CPU 80x86 / IA32 / IA64

Architettura Intel

- Scelta Intel
  - Mantenere Compatibilità all'indietro verso architetture di processori precedenti
- Pro:
  - Compatibilità Software scritto per precedenti modelli di processore
- Contro:
  - Più difficile ottimizzazione dei nuovi sistemi.

Architettura Intel

- Per capire l'architettura dei processori IA 32/64 è necessario analizzare l'architettura dei processori 80x86.
- L'architettura di base ed il funzionamento dei nuovi processori rimane simile
- Si aggiungono nuovi componenti e funzionalità per superare i limiti dei processori precedenti
- (teoricamente) un programma compilato per un 8086 deve poter girare anche su un recente core2.

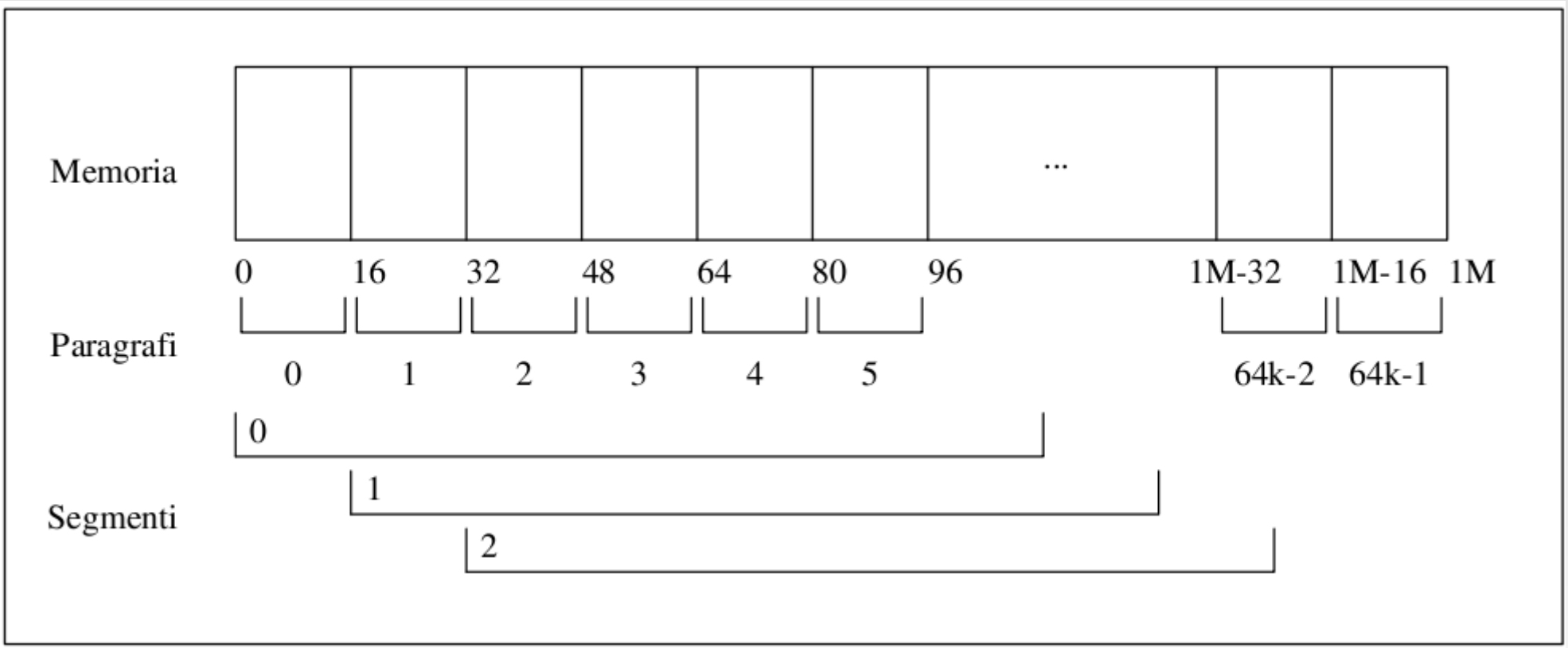
Intel 8086: Panoramica generale

- Processore *general purpose* a *16 bit*
- Capacità di Indirizzamento: 1 Mbyte
- 14 registri interni da 16 bit
- 7 modi di indirizzamento
- Alimentazione 5 V
- 48 Pin
- CISC
- I/O Mapped

Gestione della Memoria

- Memoria suddivisa in:
  - Paragrafi :
    - zone di memoria costituite da 16 byte contigui
    - Numerati a partire dalla locazione 0x0h
    - Non possono sovrapporsi
  - Segmenti:
    - Zone di memoria costituite da 64K byte contigui.
    - Si possono gestire fino a 64K segmenti
    - Ogni segmento inizia in corrispondenza di un paragrafo (un indirizzo multiplo di 16)
    - I segmenti possono sovrapporsi (Overlapping Segments)

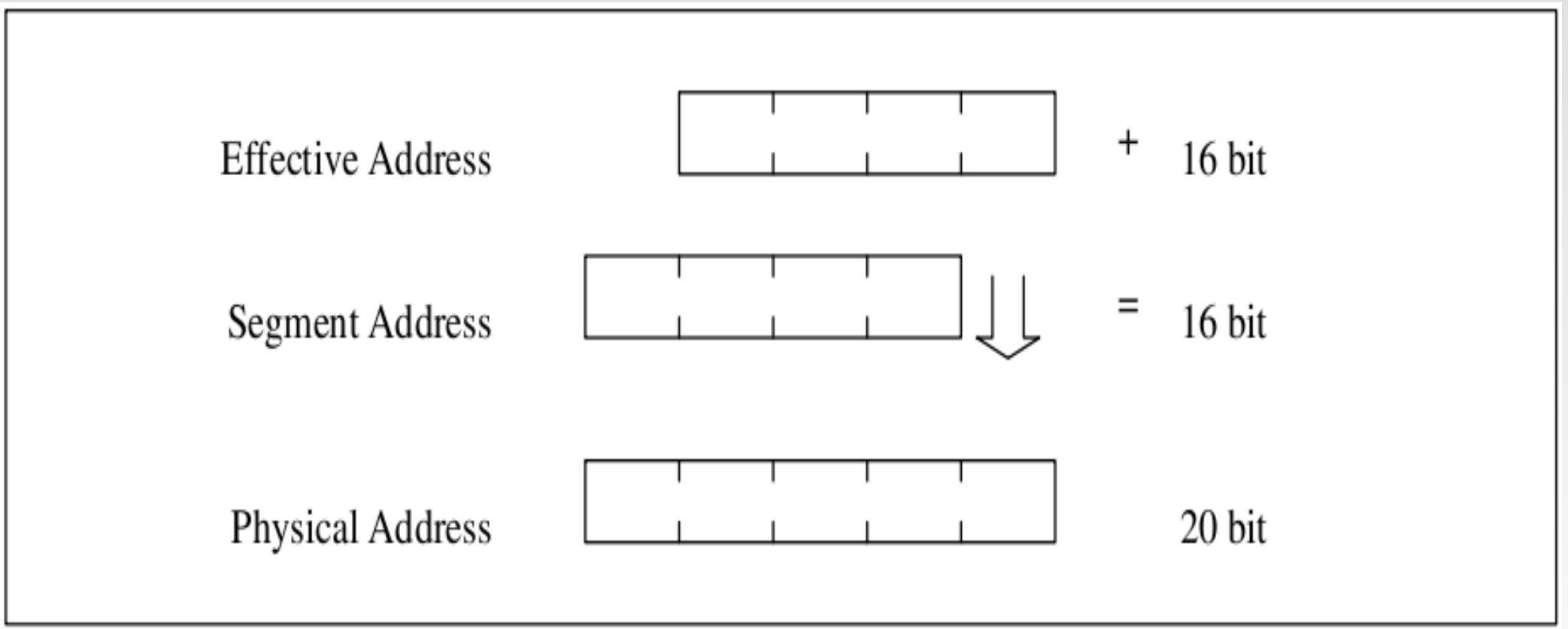
Gestion della Memoria



Indirizzamento

- Indirizzo Fisico di una cella di Memoria è espresso da 20 bit (1MByte di memoria)
- I Registri sono a 16 bit
- Non si pò indirizzare una locazione di memoria con 1 solo registro
- Si utilizzano due registri:
  - Segment Address
    - Indirizzo di testa del segmento, ottenuto moltiplicando per 16 il numero del segmento
  - Effective Address
    - Indirizzo effettivo all'interno del segmento (offset)

