Assembler 68000

di Marco Pesce

In questo numero affronteremo una lunga serie di istruzioni (quasi fossero un nemico da sconfiggere) e ci proponiamo di esaurire il discorso per quanto riguarda le possibilità del 68000 nudo e crudo. I passi successivi saranno quelli di rendere le nostre conoscenze «compatibili» con il sistema operativo di Amiga

Quello che segue sarà più o meno un lungo elenco in ordine alfabetico di istruzioni con sintassi, modi di indirizzamento possibili e altre specifiche, oltre ad un breve commento esplicativo, il tutto nel tentativo di realizzare una sorta di promemoria sufficientemente ordinato e pronto all'uso. Veniamo ai simboli utilizzati. Per quanto riguarda i flag occorre far riferimento alla seguente tabella:

— = inalterato
0= azzerato
1= settato
a= alterato tranne se «DESTINAZIONE» è un registro degli indirizzi
A= alterato (in funzione del valore)
?= non necessariamente alterato (dipende dal valore)
X= indefinito

Per quello che riguarda la sintassi di indirizzamento i simboli sono i seguenti:

An, Ax o Ay= registro degli indirizzi
Dn, Dx o Dy= registro dati
Rn, Rx o Ry= qualunque registro dati o indirizzi
(An)= registro degli indirizzi indiretto D
(An)= registro degli indirizzi indiretto con scostamento
-(An)= registro degli indirizzi indiretto con decremento precedente
(An)+ = registro degli indirizzi indiretto con incremento successivo
{dato}= dato immediato
CCR= byte più basso del registro di stato
SR= registro di stato

Con la prossima tabellina elenchiamo i simboli adottati per specificare la modalità di indirizzamento:

{ea}; uno qualunque dei modi di indirizzamento effettivo
{a/ea}; modo alterabile
{c/ea}; modo controllo
{d/ea}; modo dati
{ca/ea}; modo controllo alterabile
{da/ea}; modo dati alterabile
{ma/ea}; modo memoria alterabile

Come detto nella scorsa puntata, lo ripetiamo per chiarezza, occorre fare riferimento alla tabella pubblicata nella prima puntata per verificare se un dato modo di indirizzamento è possibile o meno con l'istruzione che ci interessa; ad esempio se nella sintassi possibile troviamo una sigla del tipo

CHK {d/ea},Dn

vorrà dire che l'istruzione «CHK» (ovvero l'istruzione di check) utilizza come operando sorgente una modalità da scegliere tra quelle del «modo dati» (vedi la tabella appena esposta) ovvero tra quelle che nella tabella della prima puntata hanno la colonna dell'indirizzamento «Dati» contrassegnata con una X; l'operando destinazione è uno qualunque dei registri dati (D1, D2 ecc.).

Da notare che alcune istruzioni hanno più di una sintassi; noi le elencheremo tutte. Andiamo quindi ad incominciare.

ABCD

Istruzione che effettua la somma di due byte decimali (in codice binario). Può essere eseguita tra i contenuti di due registri dati o tra quelli di due indirizzi di memoria specificati utilizzando la modalità di indirizzamento con «decremento precedente»

sintassi: ABCD Dx, Dy
ABCD -(Ax), -(Ay)
formato: byte
alterazione dei flag: N=x; Z=?; V=X; C=A; X=A

ADD

Ne esistono 5 varianti. La funzione è quella di sommare i due operandi; il risultato è come al solito depositato nell'operando destinazione.

La variante «ADDQ» utilizza il modo Quick. La variante «ADDX» aggiunge il valore del flag di estensione del segno al risultato

sintassi: ADD {ea},Dn
ADD Dn,{ma/ea}
ADDA {ea},An
ADDI #{dato},{da/ea}
ADDQ #{dato},{a/ea}
ADDX Dx,Dy
ADDX -(Ax), -(Ay)
formato: byte,word e longword
alterazione dei flag ADD: N=A; Z=A; V=A; C=A; X=A
alterazione dei flag ADDA: N=-; Z=-; V=-; C=-; X=-
alterazione dei flag ADDI: N=A; Z=A; V=A; C=A; X=A
alterazione dei flag ADDQ: N=a; Z=a; V=a; C=a; X=a
alterazione dei flag ADDX: N=A; Z=?; V=A; C=A; X=A

