

Università di Napoli Federico II – Scuola Politecnica e delle Scienze di Base
Corso di Laurea in Ingegneria Informatica

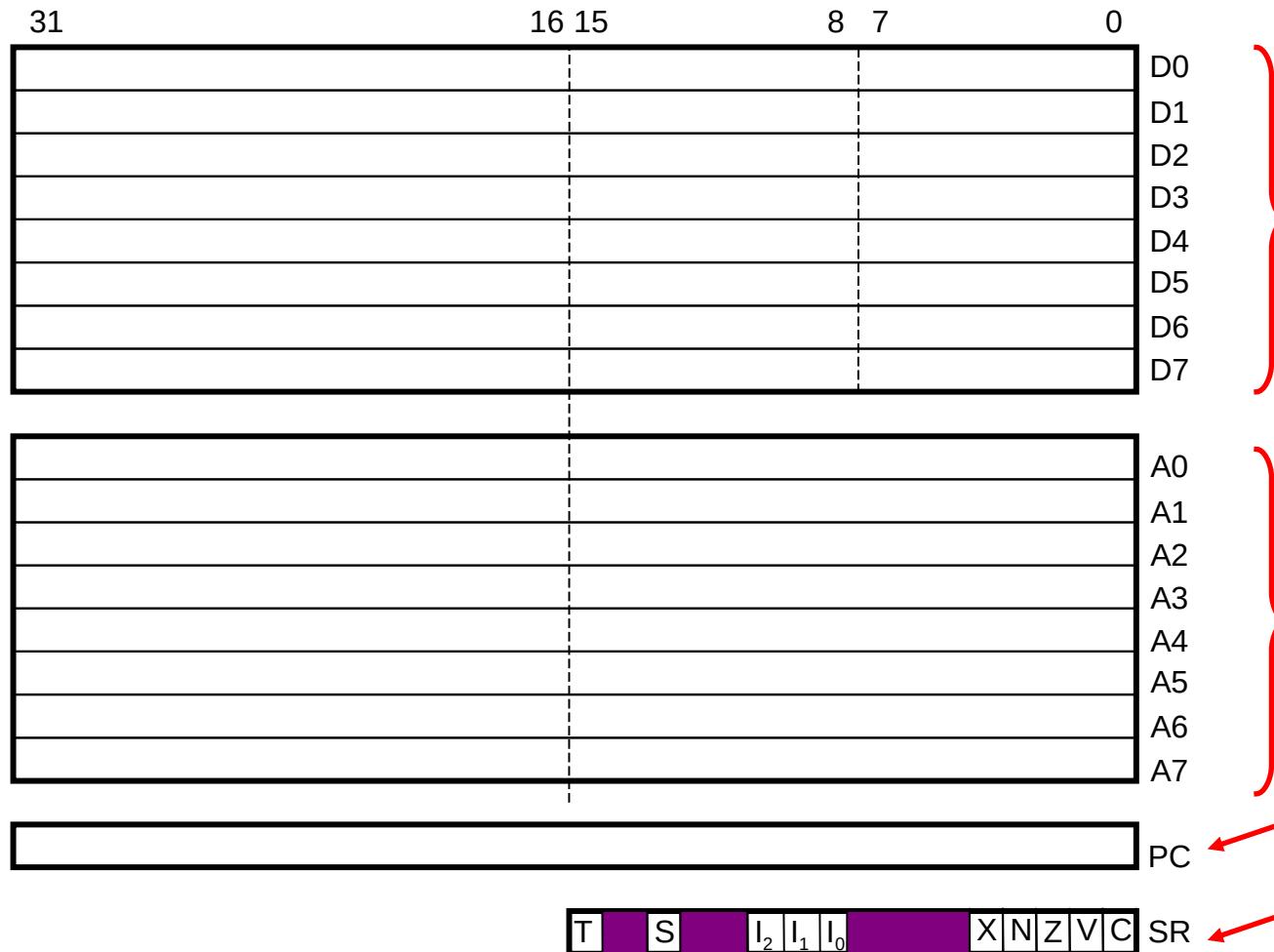


Corso di Calcolatori Elettronici I

MC68000: programmazione assembly



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

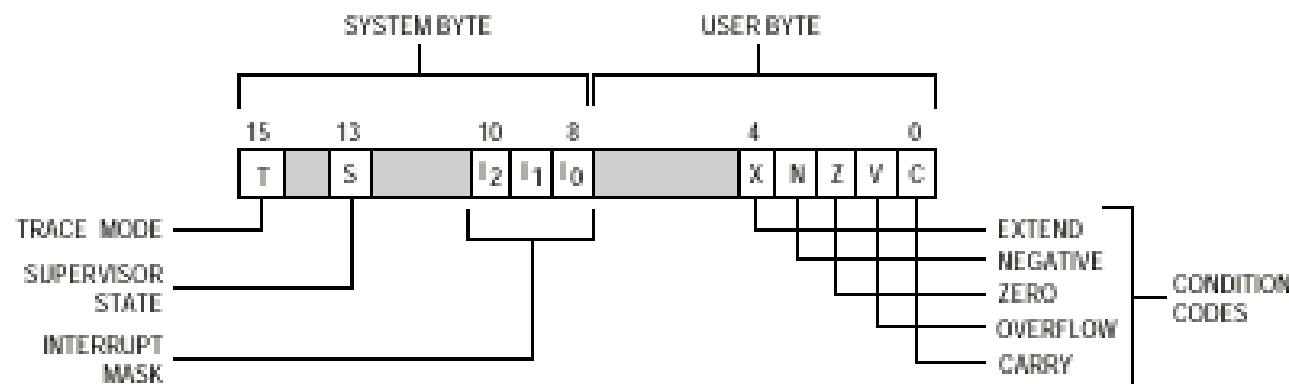


Figure 2-4. Status Register

Istruzione CLR: Clear



Operazione: [destination] \leftarrow 0

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

Istruzione MOVE

Operazione: [destination] \leftarrow [source]

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 sempre posto pari a 0 o 1, rispettivamente

Istruzione ADD



Operazione:

[destination] \leftarrow [source] + [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] \leftarrow [destination] + <literal>

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)

CMP: Compare

Operazione: [destination] – [source] → cc

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] → cc

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 di byte, word, o long

Bcc: Branch on condition cc



Operazione:

IF $cc = 1$ THEN $[PC] \leftarrow [PC] + d$

Sintassi:

Bcc <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 riscontrare nel registro di stato come effetto delle istruzioni precedenti (*vedi seguito*), ad esempio una *compare* (CMP)

Bcc: 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) $N \oplus V = 1$
 - BGE branch on greater than or equal $N \oplus V = 0$
 - BLE branch on less than or equal $(N \oplus V) + Z = 1$
 - BGT branch on greater than $(N \oplus V) + Z = 0$
- 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

- \$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

Istruzione di salto incondizionato JMP



