ASSEMBLER 8086
Caratteristiche e Istruzioni Fondamentali

1. Modalità di indirizzamento

Sono disponibili, in generale, le modalità *immediata*, *tramite registro*, *assoluta* (o *diretta*), *indiretta*, *indicizzata*. Tuttavia, non tutte le modalità sono attive con ogni tipo di istruzione. Si riassumono qui di seguito le caratteristiche fondamentali delle diverse modalità.

Indirizzamento immediato: l'operando è una costante che segue immediatamente il codice mnemonico dell'istruzione.
Esempio: ADD AL, 8

Indirizzamento tramite registro: l'operando è contenuto in un registro indicato dal codice mnemonico dell'istruzione.
Esempio: MOV CX, DX

Indirizzamento assoluto: l'operando si trova a un indirizzo specificato direttamente nell'istruzione.
Esempi: ADD ALFA, 8 MOV BX, RES

Indirizzamento indiretto: l'operando si trova a un indirizzo puntato da un registro.
Esempi: ADD AX, (BX) MOV (BX), CX

Indirizzamento indicizzato: l'operando si trova a un indirizzo puntato da un registro base e/o un registro indice (BX, BP, SI o DI). A seconda di quali registri sono specificati, questa modalità dà luogo a molte varianti:

- con base[1]: ADD AX, (BX+8) MOV (BP+COST), CX
- con indice[2]: ADD AX, (DI+8) MOV (SI+COST), CX
- con entrambi: MOV (BX+DI), AX ADD(BX+DI+COST),AX

2. Elenco istruzioni

Istruzioni aritmetiche:

ADD (ADD) *Addizione*
ADC (ADd with Carry) *Addizione con riporto*

SUB (SUBtract) *Sottrazione*
SBB (SuBtract with Borrow) *Sottrazione con prestito*

---

1 Solo registri BX e BP. Può essere presente un registro di segmento.

2 Solo registri SI o DI. Non possono essere presenti registri di segmento.

MUL (MULtiply) *Moltiplicazione senza segno*
Il moltiplicando e il risultato sono sempre posti in registri fissi: solo il moltiplicatore (che è anche l'unico dato specificato) può essere preso da sorgenti diverse. Più esattamente, per moltiplicatori a 8 bit il moltiplicando sta sempre in AL e il risultato è sempre fornito in AX, mentre per moltiplicatori a 16 bit il moltiplicando sta sempre in AX e il risultato è sempre fornito in DX e AX. In entrambi i casi, se le parti più significative del prodotto (rispettivamente AH o DX) sono nulle, i flag CF e OF vengono resettati

IMUL (Integer MULtiply) *Moltiplicazione con segno* (fra interi).
Analoga a MUL, a parte il fatto che operandi e risultato sono considerati con segno.

DIV (DIVide) *Divisione senza segno*
Il dividendo e il risultato sono sempre posti in registri fissi: solo il divisore (che è anche l'unico dato specificato) può essere preso da sorgenti diverse. Più esattamente, per divisori di 8 bit il dividendo è sempre in AX e il resto e il quoziente sono forniti rispettivamente in AH e AL, mentre per divisori di 16 bit il dividendo sta sempre in AX e DX, il resto è fornito in DX e il quoziente in AX

IDIV (Integer DIVdide) *Divisione con segno* (fra interi)
Analoga a DIV, tranne per il fatto che gli operandi sono considerati con segno. Il resto assume il segno del dividendo.

NEG (NEGate) *Inversione del segno*
Inverte il segno di un valore, sottraendolo da zero. Modifica il dato su cui opera.

INC (INCrement) *Incremento (di 1)*
DEC (DECrement) *Decremento (di 1)*

Istruzioni di conversione byte/word/double word:

CBW (Convert Byte to Word) *Conversione da byte a word*
Converte un byte (posto in AL) in una word (posta in AX) estendendo il bit di segno del byte verso sinistra, fino a riempire la word.

CWD (Convert Word to Double word) *Conversione da word a double word*
Converte una word (posta in AX) in una double word (posta in DX e AX) estendendo il bit di segno della word verso sinistra, fino a riempire la double word.

Istruzioni logiche e di confronto:

NOT *not* logico
AND *and* logico
OR *or* logico
XOR *or esclusivo* logico

TEST (TEST) *Confronto logico*
Effettua un "and" logico fra i due operandi (che non vengono alterati) influenzando di conseguenza i flag e permettendo di condizionare le operazioni successive (tipicam. salti)

CMP (CoMPare) *Confronto (aritmetico) fra due valori*

Effettua una sottrazione fra i due valori senza modificare però nessuno dei due operandi e scartando il risultato. Influenza lo stato dei flag in modo da permettere di condizionare di conseguenza le operazioni successive.