AND

Effettua l'AND logico tra sorgente e destinazione

sintassi: AND {d/ea},Dn
AND Dn,{ma/ea}
ANDI #{dato},{da/ea}
ANDI #{dato},CCR
ANDI #{dato},SR
formato AND: b, w, l
formato ANDI: b, w, l formato ANDI con CCR: byte
formato ANDI con SR: word
alterazione dei flag AND: N=A; Z=A; V=0; C=0; X=-
alterazione dei flag ANDI: N=A; Z=A; V=0; C=0; X=-
alterazione dei flag ANDI con CCR: N=?; Z=?; V=?; C=?; X=?
alterazione dei flag ANDI con SR: N=?; Z=?; V=?; C=?; X=?

ASL e ASR

Tali istruzioni permettono di effettuare lo spostamento dei bit del dato sorgente in una delle due direzioni (rispettivamente verso sinistra e verso destra). Il bit, rispettivamente, più significativo o meno significativo viene trasferito nel flag di Carry e in quello di estensione del segno. Il posto lasciato «vuoto» viene riempito con uno zero

sintassi: ASL Dx,Dy (oppure ASR)
ASL #{dato},Dy (oppure ASR)
ASL {ma/ea} (oppure ASR)
formato: b, w o l
alterazione dei flag: tutti «A»

Bcc

In questo gruppo di istruzioni sono incluse tutte quelle che effettuano dirmazioni (ovvero salti ad altri punti del programma) in funzione del risultato delle operazioni precedenti, quindi in funzione del registro di stato.

Ne esistono 16 e noi le elencheremo affiancandovi il flag che viene testato e la condizione che deve verificarsi in esso affinché ci sia la diramazione; a volte verrà indicata tra parentesi una descrizione più «umana» di quello che viene controllato.

La sintassi prevede l'utilizzo del mnemonico con affiancata la label destinazione (esempio: BEQ SUBROUT)

BCC se C=0
BCS se C=1 BEQ se Z=1
BGE se N=1 e V=0 oppure se N=0 e V=1 (se > o =)
BGT se N=1 e V=1 e Z=0 oppure se N=0 e V=0 e Z=0 (se maggiore di)
BHI se C=0 e Z=0
BLE se N=1 e V=0 oppure se N=0 e V=1 oppure se Z=1 (se < o =)
BLS se C=1 oppure Z=1
BLT se N=1 e V=0 oppure se N=0 e V=1 (se minore di)
BMI se N=1
BNE se Z=0
BPL se N=0
BVS se V=1
BVC se V=0 BRA salta in ogni caso (da usare in alternativa a JMP)
BSR salta a una subroutine (da usare in alternativa a JSR)

formato: b e w
alterazione dei flag: tutti inalterati

BCHG

Viene utilizzata per testare il valore di un bit.

Questo è specificato dall'operando sorgente e si trova nell'operando destinazione.

In quest'ultimo il bit viene complementato (invertito). Con l'istruzione BCLR invece tale bit viene azzerato, mentre con la BSET viene posto a 1. Infine con la BTST il bit viene lasciato inalterato.

Da notare che in ogni caso il bit viene copiato nel flag Z

sintassi: BCHG Dn,{da/ea}
BCHG #{dato},{da/ea}
BCLR Dn,{ea}
BCLR #{dato},{ea}
SET Dn,{ea}
BSET #{dato},{ea}
BTST Dn,{d/ea}
BTST #{dato},{d/ea}
formato: byte se DEST. è memoria e longword se è un registro dati
alterazione dei flag: solo Z con «A»

CHK

Viene utilizzata per confrontare l'operando sorgente con il byte basso di un registro dati. Se la destinazione è minore di zero o se è maggiore della sorgente viene attivata una «eccezione»; quest'ultima è un'operazione che prevede un complesso sistema di salti ed è normalmente utilizzata dal sistema operativo per situazioni particolari. Noi non tratteremo le eccezioni, ma per completezza elencheremo anche le istruzioni che implicano il loro funzionamento

sintassi: CHK {d/ea},Dn
formato: Word
alterazione dei flag: N=X; Z=X; V=X; C=X; X=-

