LABORATORIO DI ARCHITETTURA DEI CALCOLATORI

lezione n° 10

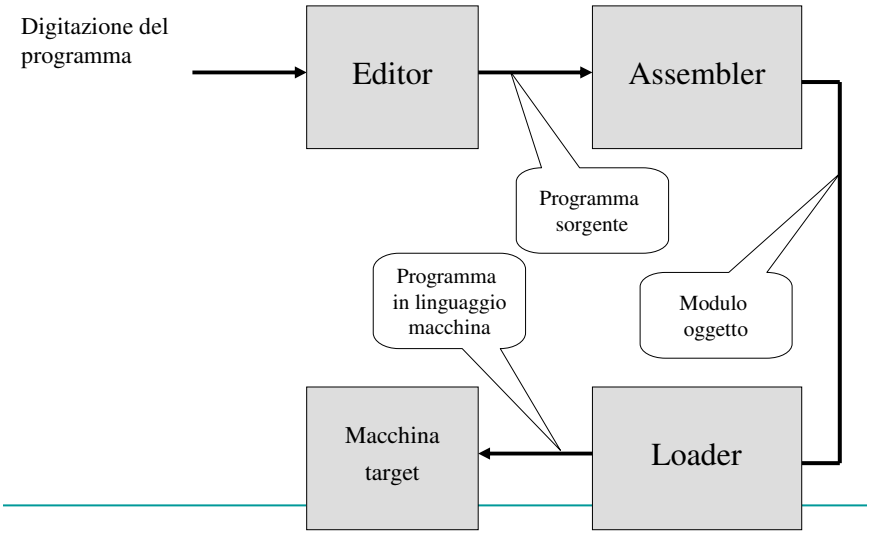
Prof. Rosario Cerbone

rosario.cerbone@uniparthenope.it

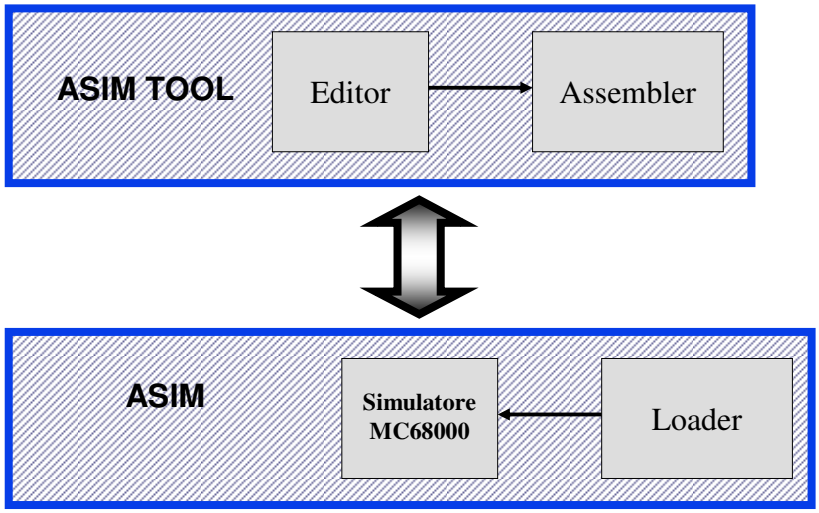
http://digilander.libero.it/rosario.cerbone

a.a. 2008-2009

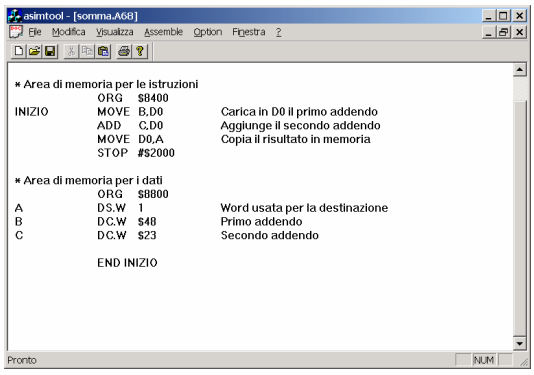
AMBIENTE DI SIMULAZIONE ASIM



AMBIENTE DI SIMULAZIONE ASIM



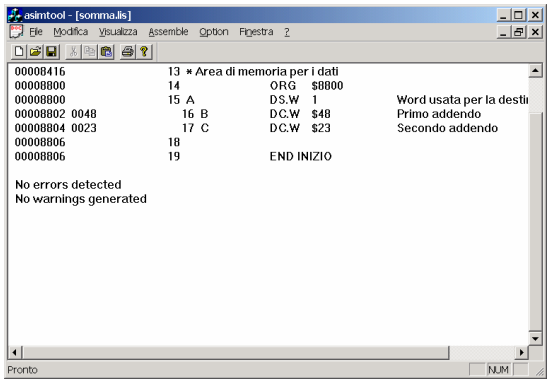
ASIMTOOL – ambiente per l'editing e l'assemblaggio



