Corso di Calcolatori Elettronici I
A.A. 2010-2011

Strutture di controllo del flusso di esecuzione in assembler

Lezione 25

Prof. Roberto Canonico

Università degli Studi di Napoli Federico II
Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica (allievi A-DE+Q-Z)
Corso di Laurea in Ingegneria dell'Automazione

MC68000: Status Register

- Contiene:
  - I codici di condizione (CCR – *Condition Code Register* da 8 bit) :
    - oVerflow (V), Zero (Z), Negative (N), Carry (C), eXtend (X)
  - Altri bit di stato - Trace (T), Supervisor (S)
  - La interrupt mask (8 livelli)
- I bit 5, 6, 7, 11, 12, e 14 sono riservati per espansioni future

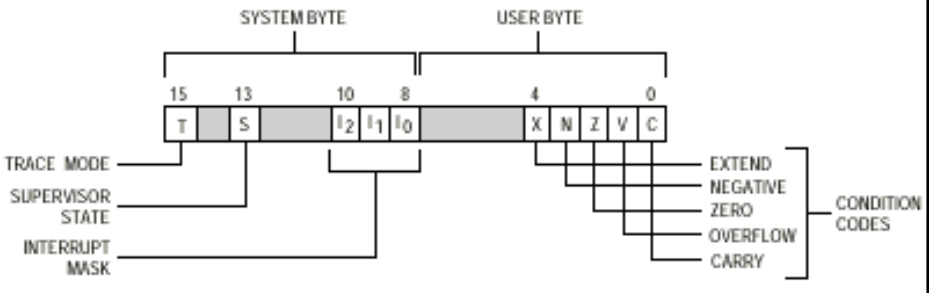

Figure 2-4. Status Register

L'istruzione Compare

Operation: [destination] - [source]
Syntax: `CMP <ea>,Dn`
Sample syntax: `CMP (A6),D2`
Attributes: Size = byte, word, longword
Description:
Subtract the source operand from the destination operand and set the condition codes accordingly. The destination must be a data register. The destination is not modified by this instruction.
Condition codes:

```
X N Z V C
_ * * * *
```

Compare memory with memory

Operation: [destination] - [source]
Syntax: `CMPM (Ay)+,(Ax)+`
Attributes: Size = byte, word, longword

Description:
Subtract the source operand from the destination operand and set the condition codes accordingly. The destination is not modified by this instruction. The only permitted addressing mode is address register indirect with post-incrementing for both source and destination operands.
Application:
Used to compare the contents of two blocks of memory.

Condition codes:

```
X N Z V C
_ * * * *
```

L'istruzione Branch on condition

Operation: IF cc = 1 THEN [PC] ← [PC] + d
Syntax: `Bcc <label>`
Attributes: Bcc takes an 8-bit or 16-bit offset (displacement)

Description:
If the specified logical condition is met, program execution continues at location [PC] + displacement, d.
The displacement is a two's complement value.

L'istruzione Branch on condition

Bcc **Branch on condition cc**

Operation: `If cc = 1 THEN [PC] ← [PC] + d`

Syntax: `Bcc <label>`

Sample syntax:
```
BEQ Loop_4
BVC *+8
```

Attributes: `BEQ` takes an 8-bit or a 16-bit offset (i.e., displacement).

Description: If the specified logical condition is met, program execution continues at location [PC] + displacement, d. The displacement is a two's complement value. The value in the PC corresponds to the current location plus two. The range of the branch is -126 to +128 bytes with an 8-bit offset, and -32K to +32K bytes with a 16-bit offset. A short branch to the next instruction is impossible, since the branch code 0 indicates a long branch with a 16-bit offset. The assembly language form `BCC *+8` means branch to the point eight bytes from the current PC if the carry bit is clear.

BCC	branch on carry clear	$\overline{C}$
BCS	branch on carry set	C
BEQ	branch on equal	Z
BGE	branch on greater than or equal	$N.V + \overline{N}.\overline{V}$
BGT	branch on greater than	$N.V.\overline{Z} + \overline{N}.\overline{V}.\overline{Z}$
BHI	branch on higher than	$\overline{C}.\overline{Z}$
BLE	branch on less than or equal	$Z + N.\overline{V} + \overline{N}.V$
BLS	branch on lower than or same	$C + Z$
BLT	branch on less than	$N.\overline{V} + \overline{N}.V$
BMI	branch on minus (i.e., negative)	N
BNE	branch on not equal	$\overline{Z}$
BPL	branch on plus (i.e., positive)	$\overline{N}$
BVC	branch on overflow clear	$\overline{V}$
BVS	branch on overflow set	V

Condizioni cc da cui dipende Bcc

Single bit	
BCS branch on carry set	C = 1
BCC branch on carry clear	C = 0
BVS branch on overflow set	V = 1
BVC branch on overflow clear	V = 0
BEQ branch on equal (zero)	Z = 1
BNE branch on not equal	Z = 0
BMI branch on minus (i.e., negative)	N = 1
BPL branch on plus (i.e., positive)	N = 0
Signed	
BLT branch on less than (zero)	N ⊕ V = 1
BGE branch on greater than or equal	N ⊕ V = 0
BLE branch on less than or equal	(N ⊕ V) + Z = 1
BGT branch on greater than	(N ⊕ V) + Z = 0
Unsigned	
BLS branch on lower than or same	C + Z = 1
BHI branch on higher than	C + Z = 0