CLR

Serve ad azzerare tutti i bit dell'operando specificato

sintassi: CLR {da/ea}
formato: B, W o L
alterazione dei flag: N=0; Z=1; V=0; C=0; X=-

CMP

È l'istruzione che permette il confronto con due dati e che è normalmente seguita da istruzioni di diramazione. L'operando sorgente viene sottratto all'operando destinazione, ma quest'ultimo resta inalterato; ovviamente vengono alterati i flag di condizione

sintassi: CMP {ea},Dn
CMPA {ea},An
CMPI __ {dato},{da/ea}
CMPM (Ax)+,(Ay)+
formato: tutte B, W o L tranne CMPA (solo W e L)
flag: tutti «A» tranne X che resta inalterato

DBcc

Istruzioni simili alle Bcc ma in più implicano un decremento dell'operando specificato. Se la condizione è verificata si effettua la diramazione all'indirizzo specificato dalla label. Anche in questo caso elenchiamo i vari tipi di condizioni e di flag coinvolti

sintassi: DBcc Dn,{label}
DBCC se C=0
DBCS se C=1
DBEQ se Z=1
DBF se falso
DBGE se N=1 e V=0 oppure N=0 e V=1
DBGT se N=1 e V=1 e Z=0 oppure N=0 V=0 e Z=0
DBHI se C=0 e Z=0
DBLE se N=1 e V=0 oppure N=0 e V=1 oppure Z=1
DBLS se C=1 e Z=1 DBLT se N=1 e V=0

oppure N=0 e V=1
DBMI se N=1
DBNE se Z=0
DBPL se N=0
DBVS se V=1
DBVC se V=0
DBT se vero
DBRA senza condizione
formato: Word
flag: nessuno alterato

DIVS

È la classica operazione di divisione. L'operando destinazione è a 32 bit e viene diviso con il sorgente che è a 16; il risultato va nell'operando destinazione. Il resto viene depositato nella parte alta del registro dati destinazione e il quoziente in quella bassa. Viene generata una «eccezione» se si tenta la divisione per zero. L'istruzione DIVU usa una aritmetica in valore assoluto.

sintassi: DIVS {d/ea},Dn
DIVU {d/ea},Dn
formato: W
flag: N=A; Z=A; V=A; C=0; X=-

EOR

Operazione logica di OR esclusivo
sintassi: EOR Dn,{da/ea}
EORI #{dato},{da/ea}
EORI #{dato},CCR
EORI #{dato},SR
formato: (rispettivamente) B, W, L; B, W, L; B; W
flag: N=A; Z=A; V=0; C=0;
X=- (tranne che con CCR e SR che invece sono tutti «?»)

EXG

Scambia fra loro i contenuti dei due registri specificati

sintassi: EXG Rx,Ry
formato: L
flag: tutti inalterati

EXT

Effettua l'estensione del segno
sintassi: EXT Dn
formato W e L
flag: N=A; Z=A; V=0; C=0; X=-

JMP

Istruzione di salto incondizionato alla locazione specificata
sintassi: JMP {c/ea}
flag: inalterati

JSR

Istruzione di salto incondizionato a subroutine con ritorno (se si incontra una RTS)

sintassi: JSR {c/ea}
flag: inalterati

LEA

Carica l'indirizzo effettivo nel registro indirizzi specificato

sintassi: LEA {c/ea},An
formato: L
flag: inalterati

LINK

È una istruzione per un utilizzo particolare dello stack pointer, ma non verrà trattata

