azzera 256 byte
```

Esempio

Lunghezza stringa (cerca byte 0):

```
1  strlen PROC
2      PUSH DI
3      MOV DI, OFFSET stringa
4      MOV AL, 0
5      MOV CX, 0FFFFh      ; Max length
6      CLD
7      REPNE SCASB         ; Cerca byte 0
8      MOV AX, 0FFFFh
9      SUB AX, CX
10     DEC AX              ; AX = lunghezza
11     POP DI
12     RET
13  strlen ENDP
```

Esempio

Confrontare due stringhe:

```
1  strcmp PROC
2      CLD
3      MOV SI, OFFSET str1
4      MOV DI, OFFSET str2
5      MOV CX, 10          ; Max 10 caratteri
6      REPE CMPSB          ; Confronta finche' uguali
7      JE equal
8      ; Diverse
9      RET
10
11 equal:
12     ; Uguali
13     RET
14 strcmp ENDP
```

9.6 Esercizi

Esercizio 9.1

Scrivere codice per invertire una stringa di 20 caratteri.

Esercizio 9.2

Implementare una funzione che conta le occorrenze di un carattere in una stringa.

Esercizio 9.3

Spiegare la differenza tra MOVSB e LODSB.

Esercizio 9.4

Convertire una stringa in maiuscolo usando LODSB e STOSB.

Esercizio 9.5

Implementare memcpy(dest, src, n) usando REP MOVSB.

Capitolo 10

Sistema di Interrupt

10.1 Introduzione agli Interrupt

Gli interrupt permettono al processore di rispondere a eventi esterni o richiedere servizi di sistema.

10.2 Tipi di Interrupt

10.2.1 Hardware Interrupts

Generati da dispositivi esterni (tastiera, timer, disco).

10.2.2 Software Interrupts

Invocati da programma con istruzione INT.

10.2.3 Exceptions

Generati dal processore (divisione per zero, overflow).

10.3 Interrupt Vector Table (IVT)

Tabella a indirizzo 0000:0000 contenente 256 vettori di 4 byte (CS:IP).

```
1 ; Vettore interrupt n a indirizzo n*4
2 ; Offset = n * 4
3 ; Segmento = n * 4 + 2
```

10.4 Interrupt DOS (INT 21h)

10.4.1 Funzione 01h: Input carattere con echo

```
1 MOV AH, 01h
2 INT 21h                ; AL = carattere letto
```

10.4.2 Funzione 02h: Output carattere

```
1 MOV AH, 02h
2 MOV DL, 'A'
3 INT 21h                ; Stampa 'A'
```

10.4.3 Funzione 09h: Stampa stringa

```
1 .DATA
2 msg DB 'Hello World$'
3
4 .CODE
5 MOV AH, 09h
6 MOV DX, OFFSET msg
7 INT 21h
```

10.4.4 Funzione 4Ch: Termina programma

```
1 MOV AH, 4Ch
2 MOV AL, 0              ; Exit code
3 INT 21h
```

10.5 Interrupt BIOS

10.5.1 INT 10h: Video Services

```
1 ; Funzione 0Eh: Teletype output
2 MOV AH, 0Eh
3 MOV AL, 'X'
4 MOV BH, 0              ; Page number
5 INT 10h
6
7 ; Funzione 00h: Set video mode
8 MOV AH, 00h
9 MOV AL, 03h            ; Modo testo 80x25
10 INT 10h
```

10.5.2 INT 16h: Keyboard Services

```

1 ; Funzione 00h: Legge tasto
2 MOV AH, 00h
3 INT 16h                ; AL = ASCII, AH = scan code

```

10.5.3 INT 13h: Disk Services

```

1 ; Funzione 02h: Legge settori
2 MOV AH, 02h
3 MOV AL, 1                ; Numero settori
4 MOV CH, 0                ; Cilindro
5 MOV CL, 1                ; Settore
6 MOV DH, 0                ; Testina
7 MOV DL, 00h              ; Drive (0=A:, 80h=HD)
8 MOV BX, OFFSET buffer
9 INT 13h

```

10.6 Gestione Custom Interrupt

```

1 ; Installare handler personalizzato per INT 60h
2 install_handler PROC
3     CLI                    ; Disabilita interrupt
4     PUSH DS
5
6     ; Calcola indirizzo IVT: 60h * 4 = 180h
7     XOR AX, AX
8     MOV DS, AX
9     MOV BX, 180h
10
11    ; Salva vecchio handler
12    MOV AX, [BX]
13    MOV old_offset, AX
14    MOV AX, [BX+2]
15    MOV old_segment, AX
16
17    ; Installa nuovo handler
18    MOV word ptr [BX], OFFSET my_handler
19    MOV [BX+2], CS
20
21    POP DS
22    STI                    ; Riabilita interrupt
23    RET
24 install_handler ENDP
25
26 my_handler PROC

```