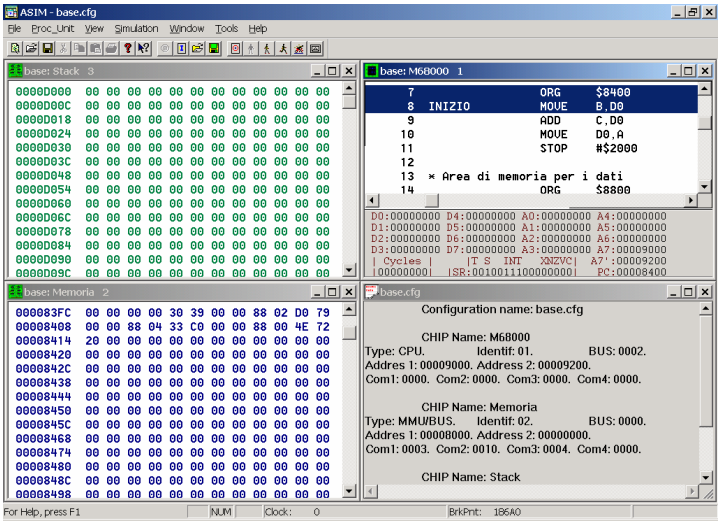
NOTA: Disabilitare in Windows la funzionalità “Nascondi le estensioni per i tipi di file conosciuti” in Strumenti→Opzioni cartella→Visualizzazione

ASIMTOOL – ambiente per l'editing e l'assemblaggio

Produce un file .lis che può essere caricato in ASIM.
Dopo aver assemblato il file sorgente, ASIMTOOL mostra il file .lis prodotto.



ASIM



Classi di istruzioni

- Un calcolatore deve avere istruzioni in grado di effettuare quattro tipi di operazioni
  - trasferimento dei dati tra la memoria e i registri di CPU;
  - operazioni aritmetiche e logiche sui dati;
  - controllo di flusso di un programma;
  - trasferimento dei dati in ingresso/uscita (I/O).

Istruzione CLEAR e MOVE (trasferimento dati)

- ***CLEAR* di un operando generale B,W o L**
  - **CLR G** effettua l'operazione G=0
- **MOVE G1, G2** effettua l'operazione G2 = G1
- ***MOVE quickly* (Move più veloce)**
  - **MOVEQ im, D** l'operando immediato è espresso su 8 bit; il registro destinazione non può essere A.
- ***MOVE* per operandi tipo indirizzo**
  - **MOVEA G, A** prevede un registro A come destinazione
  - **LEA G,A** l'indirizzo effettivo in G viene caricato in A (solo longword).
- **MOVEA.L #$1234,A0** e **LEA $1234,A0** eseguono la stessa operazione.

Istruzioni aritmetiche – operazioni unarie del 68000

- Rappresentazione in complementi alla base
- ***NEG*** cambia il segno dell'operando
  - **NEG.B D0** **produce D0 ← -[D0]**
    - se prima [D0]=11100111 dopo è [D0]=00011001
- ***NEGX*** sostituisce l'operando OP con – (OP+X)
  - **NEGX.B D0** **produce D0 ← -([D0]+X)**
    - se prima [D0]=11100110 e X=1 dopo è [D0]=00011001
    - se prima [D0]=11100110 e X=0 dopo è [D0]=00011010

Istruzione ADD e Istruzioni Logiche

ADD O1,O2 effettua l'operazione **[O1]+ [O2]→ [O2]**

- Poiché ***almeno uno tra O1 e O2 deve essere un registro dati*** le due sole forme ammesse sono:
  - **ADD G,D** ***op.sorgente generale, op.destinazione registro D_i***
  - **ADD D,G** ***op.sorgente registro D_i, op. destinazione generale***

ISTRUZIONI LOGICHE

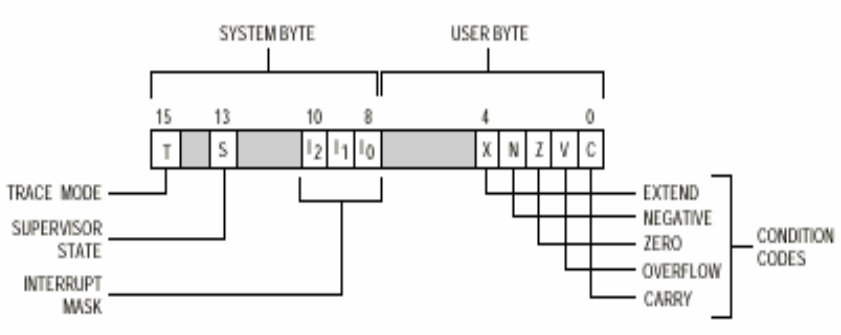
Operazione unaria **NOT**: effettua il not bit a bit;