Indirizzamento



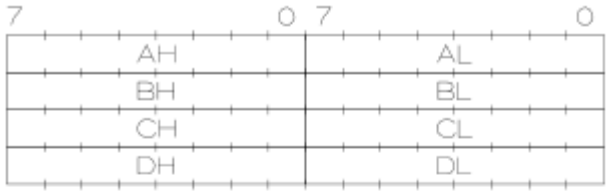
Registri

- 4 General Purpose
- 5 Pointer (o Index o Offset) Register
- 4 Segment Register
- 1 Flag Register

Registri

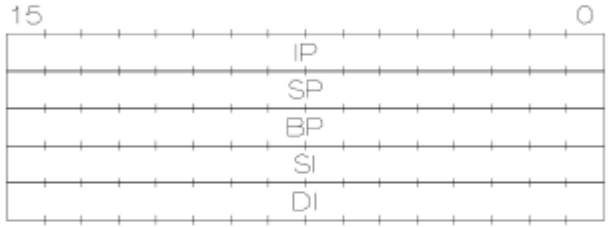
General Purpose Registers

AX (Accumulator)
BX (Base)
CX (Counter)
DX (Data)



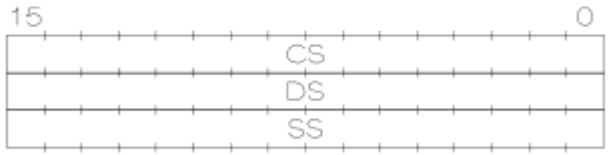
Pointer Registers

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



Segment Registers

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



Flag Register

Flags

General Purpose Registers

- AX: Accumulatore : AH+AL
- BX: Base : BH+BL
- CX: Count : CH + CL
- DX: Data : DH + DL
- I registri sono indirizzabili direttamente nella parte alta o bassa come se fossero 2 registri da 8 bit
- Anche essendo a registri generali, i registri sono ottimizzati per particolari usi

Pointer Registers

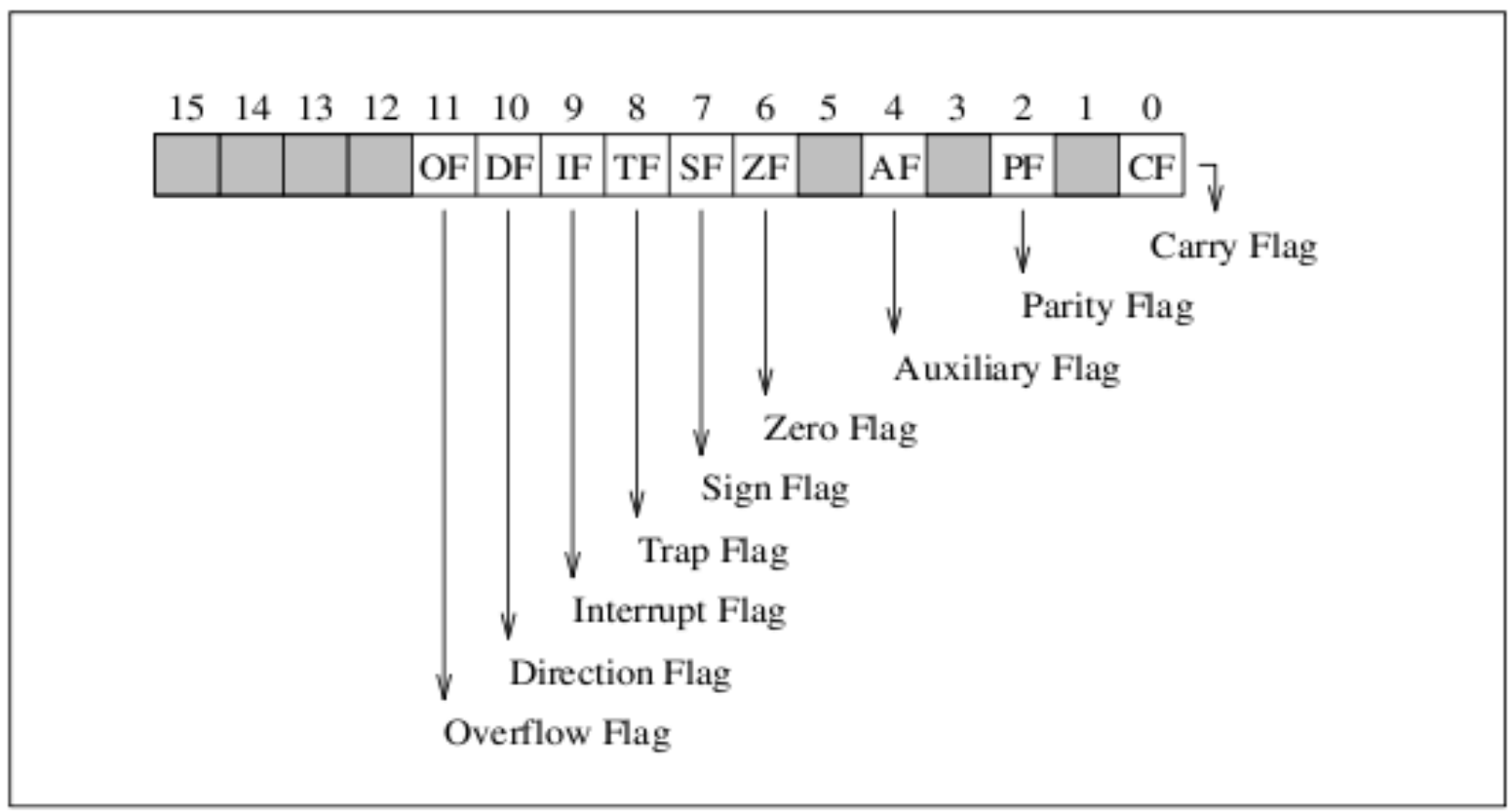
- Usati come registri indice o offeset per puntatori a dati in memoria
- Base Pointer
  - BX (Data Segment Base Pointer, usato anche come GP)
  - BP: Stack Segment Base Pointer
- Index Pointer
  - SI: Source Index Pointer
  - DI: Destination Index Pointer
- Stack Pointer
  - SP : Stack Pointer
- Instruction Pointer
  - IP: Instruction Pointer

Segment Registers

- Contengono l'indirizzo di testa dei segmenti usati dal programma in esecuzione
- CS: Code Segment Register
  - Per il segmento Codice
- SS: Stack Segment Register
  - Indirizzo di testa del segmento Stack
- DS: Data Segment Register
  - Indirizzo di testa del segmento dati
- ES: Extra Segment Register:
  - Usato per definire segmenti ausiliari