Istruzioni di aggiustamento per valori codificati in BCD:

AAA (ASCII Adjust after Addition) *Aggiustamento ASCII dopo l'addizione*

Dopo un'addizione fra due numeri BCD non compattati[3], converte il risultato, contenuto di AL, in un numero BCD anch'esso non compattato e correttamente rappresentato. Perciò, al termine il nibble più significativo di AL è zero. Il valore finale è contenuto in AH (che può essere stato incrementato di 1 in caso di riporti) e AL.

Esempio: input AX=0307H (versione non compattata del valore BCD 37), BL=9;
output (dopo ADD AL,BL) AX=0310H;
output (dopo AAA) AX=0406H (valore BCD 46=37+9 non compattato).

Per ottenere la codifica ASCII è sufficiente a questo punto sommare la costante 30H a ogni byte del risultato[4].

AAS (ASCII Adjust after Subtraction) *Aggiustamento ASCII dopo la sottrazione*

Dopo una sottrazione fra due valori BCD non compattati (il minuendo contenuto in AH e AL, il sottraendo in qualche altro registro a 8 bit), converte il risultato, posto in AH e AL, in un valore BCD corretto anch'esso non compattato posto in AH (che può essere stato decrementato di 1 in caso di prestito) e AL.

Esempio: input AX=0307H (versione non compattata del valore BCD 37), BL=9;
output (dopo SUB AL,BL) AX=03FEH;
output (dopo AAS) AX=0208H (valore BCD 28=37-9 non compattato).

Anche qui, per ottenere la codifica ASCII è sufficiente a questo punto sommare la costante 30H a ogni byte del risultato.

AAM (ASCII Adjust after Multiplication) *Aggiustamento ASCII dopo la moltiplicazione*

Converte il risultato del prodotto (effettuato tramite MUL) di due numeri BCD non compattati, inizialmente posti in AH e AL, in un valore BCD pure non compattato e correttamente rapprtesentato, posto in AX.

Esempio: input AX=0309H (equivalente ai valori BCD 3 e 9);
output (dopo MUL AH) AX=001BH;
output (dopo AAM) AX=0207H (valore BCD 27=3*9 non compattato).

Anche qui, per ottenere la codifica ASCII è sufficiente a questo punto sommare la costante 30H a ogni byte del risultato.

AAD (ASCII Adjust before Division) *Aggiustamento ASCII prima della divisione*

Converte due cifre BCD non compattate, poste in AH e AL, in un singolo valore BCD compattato (pronto a fungere da dividendo in una divisione) posto in AL, azzerando AH.

Esempio: input AX=0309H (equivalente ai valori BCD 3 e 9), BL=2;
output (dopo AAD) AX=0027H (valore binario 27H=39, compattato).

---

[3] Un valore BCD si dice *compattato* quando ogni cifra è rappresentata da un nibble (e quindi un byte contiene due cifre), e *non compattato* quando invece ogni cifra è rappresentata su un byte, il cui nibble più significativo è mantenuto azzerato. Questa seconda forma, ancorché più dispendiosa sotto il profilo della memoria occupata, è utile quando il dato dev'essere convertito o presentato in formato ASCII, dato che in questa codifica ogni cifra è espressa da un valore binario (pari, detta N la cifra, a 30H+N) rappresentato su un byte.

[4] Ovvero, il che è equivalente, effettuare su ogni byte l'operazione OR 30H.

output (dopo DIV BL) AH=01 (resto), AL=13H=19 (quoziente);

Anche qui, per ottenere la codifica ASCII è sufficiente a questo punto sommare la costante 30H a ogni byte del risultato.

DAA (Decimal Adjust after Addition) *Aggiustamento BCD in seguito ad addizione*
Converte il risultato potenzialmente scorretto e incoerente di un'addizione fra valori BCD (compattati), contenuto in AL, in un valore BCD (compattato) corretto.

DAS (Decimal Adjust after Subtraction) *Aggiustamento BCD in seguito a sottrazione*
Converte il risultato potenzialmente scorretto e incoerente di una sottrazione fra valori BCD (compattati), contenuto in AL, in un valore BCD (compattato) corretto.