LSL e LSR

Identiche alle istruzioni ASL e ASR ma il flag V viene azzerato

MOVE

È l'istruzione più usata e serve a trasferire valori dalla destinazione alla sorgente. Ne esistono ben 19 varianti

sintassi: MOVE {ea},{da/ea}
MOVEA {ea},An
MOVEM {serie di registri},{ca/ea} (movimento multiplo)
MOVEM {ca/ea},{serie di registri} (mov. multiplo)
MOVEM (An)+,{serie registri} (m.m.)
MOVEM {serie registri},-(An) (m.m.)
MOVEP Dx,d(Ay)
MOVEQ #{dato},Dn
MOVE {d/ea},CCR
MOVE {d/ea},SR
MOVE SR,{da/ea}
MOVE USP,An (USP= stack pointer utente)
MOVE An,USP
MOVEC Rn,(Cr) (istruzione privilegiata da usare quindi solo in stato supervisore)
MOVEP d(Ay),Dx (trasferisce i singoli byte a locazioni alternate, ovvero una sì e una no)
MOVEC (Cr),Rn (istr. priv.)
MOVES Dn, {ea} (istr. priv.)
MOVES {ea},Dn (istr. priv.)
MOVE CCR,{ea}

MULS

Questa istruzione permette di eseguire delle moltiplicazioni aritmetiche. Viene moltiplicato l'operando sorgente (a 16 bit) con la parola più bassa del destinazione e il risultato va ovviamente nel destinazione. L'istruzione MULU usa aritmetica in valore assoluto

sintassi: MULS {d/ea},Dn
MULU {d/a},Dn
formato: Word
flag MULS: N=A; Z=A; V=0; C=0; X=-
flag MULU: tutti inalterati

NBCD

Effettua una sottrazione da zero sull'operando destinazione e sul bit di estensione, adottando una aritmetica decimale. Il risultato è il complemento a 10 o a 9 del dato a seconda che, rispettivamente, il flag di estensione è settato o resettato

sintassi: NBCD {da/ea}
formato: Byte
flag: N=X; Z=?; V=X; C=A; X=A

NEG

Questa istruzione sottrae l'operando destinazione dal valore zero. L'istruzione NEGX sottrae anche il bit di estensione

sintassi: NEG {da/ea}
NEGX {da/ea}
formato: B, W o L
flag NEG: tutti «A» flag NEGX: tutti «A», tranne Z=?

NOP

È la classica istruzione nulla, ovvero quell'istruzione che non effettua alcuna operazione ma che può essere utile per riempire spazi vuoti, tuttavia è molto più utilizzata nel caso di programmazione con un monitor che nel caso si utilizzi un vero assemblatore

sintassi: NOP
flag: non alterati

NOT

Effettua l'inversione di tutti i bit dell'operando specificato (da 1 a 0 e viceversa)

sintassi: NOT {da/ea}
formato: B, W o L
flag: N=A; Z=A; V=0; C=0; X=-

OR

Effettua un OR logico tra sorgente e destinazione

sintassi: OR {d/ea},Dn
OR Dn,{ma/ea}
ORI #{dato},{da/ea}
ORI #{dato},CCR
ORI #{dato},SR (privilegiata)
formato: B, W o L (per CCR e SR, rispettivamente solo, B e W)
flag: N=A; Z=A; V=0; C=0; X=- (per CCR e SR tutti «?»)

PEA

È una istruzione che permette di calcolare un indirizzo effettivo e di depositarlo nell'area stack

sintassi: PEA {c/ea}
formato: Longword
flag: nessuna alterazione

RESET

Il reset è una istruzione privilegiata (si usa solo in stato supervisore, pena l'attivazione di una eccezione di TRAP). Essa resetta tutti i dispositivi esterni. Dall'utente non dovrebbe mai essere utilizzata

sintassi: RESET

ROL e ROR

Sono le classiche istruzioni di rotolamento, ovvero quelle che ruotano verso sinistra (ROL) o verso destra (ROR) l'intero contenuto dell'operando specificato, depositando il bit che «esce» nel flag CARRY e il contenuto di quest'ultimo nel bit rimasto vuoto. Le istruzioni ROXL e ROXR sono simili alle precedenti ma depositano il bit uscente nel flag di estensione. La sintassi vale sia per la ROL che per la ROR (basta sostituire alla L la R)

sintassi: ROL Dx,Dy
ROL #{dato},Dy
ROL {ma/ea}
ROXL Dx,Dy
ROXL #{dato},Dy
ROXL {ma/ea}
formato: B, W o L flag ROL (o ROR): N=A; Z=A; V=0; C=A; X=-
flag ROXL (o ROXR): N=A; Z=A; V=0; C=A; X=A