Flag Register

- 9 Bit di Flag
- Divisi in Flag di Stato e di Controllo



Flag di Stato

- CF: Carry Flag – riporto
- PF: Parity Flag - pari ad 1 se un'operazione ha un risultato con numero di 1 pari
- AF: Auxiliary Carry Flag - posto ad 1 quando si produce un riporto tra il bit 3 e 4 di un operando; usato sui decimali packed
- ZF: Zero Flag – Risultato nullo
- SF: Sign Flag – ripete il bit più significativo del risultato di un'operazione
- OF: Overflow Flag – overflow

Flag di Controllo

- TF: Trap Flag: Forza l'esecuzione single step
- IF: Interrupt Enable Flag: disabilita (0) eventuali Interruzioni Esterne
- DF: Direction Flag – usato per regolare incremento(0) o decremento(1) dei registri indice durante le operazioni sulle stringhe/vettori

Mnemonico per le istruzioni

- Nel gergo Intel, le Istruzioni appaiono nel seguente formato:
  - MOV AX, BX
- Istruzioni a 0,1,2 operandi
- L'operando a sinistra: DESTINAZIONE
- Operando a destr: SORGENTE
  - AX ← BX

Modi di Indirizzamento

- Registro
- Immediato
- Diretto
- Registro Indiretto
- Relativo alla Base
- Diretto Indicizzato
- Base Indicizzato

Indirizzamento registro

- L'<ea> è un registro:
- ES:
  - MOV AX,BX

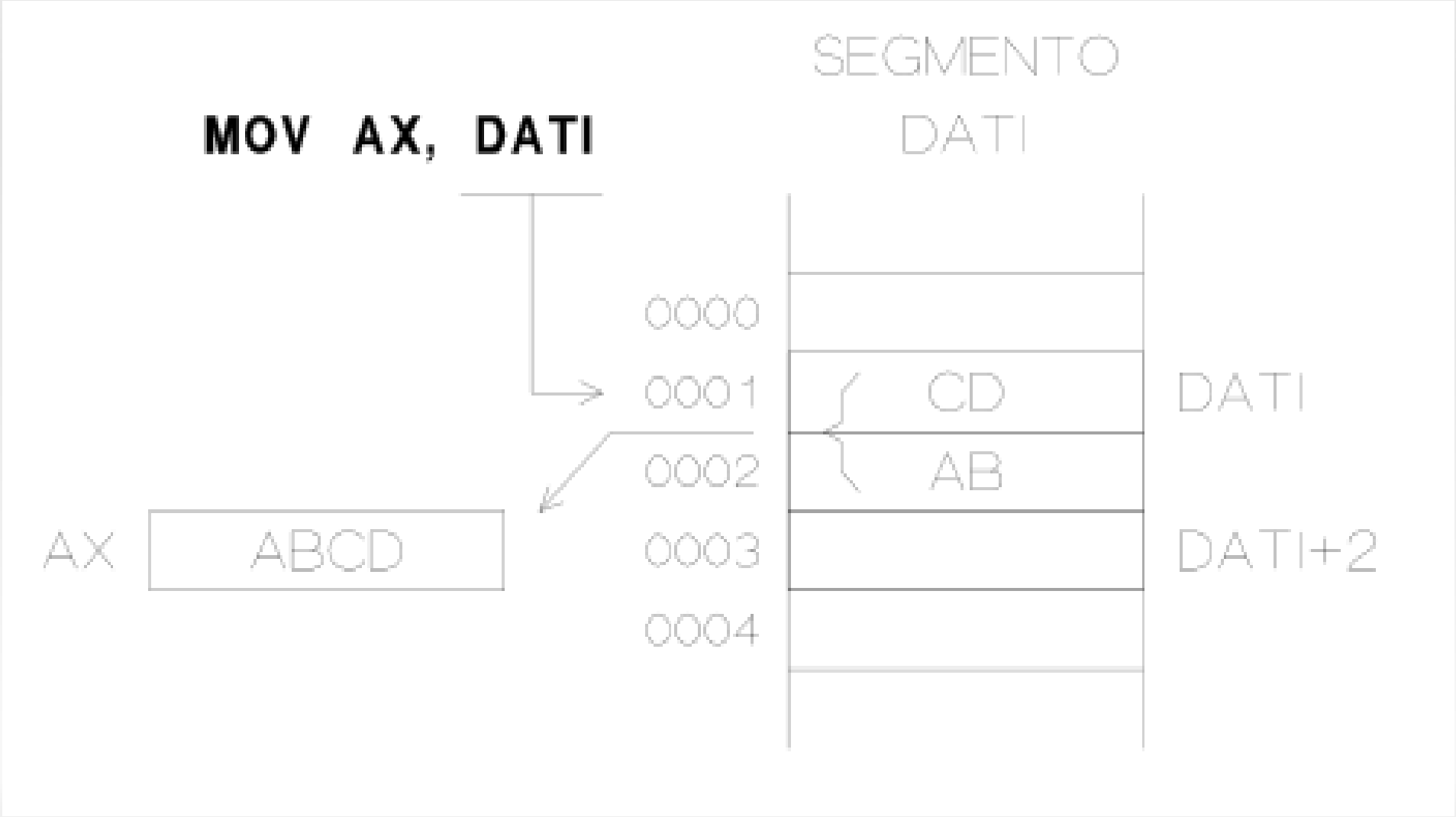
Indirizzamento Immediato

- L'<ea> è il dato da utilizzare
- ES:
  - MOV AX, 40h
  - MOV AL, (4*3h)/10b
- L'operando può anche essere definito da una direttiva EQU
  - OP EQU 42
  - MOV AX,K

Indirizzamento Diretto

- Nell'istruzione è specificato il nome di una variabile
- L'<ea> fa riferimento all'indirizzo della parola di memoria identificata da una label
- L'indirizzo FISICO si ottiene sommando l'<ea> con il DS shiftato di 4 posti a sinistra.
- ES:
  - MOV AX, DATI
  - MOV AX, DATI+4

