Tutorato di Informatica - Soluzioni Foglio 2

28 Marzo 2025 - Foglio 2

Esercizio 1

Si consideri l'istruzione "`j xx`".
Domanda: Occupa più spazio in memoria la sua rappresentazione ASCII o l'istruzione macchina corrispondente dopo che è stata assemblata?

Soluzione

La rappresentazione ASCII è lunga 4 caratteri, quindi occupa 4B, cioè 32bit, che è esattamente pari alla lunghezza di una qualsiasi istruzione macchina in MIPS32, dunque occupano lo stesso spazio.

Esercizio 2

Interpretare il codice esadecimale `0x0232502A` come istruzione MIPS, ed indicare:

- La sua rappresentazione binaria
- L'Op-code ed il formato (R-type, I-type o J-type)
- la rappresentazione simbolica dell'istruzione, completa del valore dei suoi parametri

Nota: usare l'appendice del libro a pagina A50

Soluzione

- In rappresentazione binaria l'istruzione diventa
  0000 0010 0011 0010 0101 0000 0010 1010
- L'op-code è dato dai 6 bit più significativi, ovvero 000000. A questo op-code corrispondono più operazioni, che si differenziano a seconda del valore dei 6 bit meno significativi (funct) rispettando duque il formato R-type: [opcode:6][rs:5][rt:5][rd:5][shamt:5][funct:6]

- Innanzitutto ricaviamo i valori dei campi, separando la rappresentazione binaria come opportuno per il formato R-Type:
  - Op-code: $000000 = 0_{10}$
  - rs: $10001 = 17_{10}$
  - rt: $10010 = 18_{10}$
  - rd: $01010 = 10_{10}$
  - shamt: $00000 = 0_{10}$
  - funct: $101010 = 42_{10}$

  L'istruzione corrispondente è identificata da opcode (0) e funct ($101010_2 = 42_{10}$), che corrispondono all'operazione `slt`, ovvero "set less then", che scrive in `rd` il risultato booleano del confronto `rs<rt`.
  La rappresentazione simbolica dell operazione è
  `slt rd, rs, rt`,
  che nel nostro caso diventa
  `slt $10, $17, $18`

Esercizio 3

Indicare il formato dell'istruzione per effettuare la differenza tra due registri, ed esprimere esplicitamente l'operazione che effettua una sottrazione tra il valore nel registro $3 e quello nel registro $4, e deposita il risultato nel registro $2:

- in forma simbolica
- in forma binaria
- in forma esadecimale

Soluzione

- L'istruzione MIPS che effettua la differenza tra i contenuti di due registri è `sub rd, rs, rt` (vedi Appendice A, pag.A-56), che è di formato R-type
- `sub $2, $3, $4`
- 000000 00011 00100 00010 00000 100010
- 0x00641022

Esercizio 4

Stiamo eseguendo il seguente programma

Indirizzo	Istruzione (parziale)
0x00400000	add $9, ...
0x00400004	sub $14, ...
0x00400008	lw $8, ...
0x0040000C	add $7, ...

e ad un certo punto il program counter (PC) contiene il valore
0000 0000 0011 1111 1111 1111 1010 0100
inoltre sta per essere eseguita un'istruzione con op-code 000010. Come devono essere i bit rimanenti di questa istruzione, affinchè dopo venga eseguita l'istruzione `lw $8, ...`?

Soluzione

l'op-code indicato è quello di un salto incondizionato `j addr`, dunque la prossima istruzione che verrà eseguita sarà quella corrispondente all'indirizzo codificato nei rimanenti 26bit dell'istruzione (campo addr). Vogliamo che la prossima istruzione sia `lw $8, ...`, che si trova all'indirizzo 0x00400008. L'indirizzo a cui si salta si ottiene a partire da addr, effettuando un bitshift di 2 a sx e copiando i 4 bit piu significativi dal PC. Innanzitutto scrivo l'indirizzo di arrivo in forma binaria:
0000 0000 0100 0000 0000 0000 0000 1000
Noto che i 4 bit piu significativi sono gli stessi del PC, dunque mi basta rimuovere quelli ed i due bit meno significativi, ottenendo quindi
0000 0100 0000 0000 0000 0000 10

Esercizio 5

Qual è l'indirizzo della più lontana cella di memoria in cui è possibile saltare (in avanti) con una istruzione `bne` se
PC=0010 0010 1100 0011 0110 0010 0100 0000 ?

Soluzione

La più lontana cella di memoria a cui possiamo saltare in avanti con un'istruzione di salto condizionato come `bne` è ottenuta sommando al contenuto del PC, il numero più grande positivo rappresentabile in CA2 con 16 bit (offset o branch address che avremmo nell'istruzione bne), ovvero 0111 1111 1111 1111, concatenati con due bit a 0.
In questo caso quindi
0010 0010 1100 0011 0110 0010 0100 0000 +

0000 0000 0000 0001 1111 1111 1111 1100 =
0010 0010 1100 0101 0110 0010 0011 1100

Esercizio 6

Scrivere un programma che calcoli la somma del valore che è memorizzato nel registro $17, e del valore memorizzato nella locazione di memoria che si trova 14 word più avanti dell'indirizzo specificato al registro $16, e memorizzari il risultato 3 parole di memoria più avanti rispetto alla locazione attuale del secondo operando.
Nota: Vedi le istruzioni `lw` e `sw`, e ricorda che puoi usare la sintassi `n(x)` per indicare un indirizzo che abbia un offset di `n` byte rispetto all indirizzo `x`

