CPUSim

Avvio di CPUSim

Avviare il programma CPUSim, digitando il comando `cpusim.sh` seguito da INVIO.

Attenzione: CPUSim in aula informatica è stato aggiornato alla versione 3.6.8 quindi è consigliabile usare la descrizione della cpu che si trova in:
`/usr/local/CPUSim3.6.8/SampleAssignments/`

Gestire CPU diverse

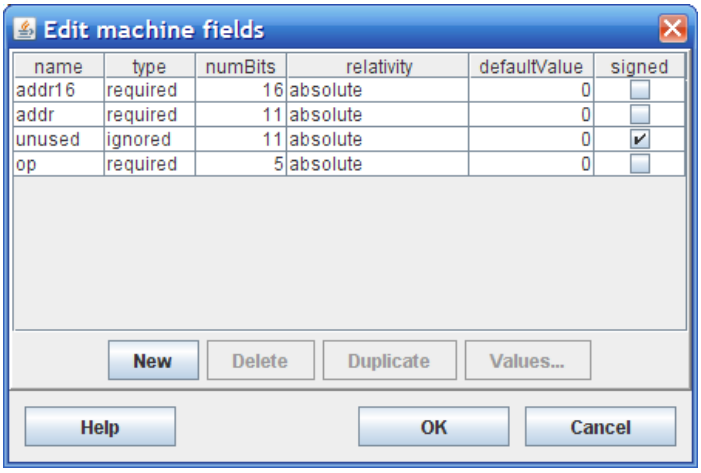
Come abbiamo visto nella lezione precedente, in CPUSim è possibile definire una nuova CPU o modificarne una esistente. Dopo aver apportato delle modiche alla CPU (ad esempio al set di istruzioni) è possibile salvare per poterla riutilizzare in seguito. Conviene salvare la descrizione della nuova macchina senza sovrascrivere quella attuale; per fare questo usare il comando File Save machine as.
Quando sarà necessario riutilizzare una CPU creata precedentemente, sarà sufficiente caricarne la descrizione con il comando File Open machine.

Machine Microistructions

Creare una nuova istruzione che copia i 12 bit meno significativi da mdr a mar (l'istruzione si chiamerà "mdr -> mar"). Di che tipo sarà?

Modificare i campi delle istruzioni

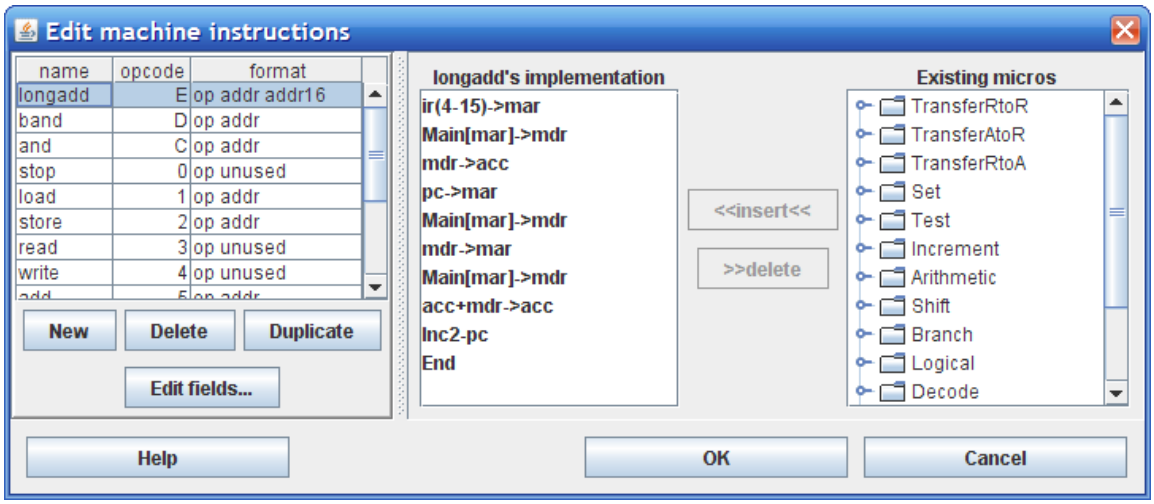
Il formato di ogni istruzione è definito utilizzando dei nomi di campo, per modificare ed inserire nuovi campi, premere il pulsane Edit fields nella finestra di modifica delle istruzioni. Negli esempi successivi, avremo bisogno di un nuovo campo, chiamato ***addr16***. Il campo servirà per definire il secondo operando dell'istruzione ***longadd***. Visto che un'istruzione deve occupare un numero di bit multiplo di 8 e minore o uguale a 64, la lunghezza di ***addr16*** deve essere **16** anche se poi ci bastano **12** bit per indirizzare la memoria. Quindi l'istruzione ***longadd*** sarà lunga 4(op)+12(addr)+16=32 bit.
La figura seguente mostra i campi che serviranno per definire le istruzioni descritte successivamente.