Istruzioni varie:

MOV (MOVe) *Trasferimento dati*
Trasferisce byte o word (8 o 16 bit risp.) fra registri, memoria e valori immediati

CALL (CALL) *Chiamata a subroutine*
RET (RET) *Ritorno da subroutine*

IN (INput) *Input da porta esterna*
OUT (OUTput) *Output da porta esterna*
LOCK (LOCK) *Blocco del bus*
Blocca il bus per impedire in hardware l'accesso al bus da parte di altre CPU durante l'esecuzione dell'istruzione successiva
WAIT (WAIT) *Attesa segnale di wait*
Attende un segnale hardware sul piedino WAIT. E' utile in presenza di coprocessori.
ESC (ESC) *Codici di escape*
E' utilizzato per inviare a un coprocessore esterno dei codici di controllo
HLT (HaLT) *Stop del processore*
Ferma il processore fino all'arrivo di un'interruzione.

INT (INTerrupt) *Generazione di un'interruzione software*
INTO (INTerrupt per Overflow) *Generazione di un'interruzione software a seguito di overflow*
IRET (Interrupt RETurn) *Ritorno da routine di gestione di un'interruzione*

NOP (No OPeration) *Nessuna operazione*

PUSH Pone un dato a 16 bit nello stack, decrementando automaticamente di 2 il puntatore SP
POP Preleva un dato a 16 bit dallo stack, incrementando automaticamente di 2 il puntatore SP

XCHG (eXCHanGe) *Scambio*
Scambia il contenuto dei due operandi
XLAT (translation) *Traslazione indice-valore*
Sostituisce il contenuto di AL, visto come indice in una tabella il cui indirizzo iniziale si suppone posto in BX, con l'elemento della tabella di indirizzo BX+AL. Risultato in AL.

Istruzioni riguardanti i flag:

CLC (CLear Carry) *Reset del flag di Carry*
STC (SeT Carry) *Set del flag di Carry*
CMC (CoMplement Carry) *Inversione del flag di Carry*

CLD (Clear Direction) *Reset del flag di Direzione*
STD (SeT Direction) *Set del flag di Direzione*
Le successive istruzioni di stringa decrementeranno i registri SI e/o DI coinvolti, anziché incrementarli.

CLI (CLear Interrupt) *Disabilitazione delle interruzioni mascherabili*
STI (SeT Interrupt) *Abilitazione del riconoscimento delle interruzioni mascherabili*

LAHF (Load AH from Flags) *Caricamento AH dai flag*
Il caricamento avviene come segue: flag di Segno = bit 7, flag di Zero = bit 6, flag Ausiliario[5] = bit 4, flag di Parità = bit 2, flag di Carry = bit 0; altri bit indefiniti.
SAHF (Store into AH Flags) *Memorizzazione in AH il registro dei flag* (dettagli come sopra)

PUSHF (PUSH Flags) Pone nello stack il contenuto del registro dei flag
POPF (POP Flags) Preleva il dato a 16 bit al top dello stack e lo pone nel registro dei flag

Istruzioni di salto:

JMP	(Jump) salto incondizionato
JA & JNBE	(Jump if Above / Jump if Not Below or Equal) salta se CF=0 oppure se ZF=0
JAE & JNB	(Jump if Above or Equal / Jump if Not Below) salta se CF=0
JB & JNAE	(Jump if Below / Jump if Not Above or Equal) salta se CF=1
JBE & JNA	(Jump if Below or Equal / Jump if Not Above) salta se CF=1 oppure se ZF=1
JC	(Jump if Carry) salta se CF=1
JCXZ	(Jump if CX is zero) salta se il contenuto del registro CX è 0
JE & JZ	(Jump if Zero / Jump if Equal) salta se ZF=1
JG & JNLE	(Jump if Greater / Jump if Not Less or Equal) salta se (SF xor OF or ZF)=0
JGE & JNL	(Jump if Greater or Equal / Jump if Not Less) salta se (SF xor OF)=0
JL & JNGE	(Jump if Less / Jump if Not Greater or Equal) salta se (SF xor OF)=1
JLE & JNG	(Jump if Less or Equal / Jump if Not Greater) salta se ((SF xor OF) or ZF)=1
JNC	(Jump if Not Carry) salta se CF=0
JNE & JNZ	(Jump if Not Zero / Jump if Not Equal) salta se ZF=0
JNO	(Jump if Not Overflow) salta se OF=0
JNS	(Jump if Not Sign) salta se SF=0
JNP & JPO	(Jump if Not Parity / Jump if Parity Odd) salta se PF=0
JO	(Jump if Overflow) salta se OF=1
JP & JPE	(Jump if Parity / Jump if Parity Even) salta se PF=1
JS	(Jump if Sign) salta se SF=1

[5] Funziona come il flag di carry ma con riferimento ai riporti/prestiti fra primo e secondo nibble di un byte, anziché fra primo e secondo byte di una word.