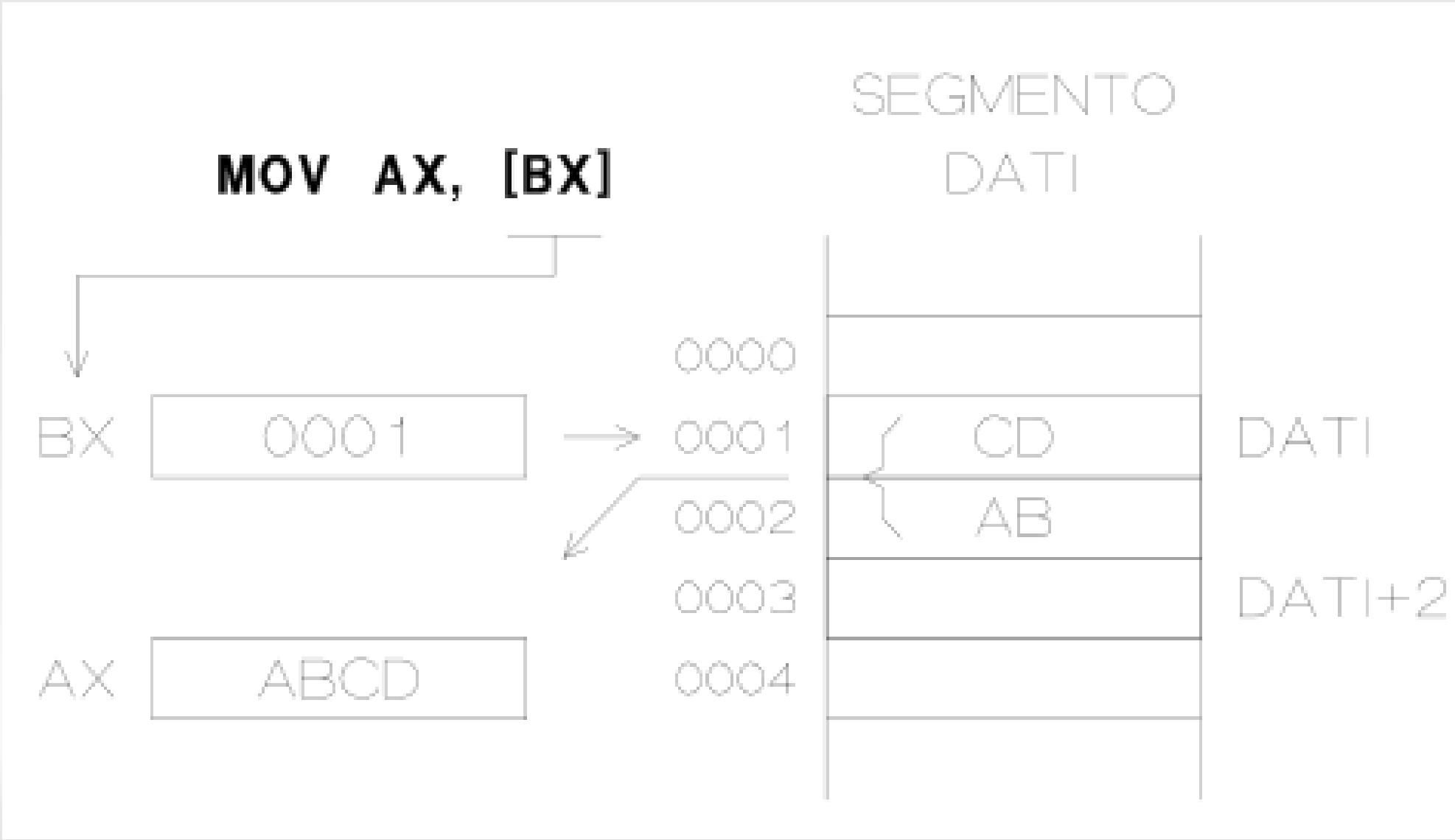
Indirizzamento Diretto



Indirizzamento Registro-Indiretto

- L'<ea> dell'operando è contentuto in uno dei registri:
  - Base
  - Index Register (DI o SI)
  - Base Pointer (BP)
- Es:
  - MOV AX, [BX]
  - MOV AX, [SI]

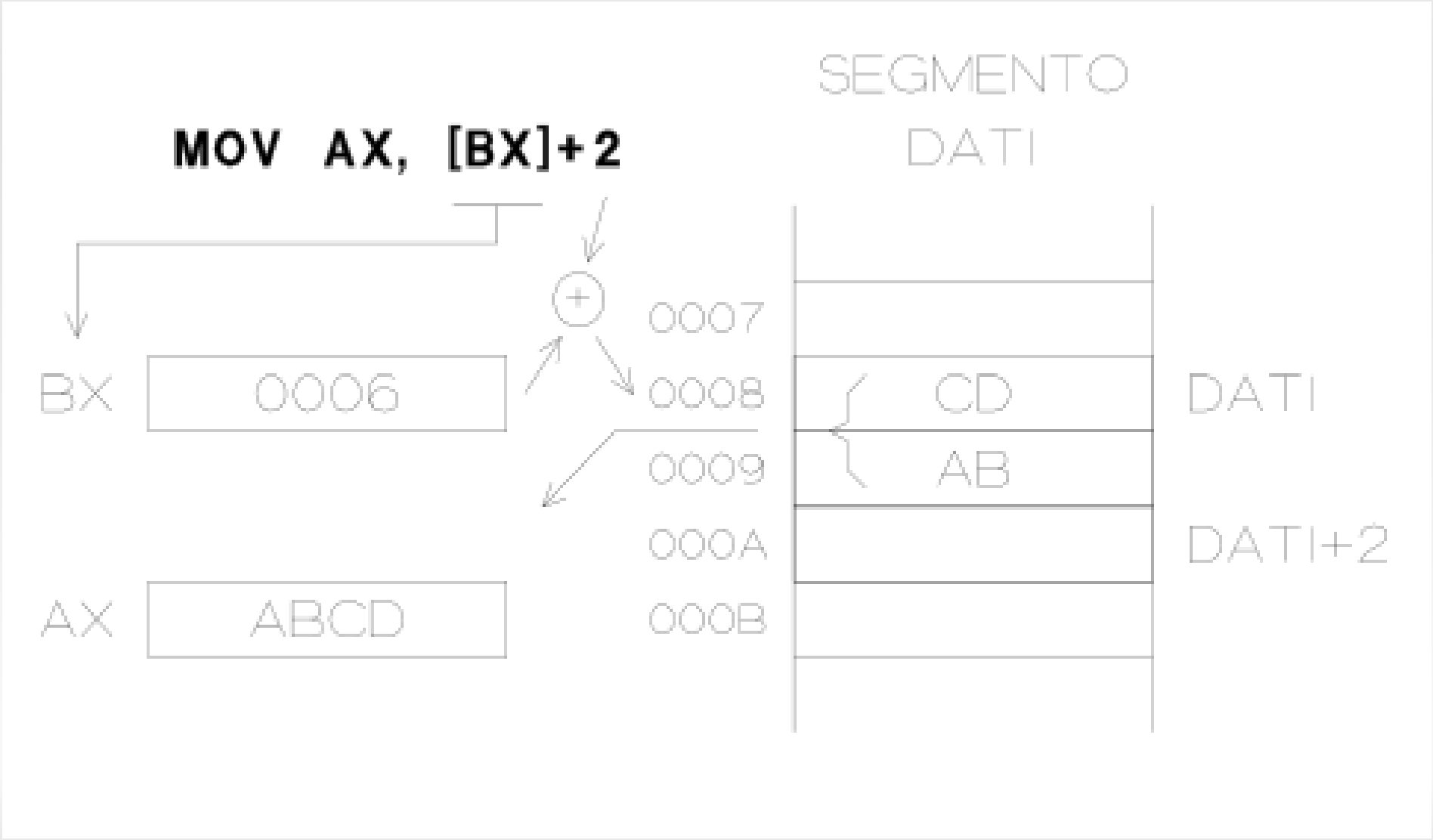
Indirizzamento Registro-Indiretto



Indirizzamento Relativo alla Base

- L'<ea> dell'operando si calcola sommando il contenuto di un Base Register (BX o BP) ad un displacement rappresentato da una costante nell'istruzione
- ES:
  - MOV AX,[BX+8]
  - MOV AX, 8[BX]
  - MOV AX, [BX] +2

