CPUSim

Preparazione ambiente di lavoro

In questa lezione iniziamo ad utilizzare il programma CPUSim, che permette di simulare l'esecuzione di programmi assembler da parte di una ipotetica CPU. Il programma ci permetterà in seguito di modificare la descrizione della "macchina" simulata creando dei processori personalizzati sia come architettura sia come set di istruzioni.

Per prima cosa è necessario eseguire una serie comandi in modo da preparare l'ambiente di lavoro. Dopo aver avviato una shell linux digitare i seguenti comandi dando INVIO dopo ognuno:

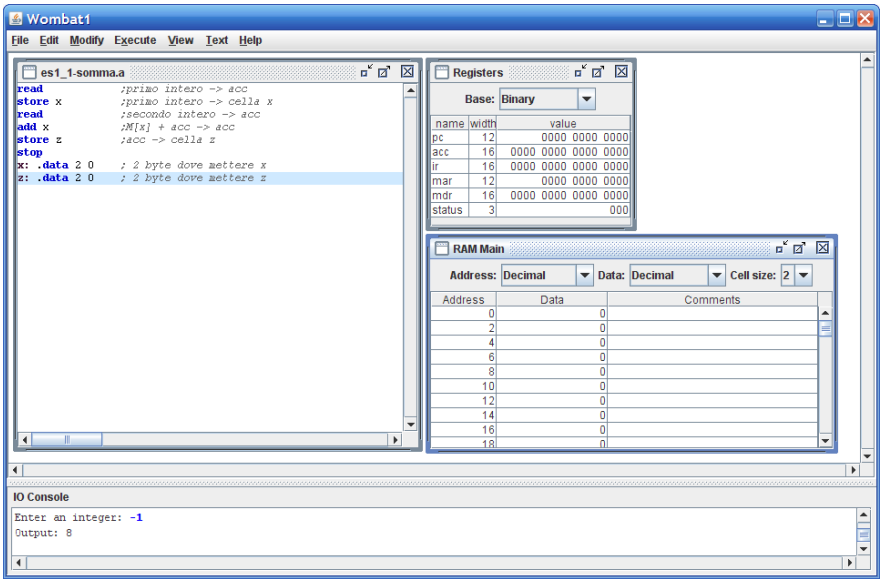
```
mkdir esercizi
cd esercizi
cp -R /usr/local/CPUSim3.6.7/SampleAssignments/* .
```

Notate che il . (punto) finale ha un significato ben preciso!…quale?

A questo punto per avviare il programma CPUSim, digitate il comando `CPUSim.sh`

Descrizione di CPUSim

All'avvio, il programma si presenta con una schermata simile alla seguente:



Si possono notare 3 finestre:

- la prima (a sinistra) mostra il listato del programma assembler da eseguire
- la seconda (in alto a destra) mostra i registri della CPU attualmente in simulazione e il loro contenuto
- la terza (in basso a destra) mostra le celle di memoria ed il loro contenuto

La parte inferiore della finestra mostra la console di Input/Output che serve per interagire con il programma in esecuzione inserendo dati ed ottenendo risultati.

Laboratorio 22-11-2010

La CPU caricata di default (se non viene caricata, File->Open Machine... e caricate Wombat1.cpu) corrisponde ad un'architettura generica in cui sono definiti i registri:

- **PC** (program counter) l'indirizzo della locazione di memoria contenente la successiva istruzione da eseguire
- **ACC** (accumulator) contiene i risultati della ALU
- **IR** (instruction register) contiene l'istruzione da eseguire, quella cioè puntata dal PC
- **MAR** (memory address register) contiene l'indirizzo della locazione di memoria che viene acceduta
- **MDR** (memory data register) contiene temporaneamente tutti i dati e le istruzioni che dalla memoria devono essere elaborati nel processore
- **Status** (registro di stato) memorizza una serie di bit indicativi dello stato corrente del processore (halt, overflow, underflow, ecc)

Il ciclo di esecuzione della CPU simulata è:

1. pc --> mar
2. Main[mar] --> mdr
3. mdr --> ir
4. inc2-pc
5. decode-ir

Istruzioni: [**etichetta:] operatore operandi [; commento]**
Ad esempio:

