

Corso di Calcolatori Elettronici I
A.A. 2012-2013

Processore M68000:
ulteriori istruzioni

ing. Alessandro Cilaro

Accademia Aeronautica di Pozzuoli
Corso Pegaso V "GArn Elettronici"

Shift

- **LSL, LSR**

shift logico a sinistra/destra

Esempio: `LSL.W #2,D2`

- **ROL, ROR**

rotazione a sinistra/destra

Esempio: `ROL #1,D0`

- **ASL, ASR**

shift aritmetico (con propagazione del segno) a sinistra/destra

Esempio: `ASR.B #4,D6`

Test e Bit-Test

- **TST**

verifica se l'operando è negativo o zero, influenzando opportunamente il registro di stato

Esempio: `TST D2`

- **BTST**

Verifica se un determinato bit è 1

Esempio: `BTST #1,D0`

- **BCHG, BCLR, BSET**

testano un bit e, rispettivamente, lo invertono, lo pongono a **0**, lo pongono a **1**

Esempio: `BCLR #4,D6`

Operazioni di scambio

- **EXG**

Scambia il contenuto di due registri

Esempio: **EXG D3 ,D7**

- **SWAP**

Scambia la parte bassa e la parte alta di un registro

Esempio: **SWAP D0**

ADD/SUB “Quick”

- Spesso occorre effettuare incrementi/decrementi di piccole quantità
- Un’istruzione però può codificare valori immediati soltanto con 16 o 32 bit
 - ciò può implicare un certo spreco di memoria

- **ADDQ, SUBQ**

somma/sottrae un immediato codificabile su tre bit (valori da 1 a 8, escludendo lo 0, che non serve)

L’immediato è codificato direttamente nei 16 bit dell’istruzione

Esempio: `ADDQ #8, D2`

`SUBQ #7, D6`

Operazioni con immediati

- l'assemblatore sceglie la codifica migliore disponibile (ad esempio **ADDQ** invece che **ADD**)
- **ADDI, SUBI**
somma/sottrae un immediato ad un operando dest.
Esempio: `ADDI #9,D2`
- **ANDI, ORI, EORI, etc.**
applica le corrispondenti funzioni logiche
Esempio: `ANDI #$FE,(A0)`
- **CMPI**
confronta un immediato con un operando destinazione
Esempio: `CMPI #100,D6`

Moltiplicazione e divisione

- **MULU, DIVU**

moltiplica/divide due operandi considerandoli come unsigned

Esempio: `MULU (A2),D2`

- **MULS, DIVS**

moltiplica/divide due operandi come signed

Esempio: `DIVS D3,D2`

La moltiplicazione opera sempre su operandi a 16 bit, in modo che il risultato richieda al massimo 32 bit

Manipolazione indirizzi

- **MOVEA**

E' una **MOVE** specializzata nel trasferimento di indirizzi

Opera solo su registri **Ax**

A differenza della **MOVE** ordinaria, non altera il registro di stato

Esempio: `MOVEA.L #$8100,A7`

`MOVEA.L A6,A5`

- **ADDA**

Similmente, applica l'addizione su un registro indirizzo

Esempio: `ADDA.L #$10,A3`

Trasferimenti multipli: MOVEM

- **MOVEM**

trasferisce “in blocco” interi gruppi di registri

E' possibile scegliere un sottoinsieme qualsiasi dei registri generici **D_x/A_x**

Si utilizza in combinazione con una modalità di indirizzamento con autoaggiornamento

Esempio: **MOVEM.L D0-D4/A0/A2-A4, -(A7)**
MOVEM.L (A7)+, D0-D4/A0/A2-A4

Confronto memoria-memoria

- **CMPM**

E' l'unica variante della **CMP** che consente confronti con entrambi gli operandi in memoria

E' ammessa solo la modalità di indirizzamento indiretto con post-incremento

E' tipicamente usata per confrontare vettori, ovvero sequenze di locazioni di memoria tutte della stessa dimensione

Esempio: **CMPM (A4)+, (A3)+**

DBcc: test, decrement, and branch

- **DBcc**

Fintantoché la condizione **cc** rimane falsa, decrementa il registro **Dx** fornito all'istruzione, e se questo non era zero prima del decremento (ovvero se ora non vale -1) salta all'etichetta fornita all'istruzione. Negli altri casi, passa all'istruzione seguente.

