La programmazione Assembly

Corso di Calcolatori Elettronici I

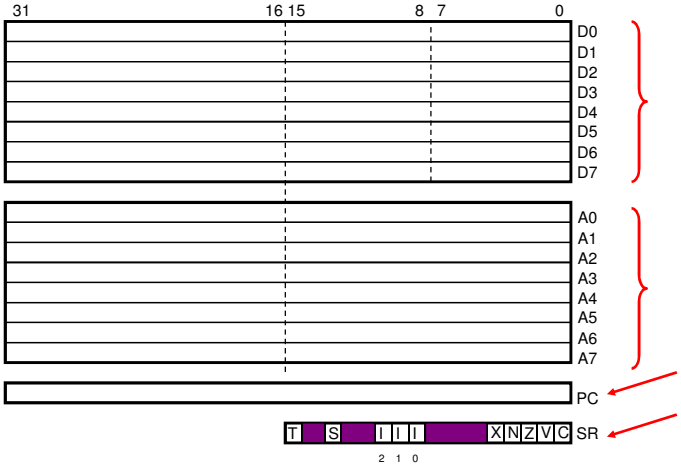
Dipartimento di Informatica e Sistemistica
Università degli Studi di Napoli "Federico II"

Sommario

- Programmazione del processore MC68000
  - Modello di programmazione
  - Alcuni codici e direttive
  - Esempi ed esercizi
  - Costrutti per il controllo di flusso (iterazione e selezione)

- **NOTA**: per tutte le istruzioni presenti nelle dispense e nelle slide fare sempre riferimento al manuale MC68K

Modello di programmazione del MC68000



Registri dati

utilizzabili come dati di 32 bit (Long Word), 16 bit (Word) oppure 8 bit (byte)

Registri indirizzo

utilizzabili per indirizzi di 32 bit (Long Word) oppure 16 bit (Word)

Program counter

Registro di stato

Registro di stato (Status register, SR)

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

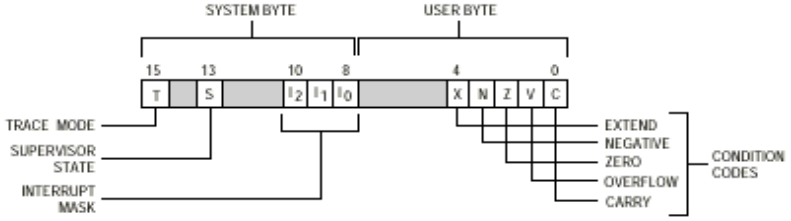


Figure 2-4. Status Register

Registro di stato: CCR

Codici di condizione (bit singoli o *flag*): informazioni riguardo i risultati prodotti da varie operazioni

Il 68000 prevede 5 codici di condizione:

N (Negative)	posto a 1 se il risultato è negativo
Z (Zero)	posto a 1 se il risultato è zero
V (oVerflow)	posto a 1 se si verifica un overflow
C (Carry)	posto a 1 se l'operazione genera un riporto
X (eXtension)	posto a 1 da istruzioni aritmetiche

I ***codici di condizione*** sono modificati:

- ***implicitamente*** in seguito all'esecuzione di ***operazioni aritmetiche, logiche o di altro tipo***;
- ***esplicitamente*** mediante **apposite istruzioni.**

Registro di stato: CCR

ATTENZIONE: Istruzioni, anche non aritmetiche, alterano i flag:

MOVE #0,CCR mette a zero tutti i flag

MOVE #-9,D0 mette a 1 il flag N

REGOLA: ***consultare il manuale per vedere se e come ciascuna istruzione altera i flag***

Pseudo-operatori

- NON sono istruzioni eseguite dal processore
  - Sono direttive che regolano il processo di traduzione del programma assembler in programma eseguibile
- Lo pseudo-operatore ORG
  - Viene usato per inizializzare il Program Location Counter (PLC), ovvero per indicare a quale indirizzo sarà posta la successiva sezione di codice o dati
  - **Esempio:** ORG $8100
- Lo pseudo-operatore END
  - Viene usato per terminare il processo di assemblaggio ed impostare l'entry point (prima istruzione da eseguire) nel programma
  - **Esempio:** END TARGETLAB