ADD x ; M[x]+acc --> acc

Pseudo-istruzioni Dati: **etichetta: .data nByte valore [; commento]**
Ad esempio:

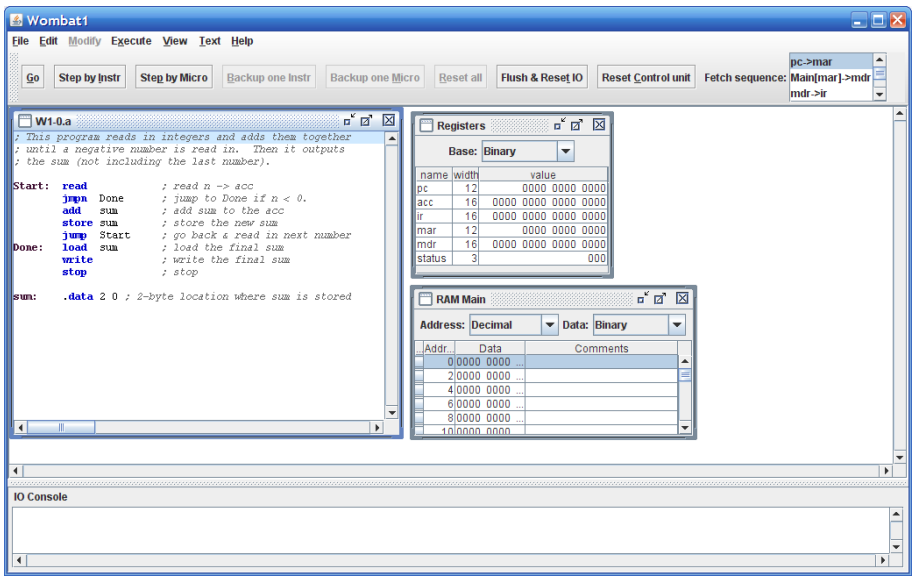
x: .data 2 0 ; x è una locaz. di memoria di 2Byte iniz. a 0

Eseguire un programma

Per eseguire un programma per prima cosa bisogna caricare il file contenente il listato usando il comando File -> Open text
In alternativa, cliccando File -> New Text comparirà una finestra vuota nella quale iniziare a scrivere un nuovo programma da zero.
Una volta deciso quale sarà il programma da eseguire, per prima cosa questo deve essere assemblato (in questa fase viene eseguita un'analisi sintattica del listato) usando Execute -> Assemble.
ATTENZIONE: il comando appena descritto sarà eseguito sul programma visualizzato nella finestra attualmente selezionata. Oppure, se non è selezionato alcun sorgente, il comando sarà applicato all'ultimo programma assemblato. Quindi è buona cosa selezionare il programma corretto prima di eseguire Execute -> Assemble.
Una volta assemblato, il programma deve essere caricato in memoria per essere eseguito; la voce di menu Execute -> Assemble&Load permette di verificare la correttezza del programma e di caricarlo in memoria. Successivamente si può avviare l'esecuzione con Execute->Run.
Per eseguire nuovamente il programma, premere Execute Clear, Assemble, Load & run.

Modalità di debug

Dal menù Execute, cliccando su Debug Mode, si può passare alla modalità di debug. In questa modalità la gestione dell'avanzamento dell'esecuzione è lasciata all'utente, il quale può eseguire una istruzione alla volta (Step by Instr) e controllare lo stato di memoria e registri dopo ogni istruzione, oppure avanzare nell'esecuzione una microistruzione alla volta (Step by Micro).



Esercizi

ESERCIZIO 1 [es1_1-somma.a]

Legge due numeri (usando la locazione di memoria x) e salva la somma nella locazione di memoria z.

```
read                ; primo intero -> acc
store x             ; primo intero -> cella x
read                ; secondo intero -> acc
add x               ; M[x] + acc -> acc
store z             ; acc -> cella z
write
stop
x: .data 2 0        ; 2 byte dove mettere x
z: .data 2 0        ; 2 byte dove mettere z
```

ESERCIZIO 2 [es1_2-val-ass.a]

