# Fondamenti di programmazione in linguaggio assembly del Motorola 68000


Daniele Paolo Scarpazza
daniele.scarpazza@elet.polimi.it

Politecnico di Milano

Ultimo aggiornamento: 10 Maggio 2005

# Bibliografia

- Libro di testo:

  Hamacher, Vranesic & Zaky

  Introduzione all'architettura dei calcolatori

  McGraw-Hill Science

- Manuale di riferimento:

  MOTOROLA M68000 FAMILY

  Programmer's Reference Manual

  © Motorola Inc., 1992

  Disponibile a:

  **`http://www.scarpaz.com/processors/`**

- Fonti:

  Codice di esempio per il simulatore:
  **`http://goforit.unk.edu/asm/mc68000.htm`**

# Strumenti

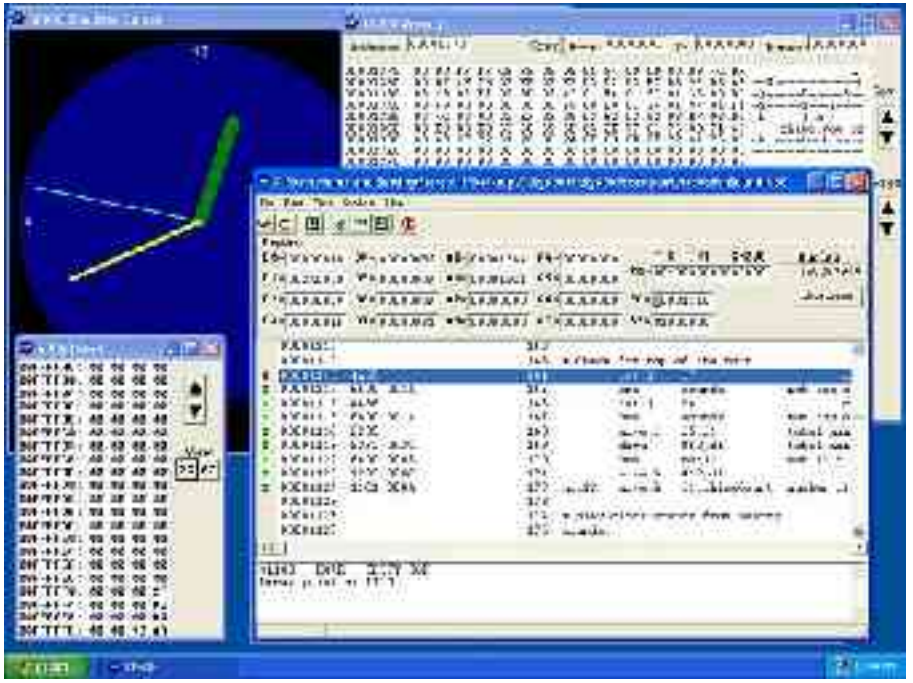
- Strumento da usare:

  EASy68K

  Editor/Assembler/Simulator for the 68000

  Disponibile presso:

  **`http://www.monroeccc.edu/ckelly/easy68k.htm`**



# Peculiarità del processore 68000

- Il 68000 ha 8 registri dati (D0-D7) e 8 registri indirizzi (A0-A7);
- I registri dati sono indifferenziati fra loro
  (non ci sono registri privilegiati come accumulatore o contatore);
- I registri sono di 32 bit (long word),
  ma si possono usare anche i soli 16 bit (word) e 8 bit (byte) meno significativi;
- Attenzione: se sposto solo un byte,
  gli altri 3 della long word restano invariati;
- La sintassi dell'istruzione MOVE è:  MOVE source, destination
- La pila cresce “verso il basso”, quindi:  push = SP--;  pop= SP++;
- Il registro A7 si usa per convenzione con *stack pointer*;
- Il registro PC (program counter) è di 24 bit.

# I codici di condizione (*flag*)

- Sono memorizzati in un registro di 16 bit chiamato SR (status register), composto da:

| | Byte di sistema | | | | | | | Byte utente (registro CCR, 8 bit) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| T | - | S | - | - | $I_2$ | $I_1$ | $I_0$ | - | - | - | X | N | Z | V | C |

**C** = riporto, carry; vale 1 se l'ultima istruzione ha generato riporto;
**V** = overflow; vale 1 se l'ultima istruzione ha generato un valore non rappresentabile;
**Z** = zero; vale 1 se l'ultima istruzione ha generato il valore zero;
**N** = negativo vale 1 se l'ultima istruzione ha generato un valore negativo;
**X** = "esteso"; copia del riporto non cambiata da tutte le istruzioni che cambiano C;