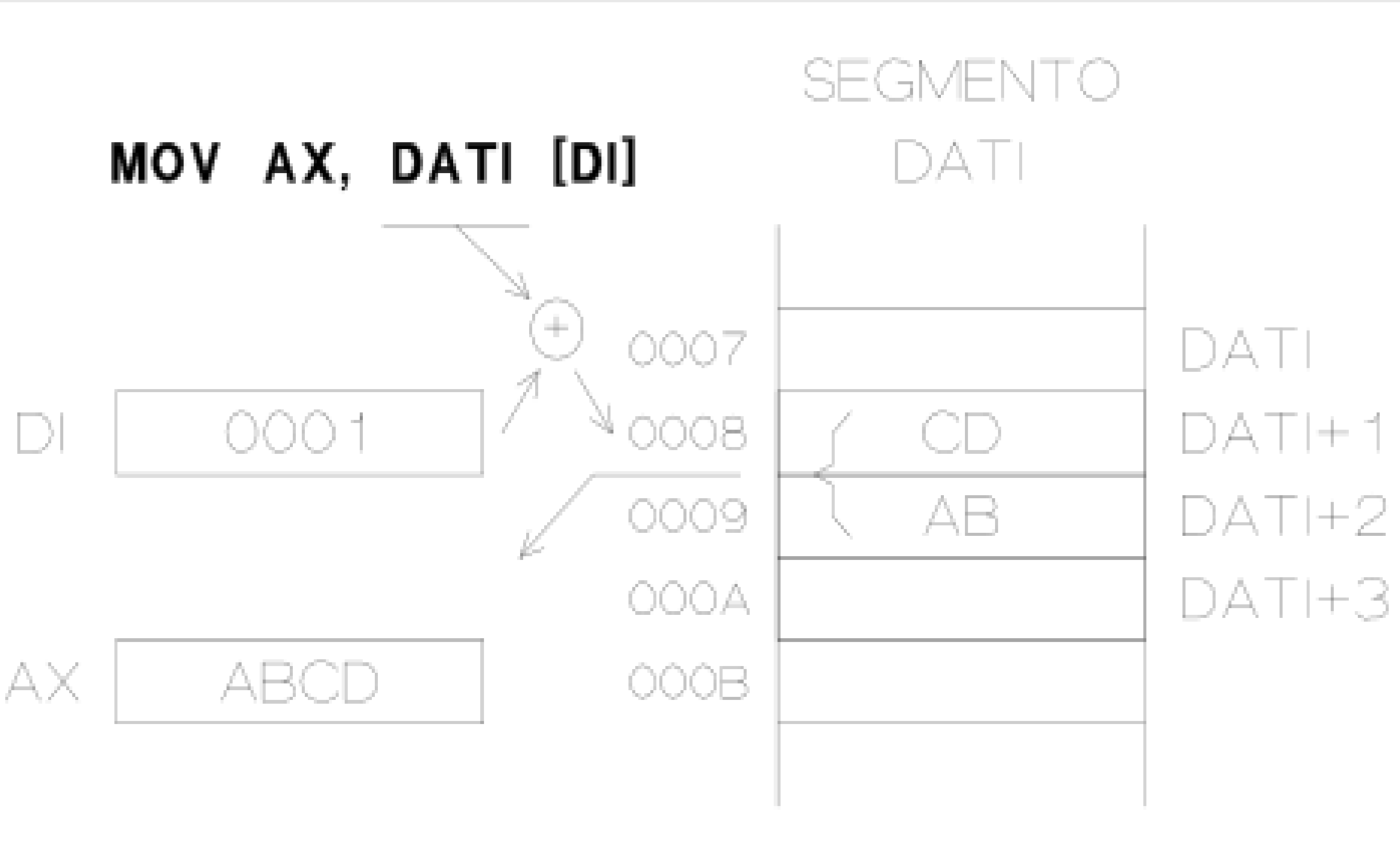
Indirizzamento Relativo alla Base



Indirizzamento Diretto Indicizzato

- L' <ea> dell'operando è calcolato sommando il valore di un offset contenuto in una variabile, ad un displacement contenuto in un Index register (SI o DI)
- ES:
  - MOV AX, DATI [DI]

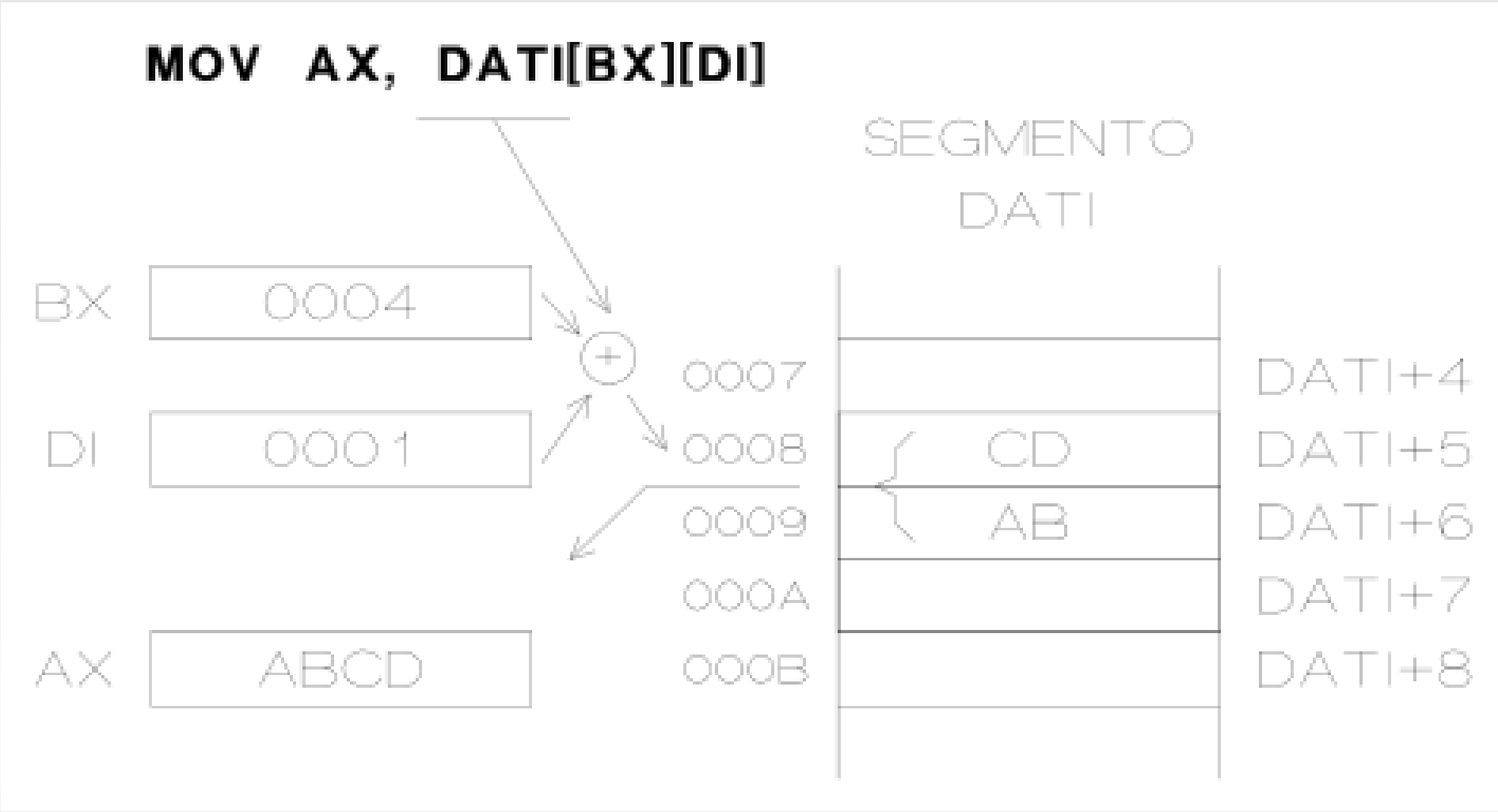
Indirizzamento Diretto Indicizzato



Indirizzamento Indicizzato alla Base

- L'<ea> dell'operando è la somma di:
  - Contenuto di un Base Register (BX o BP)
  - Contenuto di un odegli Index Register (SI o DI)
  - Un displacement contenuto nell'istruzione
- Es:
  - MOV AX, DATI[BX][DI]

Indirizzamento Indicizzato alla Base



Segment Register utilizzati

- I Segment Register utilizzati in ogni tipo di indirizzamento:

Indirizzamento	Formato	SegReg
Register	registro	nessuno
Immediate	dato	nessuno
Direct	variabile	DS
Register Indirect	[BX]	DS
	[BP]	SS
	[DI]	DS
	[SI]	DS
Base Relative	label [BX]	DS
	label [BP]	SS
Direct Indexed	label [DI]	DS
	label [SI]	DS
Base Indexed	label [BX] [SI]	DS
	label [BX] [DI]	DS
	label [BP] [SI]	SS
	label [BP] [DI]	SS