Pseudo-operatori

- Lo pseudo-operatore DS
  - Viene usato per incrementare il Program Location Counter (PLC), in modo da riservare spazio di memoria per una variabile
  - **Esempio:** LABEL DS.W NUMSKIPS
- Lo pseudo-operatore DC
  - Viene usato per inizializzare il valore di una variabile
  - **Esempio:** LABEL DC.W VALUE
- Lo pseudo-operatore EQU
  - Viene usato per definire una costante usata nel sorgente assembler
  - **Esempio:** LABEL EQU VALUE

Etichette (label)

- Sono stringhe di testo arbitrarie (opzionali) anteposte ad una istruzione o ad un dato all'interno del programma assembler
- Servono a riferirsi al particolare indirizzo che contiene quella istruzione o dato
  - usati per gestire i salti
  - usati per gestire variabili (manipolate nel programma assembler attraverso le loro etichette in maniera simile alle variabili di un linguaggio di programmazione di alto livello)
- Ad esempio:
  - ALOOP è un'etichetta usata per riferisti all'istruzione MOVE, SUM e CNT sono etichette usate per gestire una variabile, mentre IVAL è una costante

```
ALOOP   MOVE.W      D0,CNT
        ADD.W       SUM,D0
   …  … …  …
CNT     DS.W        1
SUM     DS.W        1
IVAL    EQU         17
   … … … … … …
```


Istruzione MOVE

Operazione: [source] -> [destination]
Sintassi: MOVE <ea>,<ea>
ad esempio: MOVE D2,D1
Attributi: Size = byte, word, longword

Descrizione:
Sposta il contenuto dell'operando sorgente nella locazione del secondo. L'operando sorgente non è modificato. Influenza i flag di stato come segue:

X	N	Z	V	C
-	*	*	0	0

- '-' : il flag non viene influenzato
- '*' : il flag viene influenzato a seconda del risultato dell'operazione
- '0' o '1': il flag viene viene sempre posto pari a 0 o 1, rispettivamente

Istruzione ADD

Operazione: [source] + [destination] -> [destination]
Sintassi: ADD <ea>,Dn (*Dn: uno degli 8 registri dati D*)
ADD Dn,<ea>
ad esempio: ADD D2,D1
Attributi: Size = byte, word, longword

Descrizione:
Somma l'operando sorgente con la destinazione, e memorizza il risultato nella destinazione. Può influenzare tutti i flag di stato:

X N Z V C
* * * * *

NOTA: esiste un'istruzione simile per la sottrazione (SUB)

Istruzione ADDQ: "add quick"

Operation: [destination] + <literal> -> [destination]
Syntax: ADDQ #<data>,<ea>
Attributes: Size = byte, word, longword

Descrizione:
Somma l'immediato <literal> alla destinazione.
L'immediato può andare da 1 a 8.
L'operazione può influenzare tutti i flag di stato:

X N Z V C
* * * * *

A differenza della ADD, l'immediato "corto" è codificato all'interno dei 16 bit dell'istruzione e non separatamente in una successiva parola

NOTA: esiste un'istruzione simile per la sottrazione (SUBQ)

SUBQ Subtract quick

Operation: [destination] ← [destination] - <literal>

Syntax: `SUBQ #<data>,<ea>`

Attributes: Size = byte, word, longword

Description:

Subtract the immediate data from the destination operand.

The immediate data must be in the range 1 to 8.

Word and longword operations on address registers do not affect condition codes. A word operation on an address register affects the entire 32-bit address.

Condition codes:

X	N	Z	V	C
*	*	*	*	*

Istruzione CLR: Clear

Operazione: 0 -> [destination]
Sintassi: CLR <ea>
ad esempio: CLR D2
Attributi: Size = byte, word, longword

Descrizione:
Pone il valore 0 all'indirizzo <ea>. Influenza i flag di stato come segue:

X	N	Z	V	C
-	0	1	0	0

CMP: Compare

Operazione: [destination] - [source]

Sintassi: CMP <ea>,Dn

ad esempio: CMP (Test,A6,D3.W),D2

Attributi: Size = byte, word, longword

Descrizione:

Sottrae l'operando sorgente (source) dall'operando destinazione (destination). **NON** memorizza il risultato: l'unico effetto è influenzare i seguenti flag di stato sulla base del risultato:

X N Z V C
\- * * * *

Ad esempio, se i due operandi sono uguali, la sottrazione dà come risultato zero, ed il flag Z viene posto ad 1. Il flag potrà poi essere letto da un'istruzione successiva (ad esempio, una BEQ = *salta se Z è 1*), per decidere quali istruzioni eseguire come conseguenza del confronto

CMPM: Compare memory with memory

Operazione: [destination] - [source]

Sintassi: CMPM (Ay)+,(Ax)+

Attributi: Size = byte, word, longword

Descrizione:

Sottrae l'operando sorgente (source) dall'operando destinazione (destination). **NON** memorizza il risultato, ma semplicemente influenza i seguenti flag di stato sulla base del risultato:

X N Z V C
\- * * * *

L'unico modo di indirizzamento ammissibile è il post-incremento.

L'istruzione è usata per confrontare due array bi byte, word, o long

Istruzioni di Salto

SALTO: ***assegna al Program Counter un determinato valore***

PC = A

Salto condizionato: ***avviene solo se una data condizione logica è verificata***

Salto incondizionato: ***avviene comunque***

- ***Assoluto*** - l'operando **O** dell'istruzione esprime direttamente il valore dell'indirizzo cui saltare: **PC=O**
- ***Relativo***: l'operando **O** esprime l'incremento da apportare a ***PC***:

PC = PC + O

B*cc*: Branch on condition *cc*

Operazione: IF **cc = 1** THEN **[PC] ← [PC] + d**

Sintassi: B*cc* <label>

Attributi: uno spiazzamento (*offset*, o *displacement*) di 8-bit o 16-bit.

Descrizione:

- Se la condizione logica specificata dalla condizione *cc* è verificata, non viene eseguita la prossima istruzione, bensì quella avente indirizzo PC+spiazzamento.
- Lo spiazzamento è in complementi a due e può pertanto essere anche negativo (salto "indietro").
- *cc* indica una delle differenti condizioni che è possibile riscontarare nel registro di stato come effetto delle istruzioni precedenti (*vedi seguito*), ad esempio una *compare* (CMP).

B*cc*: possibili condizioni *cc*

- **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

C, V, Z, N sono flag del registro di stato

C = flag di carry

V = flag di overflow

Z = flag di zero

N = flag di negative

Significato delle condizioni nella BRANCH

- Se i numeri sono interpretati come **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

- Se i numeri sono interpretati come **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

Differenza *signed* / *unsigned* nel confronto

Ad esempio:

- $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. Sta al programmatore conoscere il formato dei dati manipolati ed interpretare correttamente gli effetti sui flag di stato.**

Istruzioni di Salto Incondizionato

Istruzioni di salto incondizionato

***assolute* (JMP A)**

JMP $7800:	salto diretto all'indirizzo 7800_{16}
JMP ciclo:	salto diretto all'indirizzo associato all'etichetta "ciclo"
JMP (A3)	salto indiretto all'indirizzo contenuto in A3

***relative* (BR*anch Always*)**

BRA A **salta all'istruzione di indirizzo PC+ *disp***

***disp* è calcolato dall'assemblatore come differenza tra il valore corrente di PC e l'indirizzo associato all'operando A**

Istruzione di salto incondizionato assoluto JMP

Operazione: [PC] <- destination
Sintassi: JMP <ea>
ad esempio: JMP loop3
Attributi: --

Descrizione:
Scrive il valore dell'operando destinazione nel Program Counter: in altre parole, realizza un salto incondizionato. Prevede differenti modi di indirizzamento per specificare l'operando destinazione (a differenza della Branch che prevede solo indirizzamento relativo). Non influenza i flag di stato:

X	N	Z	V	C
-	-	-	-	-

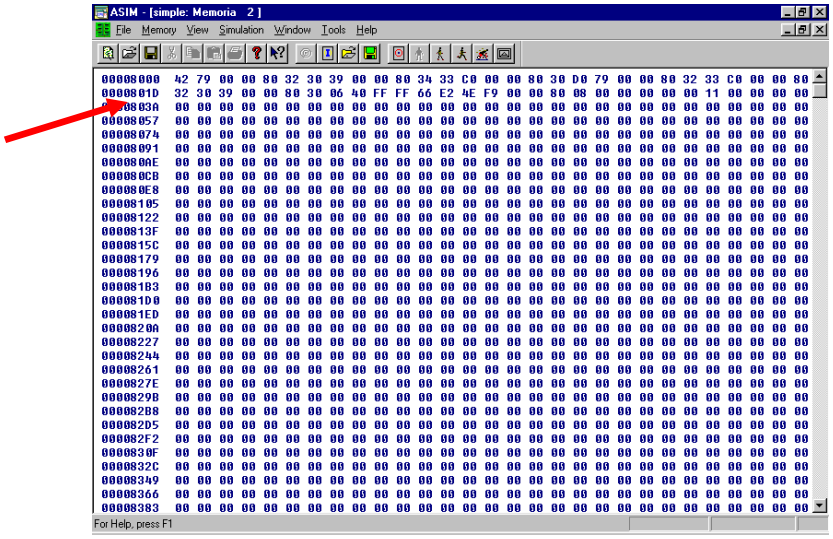
Esempio di programma assembler 68000

```
PLC       contenuto          label   opcode    operands   comments
00000000                     1   * Programma per sommare i primi 17 interi
00000000                     2   *
00008000                     3           ORG       $8000
00008000  4279 00008032      4   START   CLR.W     SUM
00008006  3039 00008034      5           MOVE.W    ICNT,D0
0000800C  33C0 00008030      6   ALOOP   MOVE.W    D0,CNT
00008012  D079 00008032      7           ADD.W     SUM,D0
00008018  33C0 00008032      8           MOVE.W    D0,SUM
0000801E  3039 00008030      9           MOVE.W    CNT,D0
00008024  0640 FFFF          10          ADD.W     #-1,D0
00008028  66E2               11          BNE       ALOOP
0000802A  4EF9 00008000      12          JMP       SYSA
00008030  =00008000          13  SYSA    EQU       $8000
00008030                     14  CNT     DS.W      1
00008032                     15  SUM     DS.W      1
00008034  =00000011          16  IVAL    EQU       17
00008034  0011               17  ICNT    DC.W      IVAL
```

Symbol Table

ALOOP	800C	CNT	8030	IVAL	0011
START	8000	SUM	8032	ICNT	8034

Programma assemblato in memoria



Esempio - Moltiplicazione di due interi

```
*  Programma per moltiplicare MCND e MPY
*
       ORG       $8000
*
MULT   CLR.W     D0          D0 accumula il risultato
       MOVE.W    MPY,D1      D1 e' il contatatore di ciclo
       BEQ       DONE        Se il contatore e' zero e' finito
LOOP   ADD.W     MCND,D0     Aggiunge MCND al prodotto parziale
       ADD.W     #-1,D1      Decrementa il contatore
       BNE       LOOP        Se il contatore non si è azz. ripete il giro
DONE   MOVE.W    D0,PROD     Salva il risultato

PROD   DS.W      1           Riserva spazio di memoria per PROD
MPY    DC.W      3           Definisce il valore di MPY
MCND   DC.W      4           Definisce il valore di MCND
       END       MULT        Fine ass., salto a entry point
```


Esempio - Moltiplicazione di due interi

- Eseguire il programma sul simulatore e sperimentare:
  - L'effetto di DC e la rappresentazione esadecimale in memoria
  - L'effetto dell'istruzione CLR su registro
  - L'effetto dell'istruzione MOVE da memoria a registro
  - L'effetto dell'istruzione BEQ sul PC
  - L'effetto dell'istruzione ADD tra memoria e registro
  - L'effetto dell'istruzione ADD tra immediato e registro
  - L'effetto dell'istruzione BNE sul PC
  - L'effetto dell'istruzione MOVE da registro a memoria e la rappresentazione esadecimale in memoria

Esempio - Moltiplicazione di due interi

- Nell'esempio precedente, effettuare le seguenti sostituzioni ed osservarne gli effetti

```
DONE   MOVE.W    D0,PROD    Salva il risultato
PROD   DS.W      1          Riserva spazio di memoria per PROD
```

```
DONE   MOVE.L    D0,PROD    Salva il risultato
PROD   DS.L      1          Riserva spazio di memoria per PROD
```

L= long (32 bit) W= word (16 bit) B= byte (8 bit)

Esempio - Somma di n interi

```
START       CLR.W       SUM
            MOVE.W      ICNT,D0
ALOOP       MOVE.W      D0,CNT
            ADD.W       SUM,D0
            MOVE.W      D0,SUM
            MOVE.W      CNT,D0
            ADD.W       #-1,D0
            BNE         ALOOP
            JMP         SYSA
SYSA        EQU         $8000
CNT         DS.W        1
SUM         DS.W        1
IVAL        EQU         17
ICNT        DC.W        IVAL
```