Operazione: [PC] \leftarrow 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

- - - - -

DBcc: Test condition, decrement, and branch



Operazione: IF (*cc* false) THEN
 [D_n] ← [D_n] - 1
 IF [D_n] ≠ -1 THEN [PC] ← [PC] + *d*

Sintassi: DBcc D_n,<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.

LEA: Load Effective Address



Operazione: $[An] \leftarrow <ea>$

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

- - - - -

TST: Test an operand



Operazione: Destination Tested → Condition
Codes

Sintassi: TST <ea>

Esempio: TST (A3)

Attributi: Size = byte,word,longword

Descrizione:

Confronta l'operando con zero e setta i flag di stato di conseguenza:

X	N	Z	V	C
-	*	*	0	0

BTST: Test a bit



Operazione: TEST (<bit number> of Destination) → Z

Sintassi: BTST Dn, <ea>
 BTST # <data>, <ea>

Esempio: BTST #0,A3

Attributi: Size = byte,longword

Descrizione:

Testa un bit dell'operando destinazione e setta Z di conseguenza. Se la destinazione è un registro, si può testare uno qualunque dei 32 bit del registro. Se la destinazione è una locazione di memoria, l'operazione avviene su un singolo byte. Influenza solo il flag di stato Z:

X N Z V C

- - * - -

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 contatore 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	e 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

Esercitazione

- Scrivere ed assemblare un programma che moltiplichи 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 JMP sul PC
 - L'effetto dell'istruzione MOVE da registro a memoria e la rappresentazione esadecimale in memoria

Soluzione - mult2ints.a68



ORG \$8000

MULT	CLR.W	D0	D0 accumula il risultato
	MOVE.W	MPY, D1	D1 e' il contatore 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	e 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