Condizioni cc da cui dipende Bcc

- Da utilizzare per numeri unsigned:

BHS	BCC	branch on higher than or same
BHI		branch on higher than
BLS		branch on lower than or same
BLO	BCS	branch on less than

- Da utilizzare per numeri signed

BGE	branch on greater than or equal
BGT	branch on greater than
BLE	branch on lower than or equal
BLT	branch on less than

Esempio esito del confronto

$FF è maggiore di $10 se i numeri sono interpretati come unsigned, in quanto 255 è maggiore di 16

Tuttavia se i numeri sono interpretati come signed, $FF è minore di $10, in quanto -1 è minore di 16.

IL PROCESSORE NON TIENE CONTO DEL TIPO DI RAPPRESENTAZIONE QUANDO SETTA I FLAG DI CONDIZIONE

Istruzioni di selezione in assembler: if-then

Linguaggio di alto livello:

```
if (espressione)
      istruzione
istruzione_successiva
```

NOTA: istruzione può essere un *compound statement*

Linguaggio assembler (processore MC 68000):

```
            B(NOT condizione) labelA
            istruzione
            ...
   labelA   istruzione_successiva
```

Esempio:

```
if (D0 == 5)
      D1++;
D2 = D0;
```

```
            CMPI.L #5,D0
            BNE    SKIP
            ADDQ.L #1,D1
      SKIP  MOVE.L D0,D2
```

Istruzioni di selezione in assembler: if-then-else

Linguaggio di alto livello:

```
      if (espressione)
            istruzione1
      else
            istruzione2
      istruzione_successiva
```

Linguaggio assembler (processore MC 68000):

```
            B(NOT condizione) labelA
            istruzione1
            ...
            BRA labelB
labelA      istruzione2
            ...
labelB      istruzione_successiva
```

Strutture iterative in assembler: do-while

Linguaggio di alto livello:

```
      do
            istruzione
      while (condizione == TRUE);
      istruzione_successiva
```

Linguaggio assembler (processore MC 68000):

```
labelA      istruzione
            ...
            Bcc labelA
            istruzione_successiva
```

Esempio: calcola 3^N (N>0)

```
D0 = 1; D1 = 1;
do {
      D0 = D0 * 3;
      D1++;
} while (D1 <= N);
```

```
            MOVE.B  #N,D2
            MOVE.B  #1,D1
            MOVE.W  #1,D0
   LOOP     MULU.W  #3,D0
            ADDQ.B  #1,D1
            CMP.B   D2,D1
            BLE     LOOP
```

Strutture iterative in assembler: while

Linguaggio di alto livello:

```
        while (condizione == TRUE)
                istruzione;
        istruzione_successiva
```

Linguaggio assembler (processore MC 68000):

```
                BRA labelB
labelA          istruzione
                ...
labelB          Bcc labelA
                istruzione_successiva
```

Esempio: calcola 3^N (N>=0)

```
D0 = 1; D1 = 1;
while (D1 <= N) {
        D0 = D0 * 3;
        D1++;
};
```

```
        MOVE.B #N,D2
               MOVE.B #1,D1
               MOVE.W #1,D0
        BRA     TEST
        LOOP    MULU.W #3,D0
                ADDQ.B #1,D1
        TEST    CMP.B  D2,D1
                BLE    LOOP
```

DBcc: Test condition, decrement, and branch

Operazione: IF (*cc* false) THEN
[Dn] ← [Dn] – 1
IF [Dn] = -1 THEN [PC] ← [PC] + 2
ELSE [PC] ← [PC] + d
ELSE [PC] ← [PC] + 2

Sintassi: DBcc Dn,<label>

Attributi: Size = word

Descrizione:

Fintantoché la condizione *cc* rimane falsa, decrementa il registro D*n*, e se questo non era zero prima del decremento (ovvero se non vale -1) salta all'istruzione a distanza *d*. Negli altri casi, passa all'istruzione seguente.

Fornisce un modo sintetico per gestire i cicli, sostituendo con un'unica istruzione il decremento di un registro di conteggio e la verifica di una condizione normalmente fatti con 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

NOTA: DBRA = DBF.

Cicli for mediante DBRA

NOTA: DBRA equivale a DBF - caso particolare di DBcc con cc=FALSE

Esempio:

```
       MOVE.L     #N,D1
       SUBQ.L     #1,D1
       MOVEA.L    #NUM,A2
       CLR.L      D0
LOOP   ADD.W      (A2)+,D0
       DBRA       D1,LOOP
       MOVE.L     D0,SOMMA
```

equivale a:

```
       MOVE.L     #N,D1
       SUBQ.L     #1,D1
       MOVEA.L    #NUM,A2
       CLR.L      D0
LOOP   ADD.W      (A2)+,D0
       SUBQ       #1,D1
       BGE        LOOP
       MOVE.L     D0,SOMMA
```