Calcola e stampa il valore assoluto di un intero ricevuto in input.

```
read                    ; input -> acc
jmpn negativo           ; se acc <0, salta a negativo
fine: write             ; acc -> output
stop                    ; stop
```

```
negativo: store copia    ; acc -> copia
load zero                ; zero -> acc
subtract copia           ; acc-copia -> acc
jump fine                ; va alla fine
zero: .data 2 0          ; 2 byte dove mettere zero
copia: .data 2 0         ; 2 byte dove mettere copia
```

ESERCIZIO 3 [es1_3-quo-rest-sott.a]

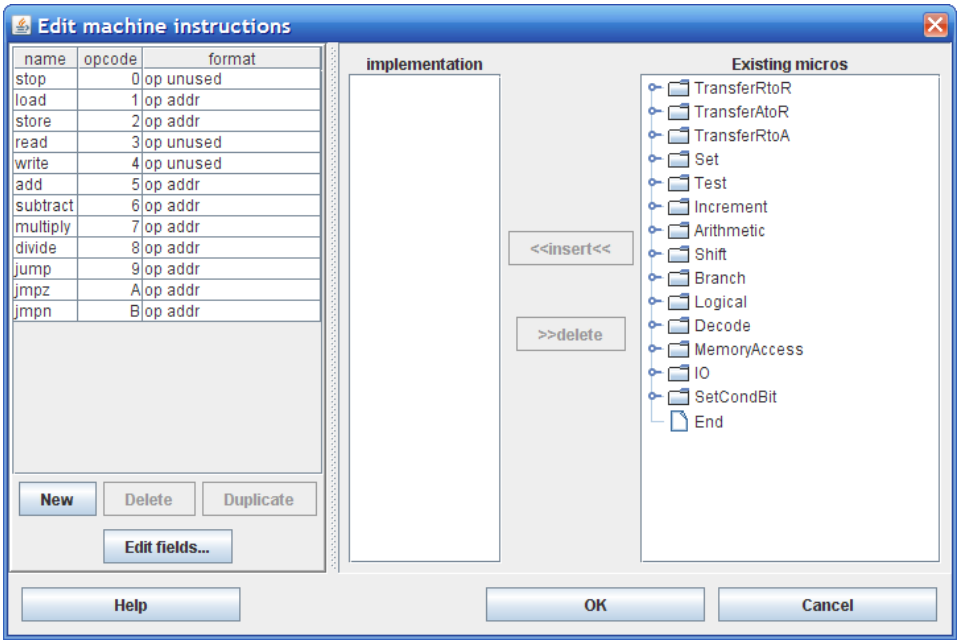
Calcola quoziente e resto della divisione di due interi x e y (x/y) usando solo sottrazioni.

```
; esegue x/y
; nota: legge prima y di x
read                     ; legge il primo numero
store y                  ; lo mette in M[x]
read                     ; legge il secondo numero
store x                  ; lo mette in M[y]
ciclo: subtract y        ; x-y -> acc
jmpn fine                ; se x<y -> vai a fine
store x                  ; altrimenti, x-y -> M[x]
load Uno                 ; 1 -> acc
add Quoziente            ; acc+ quoziente -> quoziente
store Quoziente              ; memorizzo Quoziente
load x                   ; carico x
jump ciclo               ; salta a ciclo
fine: load Quoziente         ; quoziente -> acc
write                    ; acc -> output STAMPA QUOZIENTE
load x                   ; M[x] -> acc
write                    ; acc -> output      STAMPA RESTO
stop                     ; stop
x: .data 2 0                      ; 2 byte dove mettere x
                               (alla  fine  ;  il  resto
                               della divisione)
y: .data 2 0             ; 2 byte dove mettere y
Quoziente: .data 2 0     ;   2   byte   dove   mettere   il
quoziente
Uno:.data 2 1           ; 2 byte dove mettere uno
                         ; (inizializzati a 1)
```