RTS

Sono comprese in questo gruppetto tre istruzioni che effettuano il ritorno da una subroutine (chiamata con una JSR o simili) ma ognuna con una modalità differente: la RTS ritorna nel modo «classico», la RTR ritorna con il ripristino del CCR mentre la RTE ritorna da una eccezione

sintassi: RTS
RTR
RTE
flag RTS: nessuna alterazione
flag RTR e RTE: tutti «A»

SBCD

Effettua la sottrazione in aritmetica decimale

sintassi: SBCD Dx,Dy
SBCD -(Ax),-(Ay)
formato: Byte
flag: N=X; Z=?; V=X; C=A; X=A

SCC e altre

In questo gruppo di istruzioni sono racchiuse tutte quelle che settano a $FF il contenuto dell'operando specificato se si verifica la condizione imposta dalla particolare variante. Ecco l'elenco completo con relativa condizione

sintassi generale: Scc {da/ea}
SCC se C=0
SCS se C=1
SEQ se Z=1 SF se falso
SGE se N=1 e V=0 o N=0 e V=1
SGT se N=1 e V=1 e Z=0 o N=0 e V=0 e Z=1
SHI se C=0 e Z=0
SLE se N=1 e V=0 o N=0 e V=1 o Z=1
SLS se C=1 o Z=1
SLT se N=1 e V=0 o N=0 e V=1
SMI se N=1 SNE se Z=0
SPL se N=0 ST se vero
SVS se V=1
SVC se V=0
formato: Byte
flag: non alterati

STOP

Istruzione privilegiata che ferma l'esecuzione prima di un trace o di un interruzione o anche di un reset. Non utilizzata dall'utente

sintassi: STOP #{dato}
flag: tutti «A»

SUB

Effettua una sottrazione binaria tra destinazione e sorgente (dest.-sorg.). La SUBQ sottrae velocemente mentre la SUBX include il flag di estensione

sintassi:
SUB {ea},Dn
SUB Dn,{ma/ea}
SUBA {ea},An
SUBI #{dato},{da/ea}
SUBQ #{dato},{a/ea}
SUBX Dx,Dy
SUBX -(Ax),-(Ay)
formato: B, W o L (tranne SUBA che è solo W o L)
flag: tutti «A» tranne per SUBA che sono tutti «-» e per SUBX che Z è «?»

SWAP

Scambia le parole di un registro dati tra di loro

sintassi: SWAP Dn
formato : W
flag: N=A; Z=A; V=0; C=0; X=-

TAS

Effettua il test del bit più significativo dell'operando; se quest'ultimo è a 1 setta il flag N, mentre se è a 0 setta il flag Z

sintassi: TAS {da/ea}
formato: Byte
flag: N=A; Z=A; V=0; C=0; X=-

TRAP

istruzione che attiva un'eccezione (non trattata)

TST

Effettua una comparazione con zero dell'operando e setta i relativi flag

sintassi: TST {da/ea}
formato: B, W o L
flag: N=A; Z=A; V=0; C=0; X=-

UNLK

È l'ultima (finalmente!) istruzione del 68000 e viene usata come funzione inversa della LINK. In ogni modo (come quest'ultima) non viene trattata

sintassi: UNLK An
flag: inalterati

È con questo abbiamo veramente concluso. Sicuramente molte delle istruzioni non verranno neppure prese in considerazione da alcuni di voi, o ce ne saranno altre delle quali ignorate persino l'utilità, ma questo è inevitabile. Il modo migliore per evitare di ricadere nei soliti schemi di programmazione, usufruendo di un limitato gruppo di istruzioni ed escludendo la maggior parte di quelle specifiche è quello di esercitarsi, realizzando piccoli programmi e poi cercando di sostituire, dove è possibile, istruzioni specifiche; se ne guadagnerà in semplicità di stesura e in velocità di esecuzione. Alla prossima.

MC