Fornisce un modo sintetico per gestire i cicli, sostituendo con un'unica istruzione il decremento di un contatore e la verifica di una condizione, che normalmente richiedono diverse istruzioni separate. Supporta tutti i **cc** usati in **Bcc**. Inoltre, ammette anche le forme **DBF** e **DBT** (**F** = false, e **T** = true) per ignorare la condizione ed usare solo il registro di conteggio.

Esempio: `DBT D0, LOOP`

```
[Dn] ← [Dn] - 1
IF [Dn] >= 0
    [PC] ← [PC] + d
ELSE
    [PC] ← [PC] + 2
```

Istruzioni STOP, NOP

- **STOP**

Modifica il registro di stato e pone il processore in una condizione di attesa

Sfrutta il meccanismo delle interruzioni

Esempio: `STOP #0010000000010010`

- **NOP**

Non fa nulla!

Esempio: `NOP`

E' un'istruzione "vuota"

Può talvolta essere utile per il *debug* dei programmi, o per occupare parti dell'area codice con spazio vuoto

Esercizio: DBcc

File: programma016.a68

- Scrivere un programma che sommi gli **N** elementi di un vettore
- Assemblare ed eseguire il programma sul simulatore
- Sperimentare l'uso dell'istruzione **DBcc**

Esercizio: DBcc

File: programma016.a68

```
ORG      $8000
START   MOVE.L #(N-1),D1
        MOVEA.L #VET,A2
        CLR.L  D0
LOOP    ADD.W  (A2)+,D0
        DBRA  D1,LOOP
        MOVE.L D0,SUM
N       EQU   8
VET     DC.W  1,4,2,3,5,3,-1,2
SUM     DS.W  1
        END   START
```

versione con DBcc
(DBRA equivale a DBF: caso
particolare di DBcc con cc=FALSE)

```
ORG      $8000
START   MOVE.L #(N-1),D1
        MOVEA.L #VET,A2
        CLR.L  D0
LOOP    ADD.W  (A2)+,D0
        SUBQ  #1,D1
        BGE  LOOP
        MOVE.L D0,SUM
N       EQU   8
VET     DC.W  1,4,2,3,5,3,-1,2
SUM     DS.W  1
        END   START
```

versione senza DBcc

Esercizio: DBcc

File: programma016.a68

```
ORG      $8000
START    MOVE.L #(N-1),D1
         MOVEA.L #VET,A2
         CLR.L  D0
LOOP     ADD.W  (A2)+,D0
         DBRA  D1,LOOP
         MOVE.L D0,SUM
N        EQU   8
VET      DC.W  1,4,2,3,5,3,-1,2
SUM      DS.W  1
         END   START
```

← Inizializza il contatore **D1** con il valore (immediato) **7**. In questo modo, in virtù del funzionamento della **DBRA**, il ciclo sarà ripetuto per valori di **D1** pari a **7, 6, 5, 4, 3, 2, 1, 0** (compreso), ovvero **8** volte

← Decrementa **D1** e salta a **LOOP** fino a quando **D1** diventa **-1**

versione con DBcc
(*DBRA* equivale a *DBF*: caso particolare di *DBcc* con *cc=FALSE*)

Esercizio: prodotto scalare

File: programma017.a68

- Scrivere un programma che esegua il prodotto scalare tra due vettori di word, etichettati **A** e **B**
 - **A** e **B** allocati staticamente ed inizializzati con **DC**
- Assemblare il programma con ASIMTOOL ed eseguirlo sul simulatore ASIM