```
27     ; Codice handler  
28     IRET  
29 my_handler ENDP
```

10.7 Esercizi

Esercizio 10.1

Scrivere un programma che legge un carattere e lo stampa 10 volte.

Esercizio 10.2

Implementare una funzione per stampare un numero decimale usando INT 21h.

Esercizio 10.3

Spiegare la differenza tra INT, CALL e JMP.

Esercizio 10.4

Scrivere codice per cambiare il colore del testo usando INT 10h.

Esercizio 10.5

Implementare un semplice handler per INT 60h che incrementa un contatore.

Capitolo 11

I/O e Interfacciamento Hardware

11.1 Introduzione

L'8086 comunica con periferiche tramite porte I/O a 16 bit (65536 porte possibili).

11.2 Istruzioni I/O

11.2.1 IN e OUT

```
1 ; I/O fisso (porta 0-255)
2 IN AL, porta
3 OUT porta, AL
4
5 ; I/O variabile (porta in DX)
6 MOV DX, porta
7 IN AL, DX
8 OUT DX, AL
```

]Programmable Interrupt Controller (8259)

11.2.2 Porte del PIC

- **20h**: Registro comando (Master PIC)
- **21h**: Registro maschera interrupt (IMR)
- **A0h**: Slave PIC comando
- **A1h**: Slave PIC maschera

```
1 ; Disabilitare interrupt tastiera (IRQ1)
2 IN AL, 21h ; Legge IMR
3 OR AL, 02h ; Setta bit 1
4 OUT 21h, AL ; Scrive IMR
```

```
5
6 ; Riabilitare
7 IN AL, 21h
8 AND AL, 0FDh ; Azzera bit 1
9 OUT 21h, AL
```

11.3 Programmable Interval Timer (8253/8254)

11.3.1 Porte del PIT

- **40h**: Canale 0 (system timer)
- **41h**: Canale 1
- **42h**: Canale 2 (speaker)
- **43h**: Control word

Esempio

Generare beep con speaker:

```
1 ; Frequenza = 1193180 / divisore
2 ; Per 1000 Hz: divisore = 1193
3
4 MOV AL, 0B6h ; Control word
5 OUT 43h, AL
6
7 MOV AX, 1193
8 OUT 42h, AL ; Byte basso
9 MOV AL, AH
10 OUT 42h, AL ; Byte alto
11
12 ; Abilita speaker
13 IN AL, 61h
14 OR AL, 03h
15 OUT 61h, AL
16
17 ; Attendi...
18
19 ; Disabilita speaker
20 IN AL, 61h
21 AND AL, 0FCh
22 OUT 61h, AL
```

11.4 Porta Parallela (LPT)

11.4.1 Porte LPT1

- **378h**: Data port (8 bit output)
- **379h**: Status port (5 bit input)
- **37Ah**: Control port

```
1 ; Invia byte alla stampante
2 MOV DX, 378h
3 MOV AL, 'A'
4 OUT DX, AL
5
6 ; Strobe
7 MOV DX, 37Ah
8 MOV AL, 0Dh
9 OUT DX, AL
10 MOV AL, 0Ch
11 OUT DX, AL
```

11.5 Porta Seriale (COM)

11.5.1 UART 8250/16550

Porte COM1 (base 3F8h):

- **3F8h**: Data register (RBR/THR)
- **3F9h**: Interrupt Enable Register (IER)
- **3FAh**: Interrupt ID Register (IIR)
- **3FBh**: Line Control Register (LCR)
- **3FCh**: Modem Control Register (MCR)
- **3FDh**: Line Status Register (LSR)
- **3FEh**: Modem Status Register (MSR)

Esempio

Inizializzare COM1 a 9600 baud, 8N1:

```

1  ; Divisore per 9600 baud: 115200 / 9600 = 12
2
3  MOV DX, 3FBh      ; LCR
4  MOV AL, 80h       ; DLAB = 1
5  OUT DX, AL
6
7  MOV DX, 3F8h      ; DLL
8  MOV AL, 12
9  OUT DX, AL
10
11 MOV DX, 3F9h      ; DLM
12 MOV AL, 0
13 OUT DX, AL
14
15 MOV DX, 3FBh      ; LCR
16 MOV AL, 03h       ; 8N1, DLAB=0
17 OUT DX, AL

```

Esempio

Trasmettere byte su COM1:

```

1  send_byte PROC
2      ; AL = byte da inviare
3      PUSH DX
4      PUSH AX
5
6  wait_ready:
7      MOV DX, 3FDh   ; LSR
8      IN AL, DX
9      TEST AL, 20h   ; THRE bit
10     JZ wait_ready
11
12     POP AX
13     MOV DX, 3F8h    ; THR
14     OUT DX, AL
15
16     POP DX
17     RET
18 send_byte ENDP

```

11.6 Tastiera (8042)

11.6.1 Porte tastiera

- **60h**: Data port
- **64h**: Status/Command port

```
1 ; Leggi scan code
2 IN AL, 60h
3
4 ; Verifica buffer pieno
5 IN AL, 64h
6 TEST AL, 01h          ; Output buffer full
7 JZ no_key
8 IN AL, 60h          ; Leggi dato
9
10 no_key:
```

11.7 VGA (Video Graphics Array)

11.7.1 Accesso diretto a memoria video

Modo testo 80x25: buffer a B800:0000

```
1 ; Scrivere 'A' rosso su sfondo nero in posizione (0,0)
2 MOV AX, 0B800h
3 MOV ES, AX
4 MOV DI, 0
5 MOV AH, 04h          ; Attributo: rosso
6 MOV AL, 'A'
7 MOV ES:[DI], AX
```

11.8 Esercizi

Esercizio 11.1

Scrivere codice per generare un beep di 2 secondi a 440 Hz (nota LA).

Esercizio 11.2

Implementare una funzione per leggere un carattere da COM1.

Esercizio 11.3

Scrivere una stringa direttamente in memoria video VGA.

Esercizio 11.4

Spiegare la differenza tra I/O mapped e memory mapped.

Esercizio 11.5

Implementare un delay preciso usando il PIT.

Parte IV

Progetti e Applicazioni

Capitolo 12

Progetti Applicativi

12.1 Progetto 1: Calcolatrice a 16 bit

Implementare una calcolatrice che supporta +, -, *, / su numeri interi.