Machine Instructions

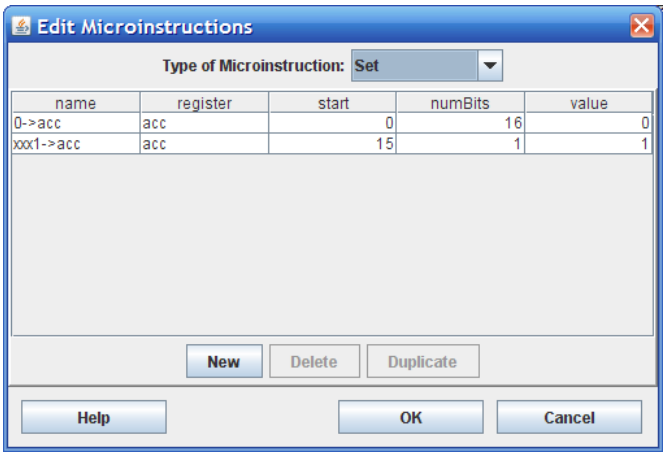
CPUSim permette di modificare l'insieme di istruzioni della cpu simulata. Nella parte sinistra della finestra, che si ottiene cliccando sul menu Modify -> Machine instructions, è presente la tabella che mostra l'insieme delle istruzioni disponibili. Per ogni istruzione troviamo il nome, il codice operativo e il formato costituito da una serie di stringhe descrittive. La parte destra invece, serve per definire una nuova istruzione (dopo aver

premuto il pulsane "New" in basso a sinistra). Per creare una nuova istruzione basta selezionare ed inserire le microistruzioni adatte dalla lista (albero) di destra.
La figura seguente mostra la finestra di modifica delle istruzioni.
Il pulsante Edit Fields permette di modificare i codici di campo usati nella definizione del formato delle istruzioni.



Machine Microistructions

Dal menu Modify -> Microistructions è possibile modificare le microistruzioni della macchina attualmente caricata in CPUSim. Una microistruzione è definita dal suo nome, dal registro sul quale opera, dal bit iniziale sul quale operare, dal numero di bit interessati e dal valore da assegnare a questi bit.
La figura seguente mostra la definizione di due nuove microistruzioni che permettono di impostare l'accumulatore a "0" (tutti e 16 i bit a 0) e a "1" (fissando a 1 solo il bit meno significativo).



Oltre alle due microistruzioni mostrate in figura creare anche una nuova istruzione

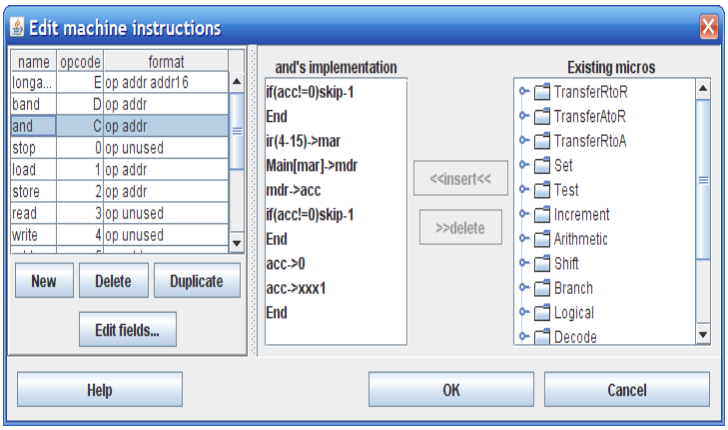
che esegue l'AND bit a bit tra i registri acc e mdr e mette il risultato in acc. L'istruzione si dovrà chiamare "accANDmdr ->acc" e sarà di tipo "Logical".

Nuove istruzioni

Creeremo alcune nuove istruzioni che ci saranno utili nella realizzazione di nuovi esercizi.

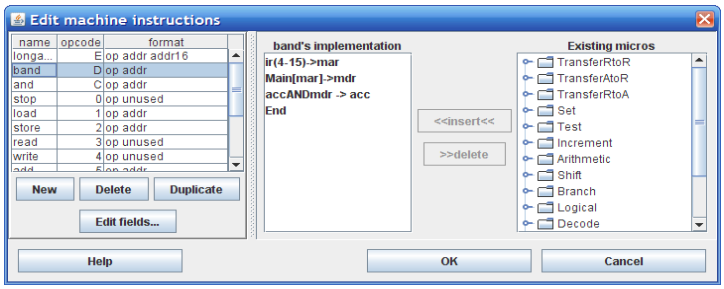
And

Esegue la funzione logica "and" tra l'accumulatore e una locazione di memoria, trattando un valore diverso da zero come "1" logico e 0 come "0" logico.