Nuove istruzioni

Longadd

Esegue la somma tra due locazioni di memoria e mette il risultato nell'accumulatore. Le locazioni di memoria contenenti i dati da sommare sono agli indirizzi *addr* e *addr16*.



Salviamo la CPU aggiornata con le nuove istruzioni e microistruzioni con File Save machine as e inserendo il nome **Wombat2**.

Esercizi

ESERCIZIO 1 [es2_4-longadd.a]

Definire ed utilizzare una nuova istruzione longadd che esegue la somma su due parametri in ingresso e scrive il risultato nell'accumulatore.

```
read
store op1
read
store op2
longadd op1 op2
write
stop
op1: .data 2 0
op2: .data 2 0
```

ESERCIZIO 2 [es2_5-longadd.a]

*Usando la nuova istruzione longadd leggere due interi x e y e calcolare il prodotto x*y.*

```
read            ; legge -> acc
store x             ; acc -> x
read            ;
store y             ; acc -> y
```

```
ciclo: jmpz fine    ;
longadd prodotto x  ; prodotto + x -> acc
store prodotto      ; acc -> prodotto
load y              ; y -> acc
subtract uno        ; acc - 1
store y             ; acc -> y
jump ciclo          ;
fine: load prodotto ; somme parziali -> acc
write               ;
stop          ;

x: .data 2 0;
y: .data 2 0;
prodotto: .data 2 0;
uno: .data 2 1;
```

Indirizzamento

Come avete visto a lezione è possibile specificare gli operandi di un'istruzione in vari modi:

- Indirizzamento immediato
- Indirizzamento diretto
- Indirizzamento indiretto
- Indirizzamento su registro
- Indirizzamento su registro indiretto
- Indirizzamento con spiazzamento (displacement)
- Indirizzamento su pila

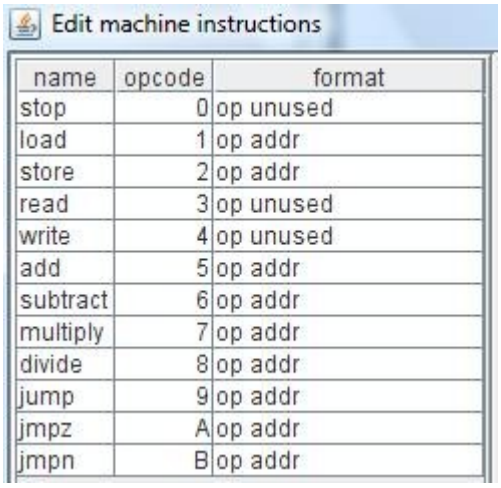
Nelle esercitazioni di laboratorio precedenti abbiamo simulato indirizzamento diretto (istruzioni: op addr) e indirizzamento su registro (acc). Nell'esercitazione di oggi modificheremo la CPU simulata per poter utilizzare diverse modalità di indirizzamento.

Apriamo la CPU ottenuta al termine dell'esercitazione precedente (Wombat2.cpu) con il comando File Open machine; poiché andremo a modificarla e non intendiamo sovrascriverla, salviamola subito con un nuovo nome (Wombat3.cpu) attraverso il comando File Save machine as.