```
1  .MODEL  SMALL
2  .STACK  100h
3
4  .DATA
5  msg1 DB 'Inserisci primo numero: $'
6  msg2 DB 'Inserisci operatore (+,-,*,/): $'
7  msg3 DB 'Inserisci secondo numero: $'
8  result_msg DB 'Risultato: $'
9  num1 DW ?
10 num2 DW ?
11 operator DB ?
12
13 .CODE
14 main PROC
15     MOV AX, @DATA
16     MOV DS, AX
17
18     ; Input primo numero
19     MOV AH, 09h
20     MOV DX, OFFSET msg1
21     INT 21h
22     CALL read_number
23     MOV num1, AX
24
25     ; Input operatore
26     MOV AH, 09h
27     MOV DX, OFFSET msg2
28     INT 21h
29     MOV AH, 01h
30     INT 21h
31     MOV operator, AL
32
```

```

33      ; Input secondo numero
34      MOV AH, 09h
35      MOV DX, OFFSET msg3
36      INT 21h
37      CALL read_number
38      MOV num2, AX
39
40      ; Esegui operazione
41      MOV AL, operator
42      CMP AL, '+'
43      JE do_add
44      CMP AL, '-'
45      JE do_sub
46      CMP AL, '*'
47      JE do_mul
48      CMP AL, '/'
49      JE do_div
50
51 do_add:
52     MOV AX, num1
53     ADD AX, num2
54     JMP display
55
56 do_sub:
57     MOV AX, num1
58     SUB AX, num2
59     JMP display
60
61 do_mul:
62     MOV AX, num1
63     MUL num2
64     JMP display
65
66 do_div:
67     MOV AX, num1
68     XOR DX, DX
69     DIV num2
70     JMP display
71
72 display:
73     CALL print_number
74
75     MOV AH, 4Ch
76     INT 21h
77 main ENDP
78
79 ; Procedura per leggere numero decimale
80 read_number PROC
81     ; ... implementazione ...

```

```

82     RET
83 read_number ENDP
84
85 ; Procedura per stampare numero
86 print_number PROC
87     ; ... implementazione ...
88     RET
89 print_number ENDP
90
91 END main

```

12.2 Progetto 2: Ordinamento Array

Bubble sort su array di 10 numeri.

```

1  .DATA
2  array DW 64, 34, 25, 12, 22, 11, 90, 88, 45, 50
3  size DW 10
4
5  .CODE
6  bubble_sort PROC
7      MOV CX, size
8      DEC CX                ; n-1 passate
9
10 outer_loop:
11     PUSH CX
12     MOV SI, 0
13     MOV CX, size
14     DEC CX
15
16 inner_loop:
17     MOV AX, array[SI]
18     CMP AX, array[SI+2]
19     JLE no_swap
20
21     ; Scambia
22     XCHG AX, array[SI+2]
23     MOV array[SI], AX
24
25 no_swap:
26     ADD SI, 2
27     LOOP inner_loop
28
29     POP CX
30     LOOP outer_loop
31
32     RET
33 bubble_sort ENDP

```

12.3 Progetto 3: Editor di Testo Semplice

Editor di testo minimale con buffer di 256 caratteri.

12.4 Progetto 4: Gioco Snake

Snake in modalità testo 80x25.

12.5 Progetto 5: Bootloader

Bootloader minimale che stampa un messaggio.