Costanti

- Binarie
  - 00011010B
- Ottali
  - 15O, 15Q
- Esadecimali
  - 0Dh, 0BADh, 12h (devono iniziare con un numero)
- Decimali
  - 42, 42D
- ASCII:
  - 'H', 'Hello world'
- Reali in base 10:
  - 3.14, 8.82E-21

Istruzioni

- Trasferimento Dati
- Aritmetiche
- Manipolazione di Bit
- Trasferimento di Controllo
- Manipolazione di Stringhe
- Gestione Interruzioni
- Controllo della CPU

Trasferimento Dati

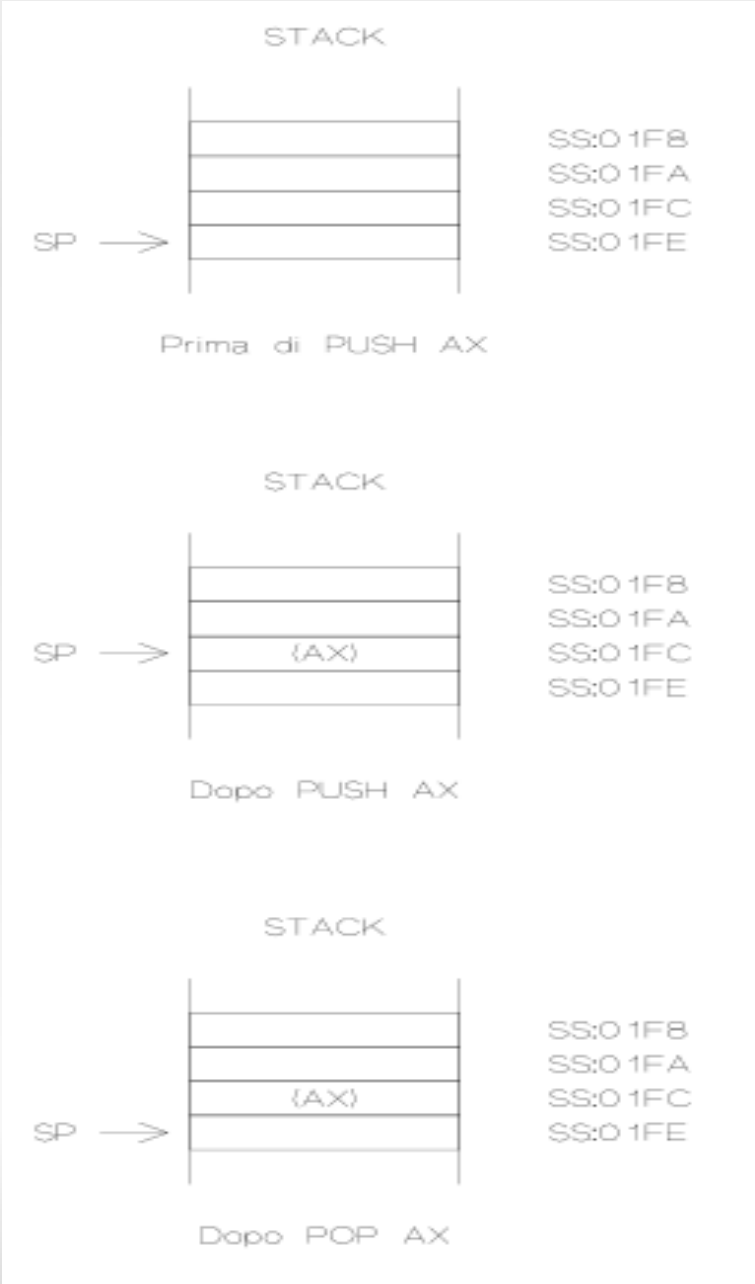
	OpCode	Descrizione
General Purpose	MOV	Move (Byte or Word)
	POP	Pop a Word from the Stack
	PUSH	Push Word onto Stack
	XCHG	Exchange Registers
	XLAT	Translate
Input/Output	IN	Input Byte or Word
	OUT	Output to Port
Trasf. di indirizzi	LDS	Load Pointer Using DS
	LEA	Load Effective Address
	LES	Load Pointer Using ES
Trasf. Flag Register	LAHF	Load Register AH from
	SAHF	Store Register AH into
	POPF	Pop Flags from the Stack
	PUSHF	Push Flags onto Stack

MOV: indirizzamenti non ammessi

- Mem ← mem
- Segment register ← immediato
- Segment register ← segment register
- Cs come destinazione

Uso dello stack

- Codici operativi di push e pop espliciti:
  - PUSH AX
  - PUSH ES
  - POP SI
  - POP BX
  - ...



Operazioni Aritmetiche

	OpCode	Descrizione
Addizione	AAA	ASCII Adjust after Addition
	ADC	Add with Carry
	ADD	Addition
	DAA	Decimal Adjust after Addition
	INC	Increment
Sottrazione	AAS	ASCII Adjust after Subtraction
	SUB	Subtract
	SBB	Subtract with Borrow
	DAS	Decimal Adjust after Subtraction
	DEC	Decrement
	CMP	Compare
	NEG	Negate
Moltiplicazione	AAM	ASCII Adjust after Multiply
	IMUL	Integer Multiply, Signed
	MUL	Multiply, Unsigned
Divisione	AAD	ASCII Adjust before Division
	DIV	Divide, Unsigned
	IDIV	Integer Divide, Signed
Conversione	CBW	Convert Byte to Word
	CWD	Convert Word to Doubleword

Dati nelle istruzioni aritmetiche

- Numeri Binari unsigned su 8 o 16 bit
- Numeri binari signed su 8 o 16 bit
- Numeri decimali *packed*
  - Ogni byte contiene due numeri decimali codificati in BCD; la cifra più significativa è allocata nei 4 bit superiori
- Numeri decimali *unpacked*.
  - Ogni byte contiene un solo numero decimale BCD nei 4 bit inferiori; i 4 bit superiori devono essere 0 se il numero viene usato in una moltiplicazione o in una divisione

Operazioni su 32 bit

- ADC: Add with Carry
  - Dest ← dest + source + carry
  - ADD AX,CX ; somma i 16 bit meno sign.
  - ADC BX,DX ; somma i 16 bit più sign.
- SBB: Subtract with borrow
  - Dest ← dest – source - carry

Moltiplicazione e Division

- Codici operativi
  - MUL, DIV (senza segno)
  - IMUL, e IDIV (con segno)
- Operazioni con 1 operando (Reg. Gen. O una locazione di memoria)
- Il parallelismo dell'operazione dipende da quello degli operandi

Moltiplicazione e Divisione

- Moltiplicazione
  - AX ← AL * source-8-bit (byte)
  - DX:AX ← AX * source-16-bit (word)
- Divisione
  - AL ← CEIL(AX/source-8-bit), AH ← resto (byte)
  - AX ← CEIL (DX:AX/source-16-bit), DX ← resto (w)