In realtà, della CPU definita nell'esercitazione precedente ci interessa mantenere solo la microistruzione per trasferire i 12 bit meno significativi di mdr in mar. Possiamo quindi eliminare le istruzioni che abbiamo definito (*band*, *bor*, *and* e *longadd*) e salvare nuovamente la CPU.

Dovremmo trovarci con il seguente set di istruzioni:

Laboratorio 29-11-2010



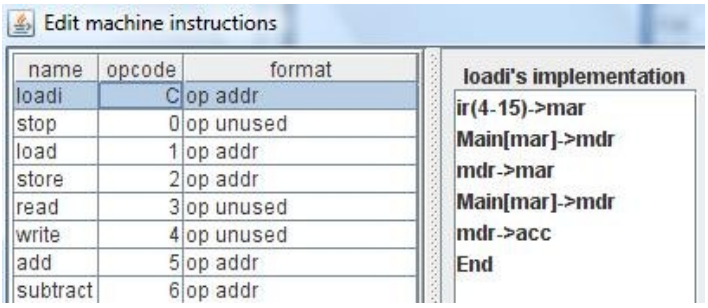
name	opcode	format
stop	0	op unused
load	1	op addr
store	2	op addr
read	3	op unused
write	4	op unused
add	5	op addr
subtract	6	op addr
multiply	7	op addr
divide	8	op addr
jump	9	op addr
jmpz	A	op addr
jmpn	B	op addr

Indirizzamento indiretto

Volendo simulare l'indirizzamento indiretto dobbiamo innanzitutto definire delle nuove istruzioni che interpretino gli operandi non come indirizzi di memoria da caricare, ma come indirizzi degli indirizzi di memoria da caricare. Ciò permette di aumentare lo spazio di indirizzamento al costo di duplicare gli accessi in memoria. È necessario definire almeno le istruzioni di load, store e add con indirizzamento indiretto (non è necessario definire nuove microistruzioni).

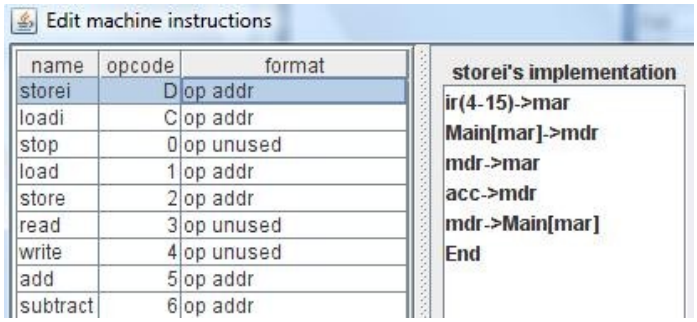
Loadi

Esegue la *load* indiretta, dove l'operando *addr* non è più l'indirizzo di memoria da caricare ma l'indirizzo dell'indirizzo della locazione di memoria da caricare.
Entriamo in modalità di modifica delle istruzioni con Modify -> Machine instructions e definiamo la nuova istruzione duplicando l'istruzione *load.* Assegniamo un *opcode* consistente e modifichiamo l'implementazione.



Storei

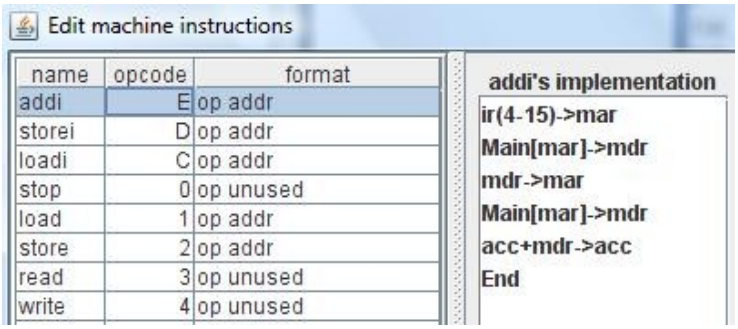
Esegue la *store* indiretta, dove l'operando *addr* non è più l'indirizzo di memoria su cui memorizzare il contenuto dell'accumulatore ma l'indirizzo dell'indirizzo della locazione di memoria di destinazione. Definiamo la nuova istruzione duplicando l'istruzione *store.* Assegniamo un *opcode* consistente e modifichiamo l'implementazione.