Soluzione

Le operazioni che vogliamo effettuare sono quindi

1. sommare il contentuo del registro $17 al valore in memoria che sta 14 word (quindi 56=14*4 byte) dopo l'indirizzo indicato nel registro $16
2. salvare il risultato in memoria ad un indirizzo che sta 3 word (12 byte) dopo quello del secondo addendo

Una possibile soluzione è

```
lw $8, 56($16) #copio in $8 il valore che sta 56B (14 word) piu avanti dell'indiriz
add $9, $17, $8 #sommo $8 con $17 e scrivo il risultato in $9
sw $9, 68($16) #copio il risultato in memoria, 12 byte (=3 word) dopo 56($16)
```

Esercizio 7

I valori relativi alle variabili della dimensione di una word a, b, c, d sono memorizzati di seguito in memoria a partire dall'indirizzo specificato dal contenuto del registro $10. Scrivere la sequenza di istruzioni assembly che aggiunge la costante 10 alle variabili a, b, c, d e salva i nuovi valori delle variabili in memoria. N.B. Per la lettura dei valori della variabili successive alla prima, sarà possibile incrementare il valore dell'offset (di 4 byte per ciascuna word) oppure incrementare il contenuto del registro che contiene l'indirizzo base. La prima di queste opzioni NON permette di realizzare un ciclo mentre la seconda sì

Soluzione

```
# Soluzione sequenziale
lw $8, 0($10)
```

```
addi $8, $8, 10
sw $8, 0($10)
lw $8, 4($10)
addi $8, $8, 10
sw $8, 4($10)
lw $8, 8($10)
addi $8, $8, 10
sw $8, 8($10)
lw $8, 12($10)
addi $8, $8, 10
sw $8, 12($10)

# Soluzione con ciclo
add $11, $zero, $zero #inizializzo a 0 il contatore di cicli in $11
addi $9, $zero, 4 # numero di cicli da eseguire in $9
ciclo:
    lw $8, 0($10)
    addi $8, $8, 10
    sw $8, 0($10)
    addi $10, $10, 4
    addi $11, $11, 1
    bne $11, $9, ciclo
```

Esercizio 8

In QtSpim, scrivere (manualmente) in memoria, nella sezione Data, la stringa seguente: "`Hello world!`". La stringa deve apparire leggibile nel giusto ordine nell'interfaccia del programma. NOTA: Usare la tabella ASCII (extended) per la codifica dei caratteri.

Soluzione

Per scrivere la stringa sono necessari 12 caratteri, ciascuno di 2B, per un totale di 24B = 3 word di 4 caratteri. Apriamo il tab Data, e cliccando sui registri col destro inseriamo i valori ASCII corrispondenti alle word di caratteri rispettando l'endianity, ovvero le seguenti word:

1. "Hell" = (l)(l)(e)(H) = 6C 6C 65 48
2. "o wo" = (o)(w)( )(o) = 6F 77 20 6F
3. "rld!" = (!)(d)(l)(r) = 21 64 6C 72

Esercizio 9

In QtSpim, inserire manualmente due valori nei registri `$t0` e `$t1`. Poi scrivere un programma assembly che calcoli la somma dei due numeri presenti nei registri `$t0` e `$t1` e metta il risultato in `$t2`.

Soluzione

Per cambiare i valori basta cliccare col destro sui registri e inserire i numeri indicati.
Il programma che dobbiamo scrivere consiste di un'unica operazione di somma, che l'appendice del libro ci ricorda essere l'operazione `add rd, rs, rt`, che nel nostro caso diventa
`add $t2, $t0, $t1`
Apriamo quindi un editor di testo, definiamo la label `main:`, per indicare dove inizia il programma, e scriviamo l'unica istruzione, quindi salviamo il file con estensione `.asm`. Possiamo quindi caricarlo in QtSpim ed eseguirlo. Se tutto è andato bene, vedremo il risultato della somma apparire in t2!

Esercizio 10

Scaricare il programma `la_mia_prima_somma_Assembly.asm` da Moodle, caricarlo ed eseguirlo passo passo in QtSpim. Annotare, ad ogni passaggio (istruzione), il valore del PC e dei registri usati, ed inoltre monitorare cosa succede nelle sezioni `.text` e `.data` .

Esercizio 11

Completare il programma `ricerca_in_array.asm` che cerca il valore 5 all'interno di un array di 10 numeri interi. Farlo girare su QtSpim, passo passo e verificarne il funzionamento. Per verificare il funzionamento dell'intero programma, modificare il vettore perché contenga, o meno, il valore che si sta cercando.

Soluzione

Vedi file `ricerca_in_array.asm` risolto

Esercizio 12

Scrivere un programma in assembly che esegua le seguenti operazioni:

1. Leggere due numeri interi (num1 e num2) usando l'apposita syscall

2. Confrontare i due valori,

3. Stampare a schermo il più piccolo dei due.

dopodiché caricarlo su QtSpim e testarlo.
Consiglio: puoi partire dal file dell'esercizio precedente e prendere spunto; la tabella delle syscall ti può tornare utile.
Extra: stampa a schermo delle istruzioni per l'utente dove opportuno (consiglio: direttive .data e .asciiz)

Soluzione

Vedi file `interazione_utente.asm`