Esempio - Somma di n interi

- Assemblare ed eseguire il programma sul simulatore
- Sperimentare:
  - L'effetto dell'istruzione CLR in memoria
  - L'effetto dell'istruzione MOVE da memoria a registro
  - L'effetto dell'istruzione ADD tra memoria e registro
  - L'effetto delle varie istruzioni sui codici di condizione
  - L'effetto dell'istruzione BNE sul PC
  - L'effetto dell'istruzione JMP sul PC

DBcc: Test condition, decrement, and branch

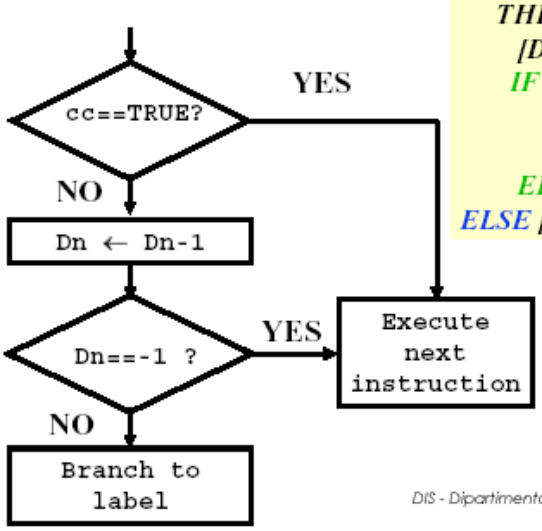
Operazione: IF (*cc* false) THEN
[Dn] ← [Dn] – 1
IF [Dn] = -1 THEN [PC] ← [PC] + 2
ELSE [PC] ← [PC] + d
ELSE [PC] ← [PC] + 2

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.

DBcc: Test condition, decrement, and branch




```
IF (condition false)
   THEN
     [Dn]←[Dn]–1 {decrement loop counter}
    IF [Dn] = -1
       THEN
         [PC]←[PC]+2{goto next instruction}
     ELSE [PC] ← [PC] + d {take branch}
ELSE [PC]←[PC] + 2 {goto next instruction}
```


LEA: Load Effective Address

Operazione: [An] ← <ea>
Sintassi: LEA <ea>,An
Esempio: LEA table,A3
Attributi: Size = longword

Descrizione:
Calcola l'indirizzo effettivo (<*ea*>) del primo operando, generalmente espresso in forma simbolica, e lo pone nel registro indirizzo specificato dal suo secondo operando. Non influenza i flag di stato:

X	N	Z	V	C
-	-	-	-	-

Esercitazione: prodotto scalare tra due vettori

- Scrivere un programma che esegua il prodotto scalare tra due vettori di interi
- Assemblare ed eseguire il programma sul simulatore

Soluzione – scalprod.a68

```
        ORG     $8000
START   MOVE.L  #A,A0
        MOVE.L  #B,A1
        MOVE.L  #N,D0
        CLR     D2
        SUBQ    #1,D0
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    1,1,1,1,1,1,1,1,1,1
        ORG     $80D0
B       DC.W    1,1,1,1,1,1,1,1,1,1
C       DS.L    1
```

Mappa della memoria:

START = 8000 ***CODICE***

A = 80B0 ***VETTORE A***

B = 80D0 ***VETTORE B***

Esercitazione

- 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

Soluzione – token.a68

```
          ORG       $8000
START     MOVEA.L   #STRING,A0
          MOVE.B    #TOKEN,D0
LOOP      CMP.B     (A0)+,D0
          BNE       LOOP
FOUND     SUBQ.L    #1,A0
          MOVE.L    A0,TOKENA

          ORG       $8100
TOKEN     EQU       ':'
STRING    DC.B      'QUI QUO:QUA'
TOKENA    DS.L      1
```