```
1  [ORG 0x7C00]
2  [BITS 16]
3
4  start:
5      MOV AX, 0x07C0
6      MOV DS, AX
7      MOV ES, AX
8
9      MOV AH, 0x0E      ; Teletype
10     MOV SI, msg
11
12  print_loop:
13     LODSB
14     OR AL, AL
15     JZ done
16     INT 0x10
17     JMP print_loop
18
19  done:
20     CLI
21     HLT
22
23  msg DB 'Bootloader avviato!', 13, 10, 0
24
25  TIMES 510-($-$$) DB 0
26  DW 0xAA55      ; Boot signature
```

12.6 Esercizi

Esercizio 12.1

Completare il progetto calcolatrice implementando `read_number` e `print_number`.

Esercizio 12.2

Modificare bubble sort per ordinamento decrescente.

Esercizio 12.3

Implementare Quick Sort ricorsivo.

Esercizio 12.4

Scrivere un bootloader che legge un settore dal disco.

Esercizio 12.5

Creare un semplice shell DOS-like con comandi DIR, TYPE, EXIT.

Capitolo 13

Esercizi Progressivi

13.1 Livello Base

Esercizio B.1 — Hello World

Scrivere un programma che stampa "Hello, Assembly!" usando INT 21h funzione 09h.

Esercizio B.2 — Somma Due Numeri

Leggere due numeri a 16 bit, sommarli e stampare il risultato.

Esercizio B.3 — Pari o Dispari

Leggere un numero e determinare se è pari o dispari (testare bit 0).

Esercizio B.4 — Massimo di Tre

Dati tre numeri, trovare il massimo.

Esercizio B.5 — Somma Array

Sommare tutti gli elementi di un array di 10 word.

Esercizio B.6 — Copia Stringa

Copiare una stringa usando MOVSB.

13.2 Livello Intermedio

Esercizio I.1 — Fattoriale

Calcolare il fattoriale di n (versione iterativa e ricorsiva).

Esercizio I.2 — Fibonacci

Generare i primi n numeri di Fibonacci.

Esercizio I.3 — Numero Primo

Verificare se un numero è primo.

Esercizio I.4 — Ricerca Binaria

Implementare binary search su array ordinato.

Esercizio I.5 — Conversione Decimale-Esadecimale

Convertire numero decimale in esadecimale e stamparlo.

Esercizio I.6 — Palindromo

Verificare se una stringa è palindroma.

Esercizio I.7 — Ordinamento Insertion Sort

Implementare insertion sort su array.

13.3 Livello Avanzato

Esercizio A.1 — Torre di Hanoi

Risolvere Torre di Hanoi con ricorsione.

Esercizio A.2 — Moltiplicazione Matrici

Moltiplicare due matrici 3x3.

Esercizio A.3 — Parser Espressioni

Valutare espressioni aritmetiche con precedenza operatori.

Esercizio A.4 — Allocatore Memoria

Implementare malloc/free semplificato.

Esercizio A.5 — Compressione RLE

Implementare compressione Run-Length Encoding.

Esercizio A.6 — File System FAT12

Leggere e parsare directory FAT12 da floppy disk.

Esercizio A.7 — Debugger Minimale

Implementare debugger con breakpoint e single-step.

13.4 Progetti Integrati

Esercizio P.1 — Sistema Operativo Minimale

Creare un micro-OS con:

- Bootloader
- Gestore interrupt
- Shell minimale
- Driver VGA e tastiera

Esercizio P.2 — Emulatore 8086

Scrivere emulatore software dell'8086 in C/Python.

Esercizio P.3 — Crittografia

Implementare cifratura/decifratura Caesar, XOR, ROT13.

Esercizio P.4 — Gioco Tetris

Implementare Tetris in modalità testo.

Esercizio P.5 — Comunicazione Seriale

Implementare protocollo seriale per comunicazione PC-PC.

13.5 Sfide Avanzate

Esercizio S.1 — Code Golf

Scrivere il programma più corto possibile per:

1. Stampare "Hello World"
2. Ordinare array
3. Calcolare Fibonacci

Esercizio S.2 — Ottimizzazione

Ottimizzare i seguenti task per velocità massima:

1. Copiare 64KB di memoria
2. Cercare byte in array 1MB
3. Calcolare checksum CRC16

Esercizio S.3 — Reverse Engineering

Analizzare e decompilare un binario .COM fornito.

Esercizio S.4 — Real Mode OS

Scrivere un OS completo in real mode con:

- Multitasking cooperativo
- Driver disco e file system
- Interfaccia grafica VGA

13.6 Riepilogo Competenze

Al completamento di tutti gli esercizi, lo studente avrà acquisito:

- Padronanza completa del set di istruzioni 8086
- Capacità di debugging e ottimizzazione
- Comprensione architettura low-level
- Esperienza con hardware e periferiche
- Basi per reverse engineering e sicurezza

Appendice A

Soluzioni agli Esercizi

Prefazione

Questa appendice contiene le soluzioni dettagliate di una selezione degli esercizi proposti nel manuale. Si consiglia di tentare autonomamente la risoluzione prima di consultare le soluzioni.

A.1 Soluzioni Capitolo 1 — Architettura

A.1.1 Esercizio 1.1

Calcolare l'indirizzo fisico:

a) CS:IP = 2000h:1000h

$$\begin{aligned}\text{Fisico} &= 2000h \times 10h + 1000h \\ &= 20000h + 1000h \\ &= 21000h\end{aligned}$$

b) DS:SI = FFFFh:0010h

$$\begin{aligned}\text{Fisico} &= FFFFh \times 10h + 0010h \\ &= FFFF0h + 0010h \\ &= 100000h\end{aligned}$$

Nota: supera 1 MB! Nell'8086 reale, diventa 00000h (wrap-around).

c) SS:SP = 3000h:FFFEh

$$\begin{aligned}\text{Fisico} &= 3000h \times 10h + FFFEh \\ &= 30000h + FFFEh \\ &= 3FFFEh\end{aligned}$$

A.1.2 Esercizio 1.2

Lo spazio indirizzabile è 1 MB = 1.048.576 byte = 100000h.

Un segmento può partire ogni 16 byte (granularità del paragraph).

Numero segmenti possibili: $100000h/10h = 10000h = 65.536$ segmenti.

A.1.3 Esercizio 1.5

ADD AX, BX modifica i seguenti flag:

- **CF**: 1 se riporto da bit 15
- **ZF**: 1 se risultato è zero
- **SF**: 1 se bit 15 del risultato è 1 (negativo)
- **OF**: 1 se overflow con segno
- **PF**: 1 se byte basso ha parità pari
- **AF**: 1 se riporto da bit 3 (BCD)

A.2 Soluzioni Capitolo 2 — Registri

A.2.1 Esercizio 2.1

Dato AX = 1234h:

a) Azzerare solo AL:

```
1 AND AL, 0 ; oppure: XOR AL, AL oppure: MOV AL, 0
```

b) Azzerare solo AH:

```
1 AND AH, 0
```

c) Scambiare AH e AL:

```
1 XCHG AH, AL ; AX diventa 3412h
```

A.2.2 Esercizio 2.2

MOV AX, BX: Copia il *valore* di BX in AX.

MOV AX, [BX]: Copia il *contenuto della memoria* puntata da DS:BX in AX.

A.2.3 Esercizio 2.3

Non è possibile MOV DS, 1000h perché i registri segmento non accettano valori immediati.

Soluzione:

```
1 MOV AX, 1000h
2 MOV DS, AX
```

A.2.4 Esercizio 2.4

Indirizzo iniziale:

$$\begin{aligned}\text{Fisico} &= 2000h \times 10h + FFFEh \\ &= 20000h + FFFEh \\ &= 2FFFEh\end{aligned}$$

Dopo PUSH AX:

- $SP = FFFEh - 2 = FFFCh$
- Nuovo indirizzo fisico: $2000h \times 10h + FFFCh = 2FFCh$

A.3 Soluzioni Capitolo 4 — Trasferimento Dati

A.3.1 Esercizio 4.1

Scambiare AX e BX senza XCHG:

```
1  PUSH  AX
2  PUSH  BX
3  POP   AX
4  POP   BX
```

Oppure con registro temporaneo:

```
1  MOV  CX,  AX
2  MOV  AX,  BX
3  MOV  BX,  CX
```

A.3.2 Esercizio 4.2

Stack iniziale: $SP = FFFEh$

```
1  PUSH  AX      ; SP = FFFCh
2  PUSH  BX      ; SP = FFFAh
3  POP   CX      ; SP = FFFCh
4  PUSH  DX      ; SP = FFFAh
```

SP finale: FFFAh

A.4 Soluzioni Capitolo 5 — Aritmetiche

A.4.1 Esercizio 5.1

Calcolare 123×45 :

```
1  MOV  AX, 123
2  MOV  BX, 45
3  MUL  BX      ; AX = 5535 (159Fh)
```

A.4.2 Esercizio 5.2

Dividere 1000 per 7:

```

1 MOV AX, 1000
2 MOV BL, 7
3 XOR AH, AH           ; Azzera AH per divisione a 8 bit
4 DIV BL               ; AL = 142 (quoziente), AH = 6 (resto)

```

Oppure a 16 bit:

```

1 MOV AX, 1000
2 XOR DX, DX
3 MOV BX, 7
4 DIV BX               ; AX = 142, DX = 6

```

A.4.3 Esercizio 5.3

Somma a 32 bit: 12345678h + ABCDEF01h

```

1 ; Numero 1: DX:AX = 1234:5678h
2 MOV DX, 1234h
3 MOV AX, 5678h
4
5 ; Numero 2: CX:BX = ABCD:EF01h
6 MOV CX, 0ABCDh
7 MOV BX, 0EF01h
8
9 ; Somma
10 ADD AX, BX           ; Somma parte bassa
11 ADC DX, CX           ; Somma parte alta + carry
12
13 ; Risultato in DX:AX = BE02:4579h

```

A.4.4 Esercizio 5.5

Valore assoluto di AX:

```

1 CMP AX, 0
2 JGE positive         ; Se già positivo, salta
3 NEG AX               ; Altrimenti nega
4
5 positive:
6 ; AX ora contiene |AX|

```

A.5 Soluzioni Capitolo 7 — Controllo Flusso

A.5.1 Esercizio 7.1

Somma $1 + 2 + \dots + 100$:

```

1  MOV AX, 0           ; Somma
2  MOV CX, 100         ; Contatore
3
4  sum_loop:
5      ADD AX, CX
6      LOOP sum_loop   ; CX = CX - 1, ripeti se CX != 0
7
8  ; AX = 5050

```

A.5.2 Esercizio 7.2

Massimo tra AX e BX:

```

1  max PROC
2      CMP AX, BX
3      JGE ax_is_max   ; Se AX >= BX
4      MOV AX, BX      ; Altrimenti AX = BX
5
6  ax_is_max:
7      RET
8  max ENDP

```

A.5.3 Esercizio 7.3

JG (Jump if Greater) è per confronti *signed*. Usa SF e OF: salta se ZF = 0 AND SF = OF.

JA (Jump if Above) è per confronti *unsigned*. Usa CF e ZF: salta se CF = 0 AND ZF = 0.

Esempio:

```

1  MOV AL, 0FFh        ; -1 signed, 255 unsigned
2  CMP AL, 1
3
4  JG greater_signed    ; Non salta (FF < 1 in signed)
5  JA above_unsigned    ; Salta (FF > 1 in unsigned)

```

A.6 Soluzioni Capitolo 9 — Stringhe

A.6.1 Esercizio 9.1

Invertire stringa di 20 caratteri:

```

1  .DATA
2  str DB 'ABCDEFGHIJKLMNQRST'
3
4  .CODE
5  reverse PROC

```

```

6      MOV SI, 0
7      MOV DI, 19
8
9  reverse_loop:
10     CMP SI, DI
11     JGE done
12
13     MOV AL, str[SI]
14     MOV BL, str[DI]
15     MOV str[SI], BL
16     MOV str[DI], AL
17
18     INC SI
19     DEC DI
20     JMP reverse_loop
21
22 done:
23     RET
24 reverse ENDP

```

A.6.2 Esercizio 9.4

Convertire in maiuscolo:

```

1  .DATA
2  str DB 'hello world', 0
3
4  .CODE
5  to_upper PROC
6      MOV SI, OFFSET str
7      MOV DI, OFFSET str
8      CLD
9
10 loop_char:
11     LODSB                                ; AL = [SI++]
12     CMP AL, 0
13     JE done
14
15     CMP AL, 'a'
16     JB not_lowercase
17     CMP AL, 'z'
18     JA not_lowercase
19
20     SUB AL, 32                            ; 'a' - 'A' = 32
21
22 not_lowercase:
23     STOSB                                ; [DI++] = AL
24     JMP loop_char
25

```

```
26 done :  
27     RET  
28 to_upper ENDP
```

A.7 Note Finali

Le soluzioni presentate sono una tra le molte possibili. L'Assembly permette approcci diversi allo stesso problema. Si incoraggia lo studente a:

- Confrontare le proprie soluzioni con quelle proposte
- Analizzare l'efficienza (numero istruzioni, clock cycles)
- Testare il codice con diversi input edge-case
- Documentare il codice con commenti chiari

Appendice B

Quick Reference

B.1 Registri 8086

B.1.1 General Purpose (16 bit)

AX Accumulatore (AH + AL)
BX Base (BH + BL)
CX Contatore (CH + CL)
DX Dati (DH + DL)

B.1.2 Puntatore e Indice

SP Stack Pointer
BP Base Pointer
SI Source Index
DI Destination Index
IP Instruction Pointer

B.1.3 Segmento

CS Code Segment
DS Data Segment
SS Stack Segment
ES Extra Segment

B.2 Flag Register

Bit	Nome	Descrizione
0	CF	Carry Flag
2	PF	Parity Flag
4	AF	Auxiliary Carry
6	ZF	Zero Flag
7	SF	Sign Flag
8	TF	Trap Flag
9	IF	Interrupt Flag
10	DF	Direction Flag
11	OF	Overflow Flag

B.3 Modalità Indirizzamento

1	MOV AX, 1234h	; Immediate
2	MOV AX, BX	; Register
3	MOV AX, [1000h]	; Direct
4	MOV AX, [BX]	; Register Indirect
5	MOV AX, [BX+4]	; Based
6	MOV AX, [SI+4]	; Indexed
7	MOV AX, [BX+SI+4]	; Based Indexed

B.4 Istruzioni per Categoria

B.4.1 Trasferimento Dati

1	MOV dest, src	; Move
2	XCHG op1, op2	; Exchange
3	LEA reg, mem	; Load Effective Address
4	PUSH operando	; Push to stack
5	POP operando	; Pop from stack
6	IN AL/AX, porta	; Input from port
7	OUT porta, AL/AX	; Output to port
8	XLAT	; Translate

B.4.2 Aritmetiche

1	ADD dest, src	; Addition
2	ADC dest, src	; Add with carry
3	SUB dest, src	; Subtraction
4	SBB dest, src	; Subtract with borrow
5	INC operando	; Increment
6	DEC operando	; Decrement
7	MUL operando	; Unsigned multiply

```

8  IMUL operando          ; Signed multiply
9  DIV operando           ; Unsigned divide
10 IDIV operando          ; Signed divide
11 NEG operando           ; Negate
12 CMP op1, op2           ; Compare

```

B.4.3 Logiche

```

1  AND dest, src          ; Logical AND
2  OR dest, src           ; Logical OR
3  XOR dest, src          ; Logical XOR
4  NOT operando           ; Logical NOT
5  TEST op1, op2          ; Test (AND without save)
6  SHL dest, count       ; Shift left
7  SHR dest, count       ; Shift right
8  SAL dest, count       ; Arithmetic shift left
9  SAR dest, count       ; Arithmetic shift right
10 ROL dest, count       ; Rotate left
11 ROR dest, count       ; Rotate right
12 RCL dest, count       ; Rotate through carry left
13 RCR dest, count       ; Rotate through carry right

```

B.4.4 Controllo Flusso