Operazioni sui Numero decimali

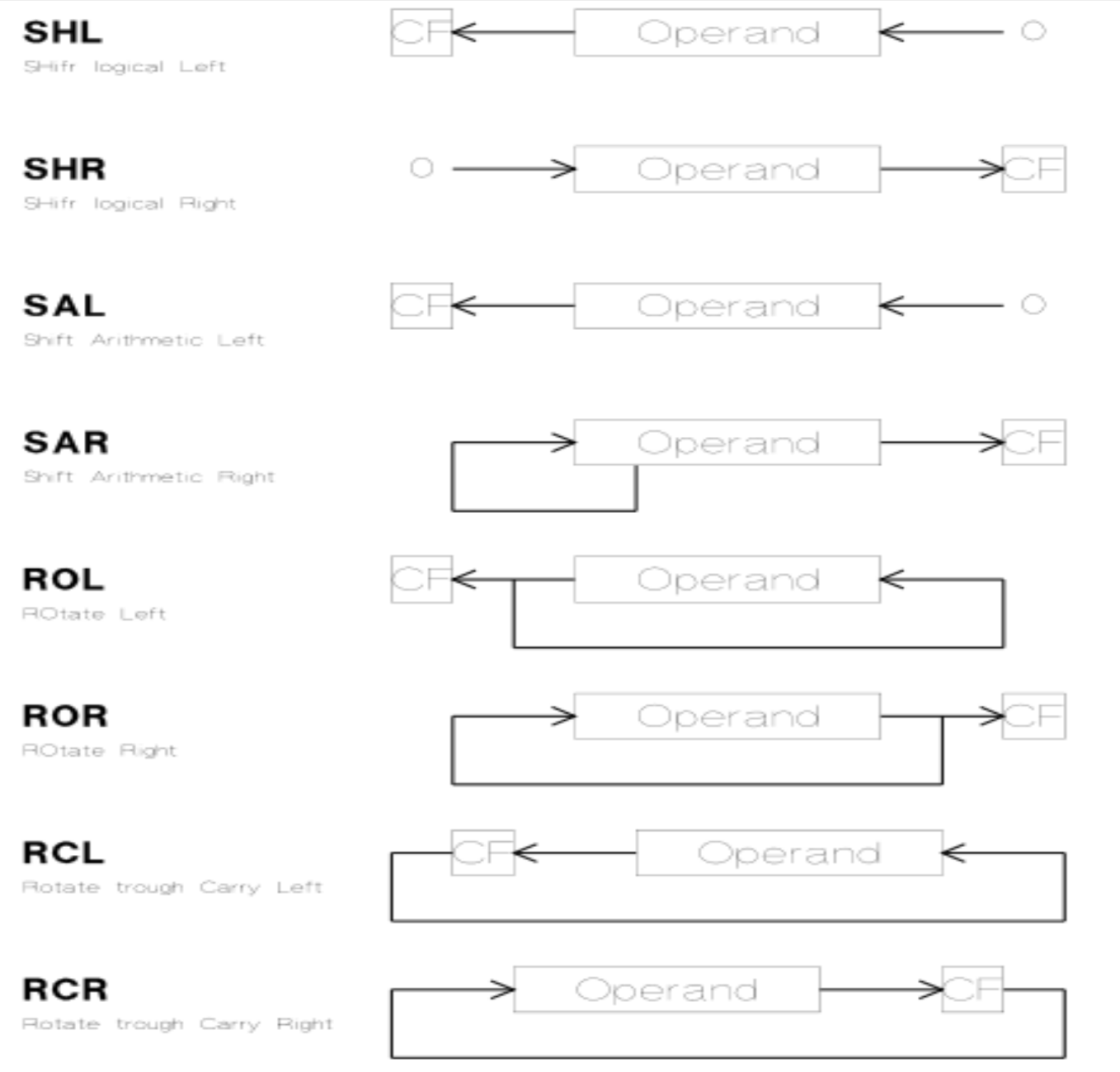
- Senza Operandi (lavorano sempre su AL)
- Bisogna applicare le conversioni di tipo sugli operandi se necessario
- AAA:
  - Risultato add → unpacked
- AAS
  - Risultato sub → unpacked
- AAM
  - Risultato mul → unpacked
- DAA :
  - Risultato Add → packed
- ---

binario:	0000 0000 0010 0011
decimale packed:	0011 0101
decimale unpacked:	0000 0011 0000 0101

Manipolazione di Bit

	OpCode	Descrizione
Logiche		
	AND	Logical AND
	OR	Logical OR
	XOR	Exclusive OR
	NOT	Logical NOT
	TEST	Test
Di Traslazione		
	SAL	Shift Arithmetic Left (=SHL)
	SAR	Shift Arithmetic Right
	SHL	Shift Logical Left (=SAL)
	SHR	Shift Logical Right
Di Rotazione		
	ROL	Rotate Left
	ROR	Rotate Right
	RCL	Rotate through Carry Left
	RCR	Rotate through Carry Right

Shift e Rotate



Salti

	OpCode	Descrizione
Salti incondizionati	CALL	Call Procedure
	RET	Return from Procedure
	JMP	Jump Unconditionally
Salti condizionati	JA, JNBE	Jump If Above
	JAE, JNB	Jump If Above or Equal
	JB, JNAE, JC	Jump If Below
	JBE, JNA	Jump If Below or Equal
	JCXZ	Jump if CX Register Zero
	JE, JZ	Jump If Equal
	JG, JNLE	Jump If Greater
	JGE, JNL	Jump If Greater or Equal
	JL, JNGE	Jump If Less
	JLE, JNG	Jump If Less or Equal
	JNC	Jump If No Carry
	JNE, JNZ	Jump If Not Equal
	JNO	Jump If No Overflow
	JNP, JPO	Jump If No Parity
	JNS	Jump If No Sign
	JO	Jump If Overflow
	JP, JPE	Jump If Parity
	JS	Jump If Sign
Istruzioni iterative	LOOP	Loop on Count
	LOOPE, LOOPZ	Loop While Equal
	LOOPNE, LOOPNZ	Loop While Not Equal

Salti Condizionati

- Il salto viene eseguito in base al valore di uno o più flag
- La distanza relativa dell'istruzione a cui saltare deve essere compresa tra -128 e + 128
- Non sono ammessi salti ad altri segmenti

Salti Incondizionati

- Il salto avviene sempre
- Istruzioni posizionate ovunque
  - Anche in diversi segmenti

Salti

- Attenzione!!!:
  - Alcuni codici di salto sono ridondanti
    - JA (jump if above) = JNBE (Jump if not below or Equal)

Procedure

- Codici di CALL e RET
- Allocazione parametri e indirizzi di ritorno sullo stack
- Procedura NEAR:
  - Può essere chiamata solo dall'interno dello stesso segmento a cui appartiene
- Procedura FAR
  - Può essere chiamata da un qualsiasi segmento di codice
- Il tipo di procedura va dichiarato

CALL

- Salva IP (e CS se FAR) sullo stack tramite un push
- Carica in IP (e CS se FAR) il calore dell'indirizzo di partenza della procedura

RET

- Ripristina tramite un pop il valore di IP (e di CS se FAR) salvato sullo stack
- Riprende la normale esecuzione

Iterazioni

- Codici Operativi espliciti per i loop
  - CodOp Label
- LOOP:
  - CX--; if CX != 0 JMP Label
- LOOPE/LOOPZ
  - CX--; if CX!=0 && ZF==1 JMP Label
- LOOPNE/LOOPNZ
  - CX--; if CX !=0 && ZF == 0 JMP Label

Manipolazioni di Stringhe

	OpCode	Descrizione
Istruzioni di Spostamento	MOVS	Move String (Byte or Word)
	MOVSB	Move String Byte
	MOVSW	Move String Word
Istruzioni di Confronto	CMPS	Compare String (Byte or Word)
	CMPSB	Compare String Byte
	CMPSW	Compare String Word
Istruzioni di Ricerca	SCAS	Scan String (Byte or Word)
	SCASB	Scan String Byte
	SCASW	Scan String Word
Istruzioni di Caricamento	LODS	Load String (Byte or Word)
	LODSB	Load String Byte
	LODSW	Load String Word
Istruzioni di Scrittura	STOS	Store String (Byte or Word)
	STOSB	Store String Byte
	STOSW	Store String Word

Manipolazioni di Stringhe

- Operano su dati consecutivi in memoria costituiti da byte o word
- Operazioni
  - Move : spostano un dato da una posizione all'altra
  - Compare: confrontano 2 dati
  - Scan: Ricercano un dato in memoria
  - Load: Caricano il dato dalla memoria in AX
  - Store: Scrivono in memoria un dato da AX