Esercizio: prodotto scalare

File: programma017.a68

```
ORG      $8000
START    MOVE.L  #A,A0
         MOVE.L  #B,A1
         MOVE.L  #N-1,D0
         CLR     D2
LOOP     MOVE   (A0)+,D1
         MULS   (A1)+,D1
         ADD    D1,D2
         DBRA   D0,LOOP
         MOVE   D2,C
DONE     JMP    DONE
N        EQU   $000A
ORG      $80B0
A        DC.W   1,0,3,1,8,4,0,3,0,-1
ORG      $80D0
B        DC.W   2,1,-3,0,1,-2,11,-3,10,0
C        DS.L   1
END      START
```

Inizializza i due puntatori A0 e A1 con i valori degli indirizzi dei due vettori

prelievo degli elementi dei due vettori nella stessa posizione tramite i puntatori A0 e A1, moltiplicazione, incremento dei due puntatori per farli puntare alla successiva coppia di elementi

Gestione del ciclo tramite il contatore D0

Sposta il risultato nella locazione C

Esercizio: stringhe

File: programma018.a68

- Scrivere un programma che:
 - Riconosca un token in una stringa
 - Ne memorizzi l'indirizzo in una locazione di memoria
- Assemblare ed eseguire il programma sul simulatore

Esercizio: stringhe

File: programma018.a68

```
ORG      $8000
START    MOVEA.L  #STRING, A0
         MOVE.B   #TOKEN, D0
LOOP     TST.B    (A0)
         BEQ      DONE
         CMP.B    (A0)+, D0
         BNE     LOOP
FOUND    SUBQ.L   #1, A0
DONE     MOVE.L   A0, TOKENA
         STOP    #$2700
ORG      $8100
TOKEN    EQU     ':'
STRING   DC.B    'QUI QUO:QUA', 0
TOKENA   DS.L    1
END      START
```

Usa **A0** per puntare ai caratteri della stringa

Verifica di non essere arrivato al terminatore di stringa. Questo accade quando il byte puntato dal registro **A0** contiene il carattere **0**. In questo caso esce dal ciclo

Confronta il byte puntato dal registro **A0** con il "token" da cercare, contenuto nel registro **D0**. Nel caso lo trovi, copia l'indirizzo nella locazione **TOKENA**

Per la stringa qui si adotta la convenzione di far terminare la sequenza dei caratteri con un valore speciale, che è **0**, il "terminatore". In questo modo, non è necessario conoscere a priori la lunghezza della stringa

Esercizio: EQU e DC

File: programma019.a68

- Riprendere il programma che sommava le word di un vettore di lunghezza **N** usando la **DBcc**
- La dimensione **N** del vettore era data attraverso una costante **EQU**
 - per cambiare la dimensione, occorre riscrivere e riassemblare il programma
- Riscrivere il programma in modo che la dimensione del vettore sia essa stessa una variabile in memoria (definita tramite **DC**)
 - è quindi possibile cambiare dinamicamente la dimensione del vettore che verrà effettivamente usato

Esercizio: EQU e DC

File: programma019.a68

```
ORG      $8000
START    MOVE.L #(N-1),D1
         MOVEA.L #VET,A2
         CLR.L  D0
LOOP     ADD.W  (A2)+,D0
         DBRA  D1,LOOP
         MOVE.L D0,SOMMA
N        EQU   8
VET      DC.W  1,4,2,3,5,3,-1,2
SOMMA    DS.L  1
END      START
```

versione con EQU

```
ORG      $8000
START    MOVE.L N,D1
         SUBQ  #1,D1
         MOVEA.L #VET,A2
         CLR.L  D0
LOOP     ADD.W  (A2)+,D0
         DBRA  D1,LOOP
         MOVE.L D0,SOMMA
N        DC.L  8
VET      DC.W  1,4,2,3,5,3,-1,2
SOMMA    DS.L  1
END      START
```

versione con DC

Confrontare i due listati generati dai due programmi. Nel programma che usa la **EQU**, il **PLC** non sarà incrementato in corrispondenza dell'etichetta **N** poiché essa non corrisponde ad una locazione fisicamente allocata in memoria