Soluzione – token.a68

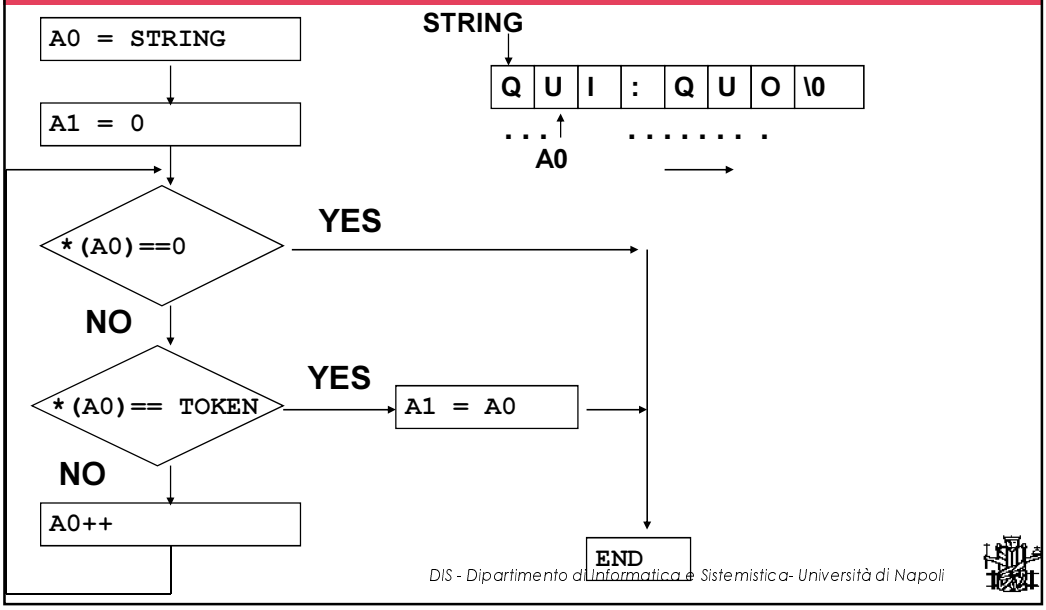
```
          ORG       $8000
START     MOVEA.L   #STRING,A0
          MOVE.B    #TOKEN,D0
LOOP      CMP.B     (A0)+,D0
          BNE       LOOP
FOUND     SUBQ.L    #1,A0
          MOVE.L    A0,TOKENA

          ORG       $8100
TOKEN     EQU       ':'
STRING    DC.B      'QUI QUO:QUA'
TOKENA    DS.L      1
```

ATTENZIONE:

In questo semplice programma, si assume che la stringa contenga sempre il token. Se non è così, il ciclo continua all'infinito. In un programma realmente utilizzabile occorrerebbe anche controllare se si è arrivati alla fine della stringa prima di ogni iterazione

Ricerca token in una stringa - Risultato in A1 (*)



ESEMPIO 3: Ricerca token in una stringa -Risultato in A1 (*)

```
          ORG       $8000
START     MOVEA.L #STRING,A0
          MOVE.B  #TOKEN,D0
LOOP      CMP.B    #TAPPO,(A0)
          BEQ     DONE
          CMP.B    (A0)+,D0
          BNE       LOOP
          SUBQ.L    #1,A0
          MOVEA.L  A0,A1
DONE      JMP      DONE

          ORG       $8100
STRING    DC.B      'QUI QUO:QUA',0
TOKEN     EQU       ':'
TAPPO     EQU         $0
          END       START
```

Istruzioni di Controllo

Istruzioni di selezione in assembler – 1/ 3

Linguaggio di alto livello:

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

NOTA: istruzione può essere un *compound statement*

Linguaggio assembler (processore MC 68000):

```
            B(NOT espressione) 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
```

DIS - Dipartimento di Informatica e Sistemistica- Università di Napoli

Istruzioni di selezione in assembler – 2/ 3

Linguaggio di alto livello:

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

Linguaggio assembler (processore MC 68000):

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


Istruzioni di selezione in assembler – 3/ 3

Linguaggio di alto livello:

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

      istruzione_successiva
```

Esempio:

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

Linguaggio assembler (processore MC 68000):

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

Esempio:

```
      CMP.L       #5,D0
      BNE         L1
      ADDQ.L      #1,D1
      BRA         L2
L1    SUBQ.L      #1,D1
L2    MOVE.L      D2,D0
```


Strutture iterative in assembler - 1/ 2

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
        BLS    LOOP
```


Strutture iterative in assembler - 2/ 2

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
        BLS    LOOP
```


Decrement and Branch always (DBRA)

equivale a:

```
        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
```

```
        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
```