Manipolazione di Stringhe

- L'accesso ai dati avviene tramite SI(per DS) e DI (per ES)
- SI e DI vengono modificati al termine di ogni operazione:
  - Incrementati se DF ==0
  - Decrementati se DF == 1
- SI e DI vengono incrementati (o decrementati)
  - Di 1 se l'operazione è su byte
  - Di 2 se l'operazione è su word
- DF viene resettato dalle istr. STD e CLD

Manipolazioni di Stringhe

- Hanno 3 forme:
  - MOVSB, sui byte, no operandi
  - MOVSW, su word, no operandi
  - MOVS su byte o word, 1 o 2 operandi
    - Vengono tradotte in MOVSB dall'assemblatore
- ES:
  - STR1 DB 100 DUP(?)
  - STR2 DB 100 DUP(?)
  - MOVS STR1,STR2

Manipolazione di Stringhe

- Prefissi di Ripetizione:
- Op. per operarein costrutti iterativi
- Uso di Prefissi di ripetizione

	OpCode	Descrizione
Prefissi di Ripetizione		
	REP	Repeat
	REPE	Repeat While Equal
	REPNE	Repeat While Not Equal
	REPNZ	Repeat While Not Zero
	REPZ	Repeat While Zero

Gestione delle Interruzioni

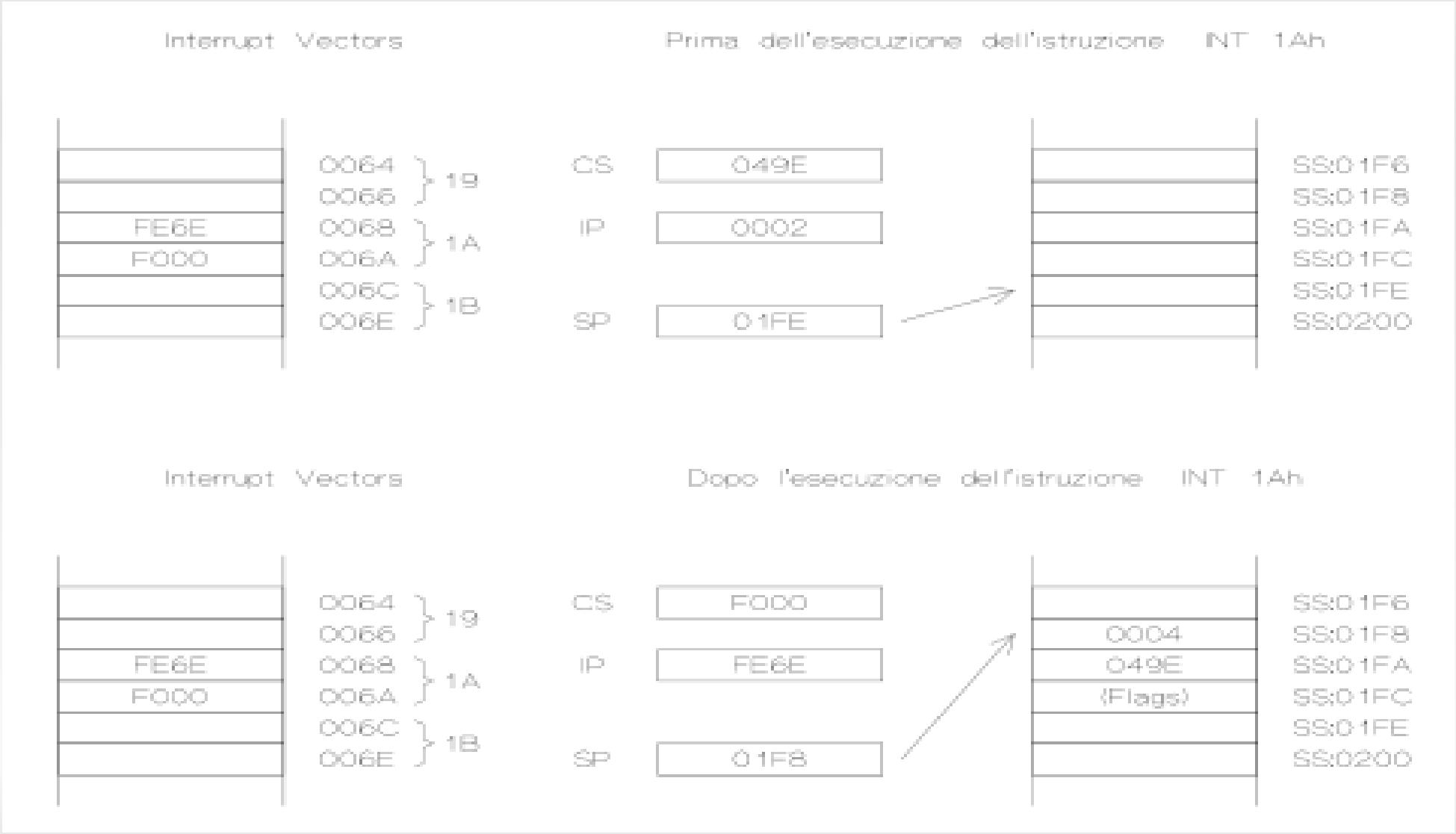
- Permettono di gestire gli Interrupt Software (Trap)
- Causano :
  - Stack ← IP,CS
  - Stack ← FlagReg
  - Esecuzione ISR (autovettorizzata)

	OpCode	Descrizione
Manipolazione di Interruzioni	INT	Interrupt
	INTO	Interrupt on Overflow
	IRET	Interrupt Return

Gestione delle Interruzioni

- Interruzioni Hardware
  - Attivano l'esecuzione della ISR (vettorizzata/autovettorizzata)
  - Vettore delle interruzioni :
    - 0000:0000h / 0000:03FFh (256 locazioni)
    - Gli elementi contengono 32 bit con l'indirizzo di partenza della ISR
- Una ISR e' sempre FAR

Gestione delle Interruzioni



Interrupt Software

- Istruzione INT
  - INT *int-number*
  - *Int-number* identifica l'indice nel vettore delle interruzioni
  - L'indirizzo fisico dell'interrupt viene ottenuto moltiplicanto per 4 l'int-number
- Un caso particolare: INTO
  - = INT 4 se OF == 1
  - NOP altrimenti

Interrupt Software

- INT :
  - Stack ← FlagReg
  - TF=0; IF = 0
  - Stack ← CS
  - CS ← Seconda word dell'Int vector
  - Stack ← IP
  - IP ← prima word dell'Int.vector
- Ritorno da Interruzione:
- IRET

Istruzioni di Controllo

	OpCode	Descrizione
Modifica dei flag	CLC	Clear Carry Flag
	CLD	Clear Direction Flag
	CLI	Clear Interrupt-Enable Flag
	CMC	Complement Carry Flag
	STC	Set Carry Flag
	STD	Set Direction Flag
	STI	Set Interrupt Enable Flag
Sincronizzazione	ESC	Escape
	HLT	Halt
	LOCK	Lock the Bus
	WAIT	Wait
Istruzione nulla	NOP	No Operation

Istruzioni di Sincronizzazione

- HLT: Forza la CPU in Idle fino a che
  - Non si riceve un INT esterno
  - Viene attivata la linea RESET
- WAIT Forza CPU in Idle
  - Ogni 5 colpi di clock di controlla la linea TEST se attivata, procede con l'istruzione successiva
- ESC : Invia istruzioni al coprocessore matematico 8087
- LOCK : è un prefisso: si usa per bloccare l'utilizzo di un bus da parte dell'istruzione (es: memoria condivisa)

Istruzione NULLA:

- NOP