Addi

Esegue la *add* indiretta, dove l'operando *addr* non è più la locazione di memoria del valore da sommare all'accumulatore ma l'indirizzo della locazione di memoria del

valore da sommare. Definiamo la nuova istruzione duplicando l'istruzione *add*. Assegniamo un *opcode* consistente e modifichiamo l'implementazione.



ESERCIZIO 3 [es3_1-indiretto.a]

Utilizzando istruzioni ad indirizzamento indiretto, 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). *Nota: se definiamo la locazione per la somma ad un indirizzo occupato dal codice del programma, viene automaticamente corretta in fase di assemblaggio. Non avviene lo stesso se definiamo una locazione dispari!*

```
inizio:    read          ; legge n -> acc
jmpn  fine               ; salata a Fine se n < 0.
addi   somma             ; aggiunge somma ad acc
storei somma             ; memorizza la nuova somma
jump  inizio                ; legge il prossimo numero
fine: loadi  somma       ; carica la somma finale
write                    ; scrive il risultato
stop                     ; si ferma

somma: .data 2 20        ; locazione (2 byte) per memorizzare
          somma
```

ESERCIZIO 4 [es3_2-sequenza.a]

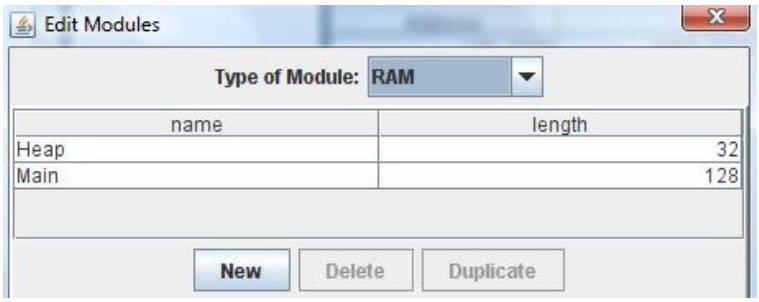
Utilizzando istruzioni ad indirizzamento indiretto, il programma legge due interi, chiamiamoli ind e cont. Successivamente, scrive dei valori partendo dall'indirizzo di memoria identificato da ind per cont volte. I valori saranno cont, cont-1, ecc fino a 1. *Attenzione: se definiamo la locazione per la somma ad un indirizzo occupato dal codice del programma questo verrà sovrascritto! Utilizzare una locazione pari!*

```
read             ; 1° indirizzo su cui scrivere
store ind        ; indirizzo->ind
read             ; valore da scrivere nel 1° indirizzo
store cont       ; valore->cont

ciclo: jmpz fine
storei ind       ; acc->indirizzo riferito ind (indiretto)
subtract uno          ; decrementa valore
store cont       ; aggiorna valore
load ind         ; carica indirizzo
add due               ; incrementa indirizzo (2 Byte)
```

```
store ind       ; aggiorna indirizzo
load cont       ; valore->acc
jump ciclo
fine: stop

ind: .data 2 0        ; locazione base
cont: .data 2 0       ; valore
uno: .data 2 1
due: .data 2 2
```