Istruzioni di rotazione e shift:

RCL (Rotate with Carry to the Left) *Rotazione a sinistra con Carry*
Ruota a sinistra i bit del suo operando di *n* posizioni, coinvolgendo nella rotazione anche il flag di Carry. Se *n*=1 può essere specificato direttamente come una costante, altrimenti occorre porre *n* in CL e usare tale registro per mantenere il conto. Esempio: RCL AX,CL

RCR (Rotate with Carry to the Right) *Rotazione a destra con Carry*
Come sopra ma verso destra.

ROL (ROtate Left) *Rotazione a sinistra (senza Carry)*
Come RCL ma senza coinvolgere il flag di Carry.

ROR (ROtate Right) *Rotazione a destra (senza Carry)*
Come RCR ma senza coinvolgere il flag di Carry.

SAL (Shift Arithmetic Left) *Shift aritmetico a sinistra*

SHL (SHift Left) *Shift logico a sinistra*
Introduce *n* zeri a destra shiftando il valore di *n* posizioni. Se *n*=1 può essere specificato direttamente, altrimenti occorre agire indirettamente tramite CL (vedi RCL). Il bit espulso a sinistra finisce nel flag di Carry.

SAR (Shift Arithmetic Right) *Shift aritmetico a destra*
Shifta il valore di *n* posizioni a destra, *replicando a sinistra il bit di segno*. Se *n*=1 può essere specificato direttamente, altrimenti occorre agire indirettamente tramite CL (vedi RCL). Il bit espulso a destra finisce nel flag di Carry.

SHR (SHift Right) *Shift logico a destra*
Analogo allo shift aritmetico a destra, *tranne per il fatto che introduce a sinistra degli zeri* anziché replicare il bit di segno.

Istruzioni di caricamento indirizzi:

LDS (Load using DS)
Carica un indirizzo a 32 bit dalla memoria in due registri, di cui uno è (implicitam.) DS

LES (Load using ES)
Carica un indirizzo a 32 bit dalla memoria in due registri, di cui uno è (implicitam.) ES

LEA (Load Effective Address)
Carica un indirizzo a 16 bit (offset) dalla memoria a un registro (a 16 bit)

Istruzioni di ciclo:

LOOP
Decrementa CX e cicla (saltando alla label indicata) se CX>0

LOOPE (Loop if Equal)

LOOPZ (Loop if Zero)
Decrementa CX e cicla se CX≠0 e inoltre il flag ZF=1

LOOPNE (Loop if Not Equal)

LOOPNZ (Loop if Not Zero)
Decrementa CX e cicla se CX≠0 e inoltre il flag ZF=0

Istruzioni riguardanti operazioni sulla stringhe:

CMPS (CoMPare Strings) *Confronto fra due stringhe*
Confronta due stringhe, indirizzate da SI la prima e da DI la seconda, entrambi incrementati/decrementati automaticamente a seconda del flag di direzione DF. Vedere anche istruzione REP.

MOVS (MOVe String) *Spostamento di una stringa*
Sposta una stringa di byte o di word da una posizione all'altra in memoria. Destinazione indirizzata da DI, sorgente da SI, entrambi incrementati/decrementati (a seconda del flag di direzione DF) automaticamente. Vedere anche istruzione REP.

SCAS (SCAn String) *Scansione di una stringa*
Effettua la scansione di una stringa (indirizzata da DI) confrontandola (nel senso di "sottrarla da") con AX o AL (per stringhe di word o byte risp.). Nessuno di questi valori viene alterato, ma i flag sono influenzati di conseguenza. DI è automaticamente incrementato/decrementato dopo l'operazione. Vedere anche istruzione REPNE & co.

STOS (STOre in String) *Memorizzazione in una stringa*
Trasferisce un operando contenuto in AX o AL in una stringa destinazione puntata da DI, che è automaticamente incrementato/decrementato a seconda del valore del flag di direzione.

Istruzioni di ripetizione per stringhe:

REP (REPeat) *prefisso di ripetizione per MOVS*
Ripete il trasferimento fino a fine stringa (CX=0)

REPE (REPeat while Equal) *prefisso di ripetizione per SCAS o CMPS*

REPZ (REPeat while Zero) *prefisso di ripetizione per SCAS o CMPS*
Ripete il confronto o la scansione (CMPS o SCAS) fino a fine stringa (CX=0) o fino a quando le due stringhe diventano diverse (ZF=0)

REPE (REPeat while Not Equal) *prefisso di ripetizione per SCAS o CMPS*

REPZ (REPeat while Not Zero) *prefisso di ripetizione per SCAS o CMPS*
Ripete il confronto o la scansione (CMPS o SCAS) fino a fine stringa (CX=0) o fino a quando le due stringhe diventano uguali (ZF=1)