**T** = traccia; causa un interrupt dopo ogni istruzione; usato dai debugger;
**S** = system-mode; le istruzioni privilegiate possono essere eseguite solo quando S=1;
**I** = livello di abilitazione degli interrupt, su 3 bit; sono abilitati solo gli interrupt > I;

# Disposizione dati in memoria: *endianness*

- I processori della famiglia 68000 sono *big-endian*:
  il byte meno significativo sta all'indirizzo più alto;
- Un aiuto alla memoria: "dove finisce il numero ?"
  il byte meno significativo (quello con cui finisce il numero)
  sta all'indirizzo più alto (l'indirizzo più *"big"*);
- Esempio: l'istruzione

```
MOVE.L  #$12345678, $10000
```

  Modifica i seguenti byte in memoria come segue:

| Indirizzo | Valore |
|---:|---:|
| 10000 | $12 |
| 10001 | $34 |
| 10002 | $56 |
| 10003 | $78 |

# Istruzioni con suffissi di dimensione

- Molte istruzioni possono avere un suffisso che indica quanti bit usare degli operandi:

  .B = byte = 8 bit, i meno significativi
  .W= word = 16 bit, i meno significativi *(default)*
  .L = long word = 32 bit, tutti

- Esempio:

```
MOVE.L  #$12345678, D1
CLR     D0
MOVE.B  D1, D0  * D0 conterrà $00000078
CLR     D0
MOVE.W  D1, D0  * D0 conterrà $00005678
CLR     D0
MOVE.L  D1, D0  * D0 conterrà $12345678
```

# Uso della pila

- Lo *stack pointer* è conservato nel registro **`A7`**;
- Il registro **`A7`** è sdoppiato per poter gestire 2 pile (user mode / system mode), a seconda del flag S;
- Non ci sono istruzioni **`PUSH`** e **`POP`**: non servono!
  Si usa l'istruzione **`MOVE`** con indirizzamento indiretto con postincremento/predecremento;
- Esempio di *push*, impila il contenuto di D0:

```
MOVE.W D0, -(A7)
```

- Esempio di *pop*, estrae il *top* della pila in D0:

```
MOVE.W (A7)+, D0
```

# Errori comuni

- Non dimenticate '**#**' prima dei valori immediati.
  Esempio: **`MOVE $10,D0`**
  scrive in D0 non il valore 10 ma il valore alla locazione 10 (potenziale segmentation fault);
- Istruzioni a 2 operandi: la sintassi degli operandi è sorgente, destinazione (non viceversa);
- Usando suffissi **`.B`** e **`.W`**, la parte alta della destinazione resta invariata; se necessario pulirla con **`CLR`**
  o estendere il segno con **`EXT`**;
- L'uso di indirizzi **`(An)`** non allineati
  (cioè dispari!) non è ammesso;

# Classi di istruzioni

- Spostamento dati
- Aritmetica intera
- Operazioni a livello di bit
  (logica, test, shift e rotazione)
- Controllo di flusso
- Altre istruzioni

# Direttive dell'assemblatore

- Non sono istruzioni:
  non causano lo svolgimento di una operazione da parte della CPU,
  ma modificano la maniera in cui l'assemblatore assembla il programma;

- Le più importanti:
  - DC  Define Constant
  - DCB  Define Constant Block

```
DC.B   'Ciao',0   * stringa null-terminated in caratteri consecutivi
DC.L   $01234567  * valore $01234567 come long word
DC.W   1,2        * due words separate: $0001 e $0002
DC.L   1,2        * due long words separate: $00000001 e $00000002
```

  - EQU  Da un nome/etichetta ad un valore

```
BLUE   EQU $00FF0000
DSEG   EQU $600
```

  - ORG  Forza la locazione del prossimo elemento (origine)
  - END  Indica la fine del programma

```
       ORG  $60
DSEG   EQU  $60
       DC.W $FEDC
       DC.W $BA98
START: ORG $1000
       ...
       END START
```

# Istruzioni per il movimento di dati

- Le istruzioni più importanti sono:
  - MOVE copia da registro o locazione di memoria in un altro registro/altra locazione di memoria;
  - Esempi:
    - `MOVE.W  D0, D7`
    - `MOVE.L  -8(A6), D1`
    - `MOVE.B  #11, D1`
    - `MOVE.L  2(A7), (A4)`
  - MOVEM salva un gruppo di registri sulla pila o ripristina un gruppo di registri dalla pila;
  - Esempi:
    - `MOVEM.L D1-D3,-(SP)` * salva i registri sulla pila;
    - `MOVEM.L (SP)+,D1-D3` * ripristina i registri dalla pila;