Band

Esegue la funzione logica and bit a bit tra l'accumulatore e una locazione di memoria.



Esercizi

ESERCIZIO 4 [es1_4-and_logico.a]

Definire ed utilizzare una nuova istruzione che esegue l'AND logico tra due numeri. Un numero diverso da zero è interpretato come "1" logico.

```
read
store op1
read
and op1
write
```

```
stop
op1: .data 2 0
```

ESERCIZIO 5 [es2_2-and_bit.a]

Definire ed utilizzare una nuova istruzione che esegue l'AND logico bit a bit tra due numeri.

```
     read
     store op1
     read
     band op1
     write
     stop
op1:      .data 2 0
```

PER CASA

ESERCIZIO 1c [es1_4-prod_somma.a]

Calcola il prodotto di due interi usando somme.

```
; leggere due interi x e y e calcolare il prodotto x*y
; non usare l'istruzione multiply

read            ; legge -> acc
store x             ; acc -> x
read            ;
store y             ; acc -> y
ciclo: jmpz fine;
load sum        ; sum -> acc
add x;              ; acc + x -> acc
store sum       ; acc -> sum
load y              ; y -> acc
subtract uno        ; acc - 1
store y             ; acc -> y
jump ciclo          ;
fine: load sum      ; somme parziali -> acc
write               ;
stop            ;
x: .data 2 0;
y: .data 2 0;
sum: .data 2 0;
uno: .data 2 1;
```

ESERCIZIO 2c [es1_5-somma_seq.a]

Il programma legge una sequenza di interi e li somma finché non legge un numero negativo. Alla fine stampa la somma (senza includere l'ultimo numero).

```
; il programma legge una sequenza di interi e li somma
; finché non legge un numero negativo. Alla fine stampa
; la somma (senza includere l'ultimo numero)


Inizio:   read      ; legge n -> acc
jmpn  Fatto         ; salata a Fatto se n < 0.
add   somma         ; aggiunge somma ad acc
store somma         ; memorizza la nuova somma
jump  Inizio            ;  salta  indietro  e  legge  il
                        prossimo
                        ; numero
Fatto:    load  somma   ; carica la somma finale
write               ; scrive il risultato
stop                ; si ferma

somma:    .data 2 0     ;  locazione  di  2  byte  dove  è
          memorizzata
               ; somma
```

ESERCIZIO 3c [es1_6-max-seq.a]

Calcola il massimo di una sequenza di interi positivi. Inserire un numero negativo per terminare.

```
; Calcola il massimo di una sequenza di interi positivi.
; Inserire un numero negativo per terminare.
;
inizio: read        ; legge un intero positivoacc
jmpn fine           ; se e' negativo, vai a fine
store valore        ; altrimenti acc -> valore
subtract massimo    ; acc -> massimo -> acc
jmpn inizio         ; acc<massimo: prossimo numero
jmpz inizio         ; acc=massimo: prossimo numero
load valore         ; altrimenti, valore -> acc
store massimo       ; acc -> massimo
jump inizio         ; legge il prossimo numero
fine: load massimo  ; massimo -> acc
write               ; acc -> output
stop                ; stop
massimo: .data 2 0  ; 2 byte per massimo
valore: .data 2 0   ; 2 byte per valore
```

ESERCIZIO 4c [es1_7-pari-dispari.a]

Il programma legge un intero in ingresso e ritorna 0 se l'intero è pari, -1 altrimenti.

```
; legge un intero in input e
; se e' pari restituisce 0,
```

```
; altrimenti restituisce -1

read      ; n -> acc
store x         ; acc -> x
divide due      ; acc/2 -> acc
multiply due    ; acc*2 -> acc
subtract x      ; acc - x -> acc
write           ; visualizza acc
stop      ; termina il programma

x: .data 2 0
due: .data 2 2
```

ESERCIZIO 5c [es2_3-or_bit.a]

Definire ed utilizzare una nuova istruzione che esegue l'OR logico bit a bit tra due numeri.

```
     read
     store op1
     read
     bor op1
     write
     stop
op1:      .data 2 0
```