Moduli di memoria aggiuntivi

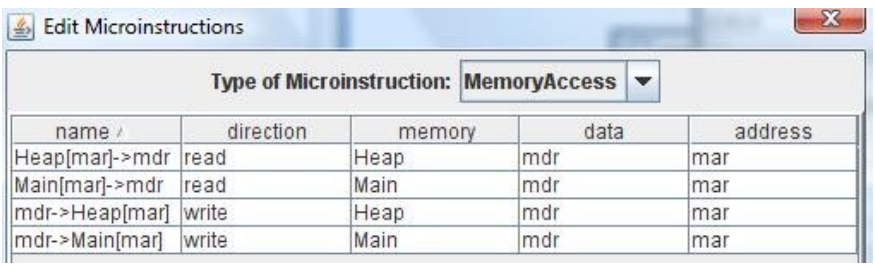
CPUSim permette di definire moduli di memoria aggiuntivi che possono essere poi usati nei programmi dopo aver definito delle nuove istruzioni e microistruzioni che usano la nuova memoria.
Per aggiungere e/o modificare i moduli di memoria della cpu bisogna accedere alla funzionalità Modify -> Hardware Modules, e poi selezionare RAM come tipo di modulo. Al momento è presente un unico modulo di memoria (*Main*); definiremo un nuovo modulo di memoria che chiameremo *Heap*. Selezioniamo New ed inseriamo Heap come nome e 32 come lunghezza (il modulo di memoria consisterà di 32 Bytes: *quanti indirizzi di memoria?*).



Come si è detto, per poter utilizzare il nuovo modulo di memoria è necessario definire delle istruzioni e microistruzioni apposite.

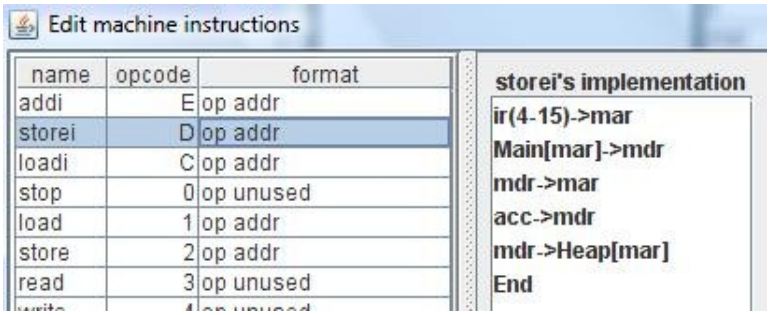
Microistruzioni per Heap

Per poter usare il nuovo modulo di memoria *Heap* bisogna innanzitutto creare due nuove microistruzioni, chiamate **Heap[mar]->mdr** e **mdr->Heap[mar]**.
Heap[mar]->mdr si occupa di trasferire il contenuto della cella di indirizzo mar della memoria Heap nel registro mdr, mentre **mdr->Heap[mar]** effettua l'operazione opposta, ovvero trasferisce il contenuto dell'mdr nella cella della Heap di indirizzo mar.
Dal menu Modify Microistructions creiamo le due microistruzioni, sotto la categoria Memory Access.



ESERCIZIO 5

Ridefinire le istruzioni per l'indirizzamento indiretto in modo che l'esercizio 4 scriva sull'Heap e non sulla memoria principale.



ESERCIZIO 6 [es3_3-sequenza.a]

Verificare il funzionamento dell'esercizio 4 con la nuova istruzione che scrive sull'Heap Si ricorda che il programma legge due interi, chiamiamoli ind e cont. Successivamente, scrive dei valori partendo dall'indirizzo di memoria identificato da ind per cont volte. I valori saranno cont, cont-1, ecc fino a 1. *Attenzione: in questo caso la scelta della locazione ind non è più critica poiché non interferisce più con la memoria in cui è memorizzato il codice del programma. È necessario comunque specificare un indirizzo pari!*

```
read            ; 1° indirizzo su cui scrivere
store ind       ; indirizzo->ind
read            ; valore da scrivere nel 1° indirizzo
store cont      ; valore->cont

ciclo: jmpz fine
storei ind      ; acc->indirizzo riferito ind (indiretto)
subtract uno        ; decrementa valore
store cont      ; aggiorna valore
load ind            ; carica indirizzo
add due             ; incrementa indirizzo (2 Byte)
store ind       ; aggiorna indirizzo
load cont       ; valore->acc
jump ciclo
fine: stop

ind: .data 2 0      ; locazione base
cont: .data 2 0     ; valore
uno: .data 2 1
due: .data 2 2
```