```

1  JMP label              ; Unconditional jump
2  JE/JZ label            ; Jump if equal/zero
3  JNE/JNZ label         ; Jump if not equal/zero
4  JG/JNLE label         ; Jump if greater (signed)
5  JGE/JNL label         ; Jump if >= (signed)
6  JL/JNGE label         ; Jump if less (signed)
7  JLE/JNG label         ; Jump if <= (signed)
8  JA/JNBE label         ; Jump if above (unsigned)
9  JAE/JNB label         ; Jump if >= (unsigned)
10 JB/JNAE label         ; Jump if below (unsigned)
11 JBE/JNA label         ; Jump if <= (unsigned)
12 LOOP label             ; Loop (decrement CX)
13 CALL procedura        ; Call procedure
14 RET                   ; Return
15 INT numero            ; Software interrupt

```

B.4.5 Stringhe

```

1  MOVSX/MOVSX           ; Move string
2  LODSB/LODSW           ; Load string
3  STOSB/STOSW           ; Store string
4  CMPSB/CMPSW           ; Compare string

```

```

5 SCASB/SCASW          ; Scan string
6 REP                  ; Repeat prefix
7 REPE/REPZ            ; Repeat while equal
8 REPNE/REPNZ          ; Repeat while not equal
9 CLD                  ; Clear direction flag
10 STD                 ; Set direction flag

```

B.5 Interrupt DOS (INT 21h)

```

1 MOV AH, 01h
2 INT 21h              ; Input character (AL)
3
4 MOV AH, 02h
5 MOV DL, char
6 INT 21h              ; Output character
7
8 MOV AH, 09h
9 MOV DX, OFFSET msg
10 INT 21h             ; Print string ($-terminated)
11
12 MOV AH, 4Ch
13 INT 21h             ; Exit program

```

B.6 Interrupt BIOS

B.6.1 INT 10h (Video)

```

1 MOV AH, 00h
2 MOV AL, mode
3 INT 10h              ; Set video mode
4
5 MOV AH, 0Eh
6 MOV AL, char
7 INT 10h              ; Teletype output

```

B.6.2 INT 16h (Keyboard)

```

1 MOV AH, 00h
2 INT 16h              ; Read key (AL=ASCII, AH=scan)

```

B.7 Struttura Programma .COM

```

1  ORG 100h                ; DOS PSP = 256 byte
2
3  start:
4      MOV AH, 09h
5      MOV DX, OFFSET msg
6      INT 21h
7
8      MOV AH, 4Ch
9      INT 21h
10
11 msg DB 'Hello$'

```

B.8 Struttura Programma .EXE