Soluzione - Assemblaggio



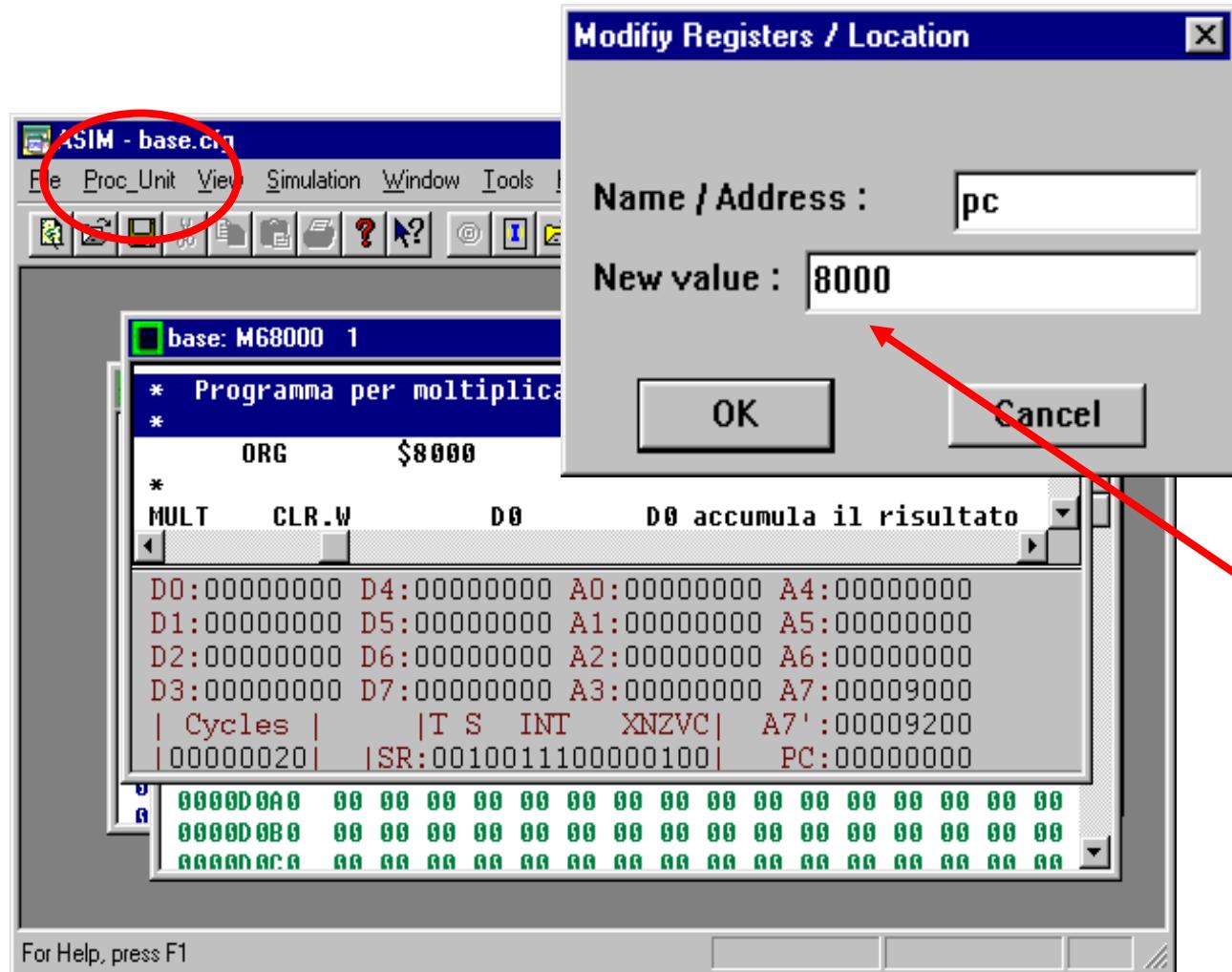
- Per assemblare il sorgente assembly visto prima, è possibile usare il programma AsimTool, che fornisce un'interfaccia grafica,
- oppure digitare da linea di comando:

68kasm.exe -l filename.a68

Soluzione - Esecuzione



DIE
TITEL.
UNI
NA



Per avviare il programma, occorre che il Program Counter (PC) contenga l'indirizzo della prima istruzione. Nel programma, tale indirizzo è determinato dalla direttiva

ORG \$8000

Il Program Counter può essere settato a mano dal menù Proc_Unit / Modify Register

Esercitazione

- 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
CNT	DS.W	1
SUM	DS.W	1
IVAL	EQU	17
ICNT	DC.W	IVAL
	END	START

Esercitazione

- Scrivere un programma che sommi i primi 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

Soluzione - sumnnums.a68



	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	\$8008
CNT		DS.W	1
SUM		DS.W	1
IVAL		EQU	17
ICNT		DC.W	IVAL

Esercitazione



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

Soluzione - scalprod.a68



DIE
TI.
UNI
NA

```
ORG      $8000
START    MOVE.L #A,A0
          MOVE.L #B,A1
          MOVE.L #N,D0
          SUBQ  #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  1,1,1,1,1,1,1,1,1,1
         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



DIE
TI.
UNI
NA

	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', 0
TOKENA	DS.L	1

Soluzione - token.a68



DIE
TI.
UNI
NA

	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



DIE
TI.
UNI
NA

Istruzioni di Controllo

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



DIE
TI.
UNI
NA

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



DIE
TI.
UNI
NA

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);
```

LOOP	MOVE.B #N, D2 MOVE.B #1, D1 MOVE.W #1, D0 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 // test della condizione del ciclo while
    istruzione_successiva
```

Esempio: calcola 3^N ($N \geq 0$)
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

Cicli for mediante DBRA

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

Esempio:

MOVE.L #N, D1	MOVE.L #N, D1
SUBQ.L #1, D1	SUBQ.L #1, D1
MOVEA.L #NUM, A2	MOVEA.L #NUM, A2
CLR.L D0	CLR.L D0
L0OP ADD.W (A2)+, D0	L0OP ADD.W (A2)+, D0
DBRA D1, L0OP	SUBQ #1, D1
MOVE.L D0, SOMMA	BGE L0OP
	MOVE.L D0, SOMMA

Somma in D0 gli N elementi del vettore memorizzato a partire dalla locazione di memoria NUM