# Aritmetica intera

Le istruzioni più importanti sono:

- ADD/SUB: addizione / sottrazione intera;
- ADDA/SUBA: idem, con destinazione un registro indirizzi;
- L'assemblatore può decidere autonomamente di usare al loro posto:
  - ADDI/SUBI se rileva un operando immediato,
  - ADDQ/SUBQ se rileva un operando immediato cortissimo (1-8);

- CLR: azzeramento di un operando;
- CMP confronto fra due operandi (varianti CMPI e CMPQ)
- CMPM confronto fra locazioni in memoria: `CMPM (Ay)+,(Ax)+`
- DIVS, DIVU divisione intera con e senza segno;
- EXT estensione del segno da byte a word o da word a longword;
- MULS, MULU moltiplicazione con e senza segno;
- NEG, NEGX complementazione a 2 di un operando;
- TST comparazione di un operando con 0;

# Operazioni a livello dei bit

- AND, ANDI — Logical AND on two binary integers
- OR, ORI — Logical OR
- EOR, EORI — Exclusive OR (XOR)
- NOT — Operand's 1's complement
- ASL, ASR — Arithmetic shift left/right.
- LSL, LSR — Logical shift left/right.
- ROL, ROR — Rotation left/right without extra carry;
- ROXL, ROXR — Rotation left/right through extra carry;
- BTST — Tests a bit
- BSET — Tests a bit, then sets it (1)
- BCLR — Tests a bit, then resets it (0)
- BCHG — Tests a bit, then inverts it (0 > 1, 1 > 0)

# Istruzioni per il controllo di flusso

Le istruzioni più importanti sono le seguenti:

- Bcc salti con 15 condizioni diverse;
- DBcc salti (in-)condizionati come sopra, che inoltre decrementano un registro contatore indicato; utili per i cicli;
- BSR, JSR istruzioni per il salto a sottoprogramma
  maggiori informazioni nella presentazione sui sottoprogrammi;
- JMP Istruzione per il salto ad un indirizzo assoluto;

# Le istruzioni Bcc

- BCC Branch Carry Clear — Branch if the C-flag is 0.
- BCS Branch Carry Set — Branch if the C-flag is 1.
- BEQ Branch EQual — Branch if the Z-flag is 1.
- BNE Branch Not Equal — Branch if the Z-flag is 0.
- BGE Branch Greater or Equal — Branch if N and V are equal.
- BGT Branch Greater Than — Branch if N and V are equal and Z=0.
- BHI Branch HIgher than — Branch if both C and Z are 0.
- BLE Branch Less or Equal — Branch if Z=1 or if N and V are different.
- BLS Branch Lower or Same — Branch if C=1 or Z=1.
- BLT Branch Less Than — Branch if N and V are different.
- BMI Branch MInus — Branch if N=1.
- BPL Branch PLus — Branch if N=0.
- BVC Branch V Clear — Branch if V=0
- BVS Branch V Set — Branch if V=1.
- BRA BRanch Always — Branch inconditionally.

# Come realizzare cicli con DBRA

- DBRA: decrementa un registro;
  se raggiunge il valore –1 continua, altrimenti salta;
- Codice di esempio: somma di dieci numeri interi *(simile a Fig. 5.8);*
  il ciclo è realizzato con l'istruzione DBRA:

```
* Program: sum.X68

SUM     EQU         $2000                   * the variable where we store the sum
ARRAY   DC.L        1,2,3,4,5,6,7,8,9,10    * an array of 10 integers
LENGTH  EQU         10                      * the array length

START   ORG         $1000
        MOVE.L      #LENGTH-1, D1       * initialize the loop index to N-1
        MOVEA.L     #ARRAY, A2          * initialize A2 to the array beginning

        CLR.L       D0                  * use D0 as accumulator; clear it
LOOP    ADD.L       (A2)+, D0           * sum current cell to D0
        DBRA        D1, LOOP            * decrement index D1 and loop
        MOVE.L      D0, SUM             * save the final sum to variable
                                        * final sum = 55 = $37
        STOP        #$2000
        END         START
```

# Altre istruzioni

- LINK/UNLK allocazione/distruzione di un record di attivazione
- STOP attende fino all'arrivo di un interrupt;
- TRAP 16 istruzioni di invocazione di routine del supervisore;
- NOP non fa nulla per un ciclo;