Operazioni binarie **AND**, **OR**, **EOR** (strutturalmente simili a all'istr. ADD)

AND G, D equivalente a **D=D^G**

AND D, G equivalente a **G=G^D**

Il registro di stato del 68000



CCR = Condition Code Register

Codici di condizione

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.**

Modifica dei valori dei codici di condizione

- **ATTENZIONE**: Istruzioni, anche non aritmetiche, alterano i flag:
- **MOVE #0,CCR** mette a zero tutti i flag
- **MOVE.L #FFFFFFFF,D0** V=0, C=0, N=1, Z=0, e X rimane inalterato

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

Istruzioni di salto

SALTO: ***riposizionamento del Program Counter ad 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

Istruzioni di salto incondizionato: JMP e BRA

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 A*lways**)

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

Istruzioni di salto condizionato: Bcc

BRANCH effettuati o meno in base al valore assunto da specifici **flag di condizione**

B*cc dest* **if *cc* then PC= PC + *disp***

- **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

CMP: Comparazione aritmetica

- **CMP O1, O2** Posiziona i flag calcolando la differenza **O2 - O1**.
- Mentre **O1** è un operando generale **O2** deve essere un registro dati **D**
- **CMP IVAL, D3**
- **CMPI #data,O2** sottrae il dato immediato dalla destinazione e posiziona i flag.
- La relazione d'ordine tra **O1** e **O2** dipende dall'aritmetica usata
- Ad esempio se **O1 = $FF, O2 = $10:**
- **O1** > **O2** se i numeri sono interpretati come **unsigned** (**255 > 16**)
- **O1** < **O2** se i numeri sono interpretati come **signed** (**-1 < 16**)
- **NEL POSIZIONARE I FLAG DI CONDIZIONE IL PROCESSORE NON TIENE CONTO DEL TIPO DI RAPPRESENTAZIONE ARITMETICA**

Bcc: Branch on condition cc

- **Numeri *signed***
- **BLT** branch on less than (O2 < O1) $N \oplus V = 1$
- **BGE** branch on greater than or equal (O2 ≥ O1) $N \oplus V = 0$
- **BLE** branch on less than or equal (O2 ≤ O1) $(N \oplus V)+Z=1$
- **BGT** branch on greater than (O2 > O1) $(N \oplus V)+Z=0$

- **Numeri *unsigned***
- **BHI** branch on higher than (O2 > O1) C + Z = 0
- **BHS** branch on higher than or same (O2 ≥ O1) C = 0
- **BLO** branch on lower than (O2 < O1) C = 1
- **BLS** branch on lower than or same (O2 ≤ O1) C + Z = 1

- nota: BHS = BCC BLO = BCS

Istruzioni BTST, BCLR, BSET, BCHG

- L'istruzione **BTST k,G** posiziona il flag Z in funzione del valore del bit **k**-mo dell'operando **G**:
- MOVE #5, D0
- BTST #0, D0 Z=0
- BTST #1, D0 Z=1

Le istruzioni **BCLR, BSET, BCHG** operano come **BTST,** inoltre rispettivamente azzerano, pongono a 1, complementano il bit sul quale è stato fatto il test

Istruzioni TST e TAS

- **TST G** posiziona i flag N e Z in funzione del valore dell'operando G:

- MOVE.L #$FF, D0
- TST.B D0 Z=0, N=1
- TST.L D0 Z=0, N=0

- **TAS G** opera come TST ma solo su un dato di tipo byte, ed inoltre pone ad 1 il bit più significativo

Esempio

- Scrivere un programma assembly che esegua la somma tra due byte val1 e val2 e ponga il risultato in una locazione di memoria res.
- Si esegua il debug e la simulazione con Asim.

Esercizio 10.1

- Scrivere un programma che esegua la somma tra due word x e y e salvi il risultato in z.
- Si esegua il debug e la simulazione con Asim.

Esercizio 10.2

- Scrivere un programma che calcoli:

  z=x+y+4

- Si esegua il debug e la simulazione con Asim.

Esercizio 10.3

- Scrivere un programma che realizzi il costrutto if seguente:

```
if q=4 then
        x=5
    else
        x=y
```

- Si esegua il debug e la simulazione con Asim.

Esercizio 10.4

- Scrivere un programma che realizzi il confronto tra tre numeri A, B, C (unsigned tipo word) e li disponga in memoria in ordine crescente.
- Si esegua il debug e la simulazione con Asim.

Esercizio 10.5

- Scrivere un programma che realizzi il confronto tra tre numeri A, B, C (signed tipo word) e li disponga in memoria in ordine decrescente.
- Si esegua il debug e la simulazione con Asim.

Esercizio 10.6

- Scrivere un programma che realizzi il confronto tra dieci numeri (signed tipo word) e li disponga in memoria prima i pari in ordine crescente e dopo i dispari in ordine decrescente.
- Si esegua il debug e la simulazione con Asim.