```

1  .MODEL SMALL
2  .STACK 100h
3
4  .DATA
5  msg DB 'Hello$'
6
7  .CODE
8  main PROC
9      MOV AX, @DATA
10     MOV DS, AX
11
12     MOV AH, 09h
13     MOV DX, OFFSET msg
14     INT 21h
15
16     MOV AH, 4Ch
17     INT 21h
18 main ENDP
19
20 END main

```

B.9 Calcolo Indirizzo Fisico

$$\text{Fisico} = (\text{Segmento} \times 16) + \text{Offset}$$

B.10 Tabella ASCII (estratto)

Dec	Hex	Char	Descrizione
0	00h	NUL	Null
7	07h	BEL	Bell
8	08h	BS	Backspace
9	09h	TAB	Horizontal Tab
10	0Ah	LF	Line Feed
13	0Dh	CR	Carriage Return
32	20h	SPACE	Spazio
48-57	30h-39h	0-9	Cifre
65-90	41h-5Ah	A-Z	Maiuscole
97-122	61h-7Ah	a-z	Minuscole

B.11 Scan Code Tastiera (estratto)

Scan Code	Tasto
01h	ESC
1Ch	Enter
39h	Spazio
3Bh-44h	F1-F10
48h	Freccia su
50h	Freccia giù
4Bh	Freccia sinistra
4Dh	Freccia destra

B.12 Porte I/O Comuni

Porta	Descrizione
20h-21h	PIC (8259) Master
40h-43h	PIT (8253/8254) Timer
60h, 64h	Tastiera (8042)
3F8h-3FFh	COM1 (UART)
378h-37Fh	LPT1 (Parallela)

Appendice C

Bibliografia e Risorse

C.1 Manuali Ufficiali

1. **Intel 8086/8088 User's Manual**
Intel Corporation, 1981
Manuale ufficiale completo dell'architettura 8086.
<https://www.intel.com/>
2. **The Art of Assembly Language Programming**
Randall Hyde
Guida completa alla programmazione Assembly.
<https://www.plantation-productions.com/Webster/>
3. **PC Assembly Language**
Paul A. Carter
Introduzione moderna all'Assembly x86.
<https://pacman128.github.io/pcasm/>

C.2 Libri di Testo

1. **Assembly Language for x86 Processors**
Kip R. Irvine
Pearson, 7th Edition, 2014
ISBN: 978-0133769401
2. **The 8086/8088 Family: Design, Programming, and Interfacing**
John E. Uffenbeck
Prentice Hall, 2002
ISBN: 978-0130406859
3. **Microprocessor 8086: Architecture, Programming and Interfacing**
Mathur Sunil
PHI Learning, 2010
ISBN: 978-8120340688

C.3 Risorse Online

C.3.1 Tutorial e Guide

- **Tutorialspoint Assembly Programming**
https://www.tutorialspoint.com/assembly_programming/
- **OSDev Wiki**
<https://wiki.osdev.org/>
Risorsa completa per sviluppo sistemi operativi
- **x86 Assembly Guide (Yale)**
<https://flint.cs.yale.edu/cs421/papers/x86-asm/asm.html>
- **8086 Instruction Reference**
http://www.gabrielececchetti.it/Teaching/CalcolatoriElettronici/Docs/i8086_instruction_set.pdf

C.3.2 Emulatori e Tools

- **EMU8086**
<https://emu8086-microprocessor-emulator.en.softonic.com/>
Emulatore completo con IDE integrato
- **NASM (Netwide Assembler)**
<https://www.nasm.us/>
Assembler open-source multi-piattaforma
- **MASM (Microsoft Macro Assembler)**
<https://docs.microsoft.com/en-us/cpp/assembler/masm/>
Assembler ufficiale Microsoft
- **DOSBox**
<https://www.dosbox.com/>
Emulatore DOS per eseguire programmi 16-bit
- **TASM (Turbo Assembler)**
Borland Turbo Assembler, compatibile MASM
- **Debug.exe**
Debugger DOS integrato, utile per disassemblaggio
- **TD (Turbo Debugger)**
Debugger avanzato con interfaccia grafica

C.3.3 Simulatori Online

- **Compiler Explorer (Godbolt)**
<https://godbolt.org/>
Compila C/C++ e mostra Assembly risultante

- **Online x86 Assembler**
<https://defuse.ca/online-x86-assembler.htm>
- **8086 Emulator Online**
https://www.tutorialspoint.com/compile_assembly_online.php

C.4 Comunità e Forum

- **Stack Overflow**
Tag: [assembly], [x86], [8086]
<https://stackoverflow.com/questions/tagged/assembly>
- **Reddit r/asm**
<https://www.reddit.com/r/asm/>
- **OSDev Forums**
<https://forum.osdev.org/>
- **Flat Assembler Board**
<https://board.flatassembler.net/>

C.5 Video Corsi

- **Ben Eater — Building an 8-bit Computer**
<https://www.youtube.com/c/BenEater>
Serie fantastica su architettura computer
- **Davy Wybiral — Assembly Language Programming**
<https://www.youtube.com/playlist?list=PLmxT2pVYo5LB5EzTPZGfFN0c2GDiSXgQe>
- **Code Vault — x86 Assembly**
<https://www.youtube.com/c/CodeVault>

C.6 Progetti e Codice Esempio

- **GitHub — Assembly Examples**
<https://github.com/topics/assembly-8086>
- **OSDev Bare Bones**
https://wiki.osdev.org/Bare_Bones
Kernel minimale in Assembly
- **Bootlin (ex Compiler Explorer)**
Confronto codice C vs Assembly

C.7 Specifiche Hardware

- **Intel Datasheets**
<https://www.intel.com/content/www/us/en/design/resource-design-center.html>
- **8259 PIC Datasheet**
Programmable Interrupt Controller
- **8253/8254 PIT Datasheet**
Programmable Interval Timer
- **8042 Keyboard Controller**
- **16550 UART Datasheet**
Universal Asynchronous Receiver/Transmitter

C.8 Reverse Engineering

- **IDA Pro**
<https://hex-rays.com/ida-pro/>
Disassembler e debugger professionale
- **Ghidra**
<https://ghidra-sre.org/>
Tool NSA open-source per reverse engineering
- **Radare2**
<https://rada.re/>
Framework open-source per RE
- **x64dbg**
<https://x64dbg.com/>
Debugger Windows open-source

C.9 Sicurezza e Exploit Development

- **Exploit Education**
<https://exploit.education/>
Wargames per imparare exploit
- **Smashing The Stack For Fun And Profit**
Aleph One, Phrack Magazine
Articolo classico su buffer overflow
- **Shellcode Database**
<http://shell-storm.org/shellcode/>

C.10 Standard e Riferimenti

- **System V ABI**
Application Binary Interface standard
- **Intel[®] 64 and IA-32 Architectures Software Developer Manuals**
<https://software.intel.com/content/www/us/en/develop/articles/intel-sdm.html>
- **AMD64 Architecture Programmer's Manual**
- **NASM Documentation**
<https://www.nasm.us/docs.php>

C.11 Risorse in Italiano

- **Appunti di Calcolatori Elettronici**
Università varie, reperibili online
- **Forum HTML.it — Assembly**
<https://forum.html.it/forum/assembly>
- **Wikibooks — Programmazione x86 Assembly**
https://it.wikibooks.org/wiki/Programmazione_x86_Assembly

C.12 Riviste e Pubblicazioni

- **Dr. Dobb's Journal**
Articoli storici su Assembly e ottimizzazione
- **Phrack Magazine**
<http://phrack.org/>
E-zine su hacking e programmazione low-level
- **MSDN Magazine**
Articoli tecnici Microsoft

C.13 Note Finali

Questo elenco non è esaustivo ma rappresenta un punto di partenza solido per approfondire la programmazione Assembly 8086. Si consiglia di:

- Praticare regolarmente scrivendo codice
- Partecipare a comunità online per confronto
- Studiare codice esistente (open source)

- Sperimentare con progetti personali
- Usare debugger per comprendere il funzionamento interno

*La programmazione Assembly è un'arte che richiede pratica costante.
Non scoraggiatevi di fronte alle difficoltà iniziali!*