Linguaggio dei computer

Thursday, 17 August 2023 09:09

- La parola del linguaggio di un computer è chiamata istruzione
  - Tutte le istruzioni iniziano con un nome che indica la istuzione, e poi i suoi parametri
    Es: add a, b, c -> a = b+c
    F = (g+h)-(i+j)
    L'istruzione in binario:
    Add t0, g, h
    Add t1, i, j
    Sub f, t0, t1
    - In particolare, la struttura di un codice in MIPS è fatta da:

Op	Rs	Rt	Rd	Shamt	Funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

      - Op = operazione
      - Rs/Rt = Primo/Secondo registro parametri
      - Rd = Risultato registro
      - Shamt = Shift
      - Funct = Selezionare variante della funzione
    - Per motiazione di "e se un parametro occupasse di più?" certe operazioni usano la seguente formattazione:

Op	Rs	Rt	Costant or address
6 bits	5 bits	5 bits	16 bits

  - Questi parametri devono venire da una spazio di memoria chiamat registri
    - Un computer può avere come massimo 32 registri
    - Riscriviamo il codice prima con i registri:
      Add $t0, $s1, $s2
      Add $t1, Ss3, $s4
      Sub $s0, $t0, $t1
    - Quando abbiamo un array, i dati vengono gestiti nella memoria e la variabile dell'array contiene in che spazio di memoria si trova.
      - Noi però non possiamo lavorare con la memoria, quindi dobbiamo trasportare i dati dalla memoria ai registri
      - Per fare questo esiste l'istruzione load

G = h + A[8]
Prima dobbiamo trasformare A[8] in un registro
Lw $t0,32($s3)
32 sta per l'offset dell'array, stiamo andando a supposizione che la memoria funziona a 4 bit e quindi, 8*4 offset
$s3 è la parte di memoria
Quindi ora abbiamo nel registro $t0 il valore A[8]
Quindi per fare la somma add $s1, $s2, $t0

- Se invece volessimo togliere un dato dal registro e metterlo nella memoria dobbiamo fare store -> $sw
  Esempio completo: A[12] = h + A[8]
  Lw $t0,32($s3) -> Portiamo nel registro A[8], 32=8*4
  Add $t0, $s2, $s3 -> Sommiamo h in A[8]
  Sw $t0,48($s3) -> Mettiamo in A[12] la variabile $t0, 48=12 *4
- Se però ci pensiamo, noi avremo più variabili di quanti spazi di registri. Quindi il computer dovrà continuare a fare Load e Store.
  - Il computer tenta di tenere le variabili più comuni nei registri
  - Questo processo di tenere le variabili più comuni viene chiamato spilling registers

- Quando dobbiamo lavorare delle costanti, si può utilizzzare addi
  - A=b+4
    Normalmente avremmo mosso dalla memoria la variabile che contine la costante 4
    Lw $t0, AddrCostant4($s1)
    E dopo avremmo sommato
  - Però ora si può fare semplicemente
    Addi $s3, $s4, 4

- IF

In MPSI le decisioni si fanno con BEQ e BNE

```
Bne register1, register2, LI
```

Se registro 1 è uguale a registro 2 vai all'operazione LI
BNE differenza dal fatto che, anzichè "uguale" è "non uguale"

E' possibile "saltare" da un operazione ad un altra con l'istruzione "jump" j POSITION
E noi in assembly diciamo dov'è questa posizione

I loop si possono fare con BNE:

- While(save[i]==k)
  I += 1
- Loop: sll $t1, $s3, 2 |-> Per prendere save[i], dobbiamo prima trovare i, e questa i ha un offset
  Add $t1, $t1, $s6 |-> Sommiamo l'offset trovato sopra con l'offset della variabile save
  Lw $t0, 0($t1) |-> $t1 contiene save[i], e lo mettiamo nel registro $t0
  Bne $t0, $s5, Exit |-> Se sono uguali save[i] e k, allora finisci il loop, senò entra
  Addi $s3, $s3, 1 |-> i += 1
  J loop |-> Rifai il loop

- Se vogliamo testare equality oppure inequality, si usano slt, slti, sltu
  - Slt $t0, $s3, $s4 |-> Se $s3 < $s4, setta $t0 a 1
  - Slti $t0, $s2, 10 |-> se $s2 è < 10, setta $t0 a 1
  - Sltu |-> come slt ma li controlla come se fossero tutti unsigned
- Le funzioni esistono anche qui, si chiamano procedure

Facciamo la seguente funzione passo per passo:

```
Int leaf_example(int g, int h, int i, int j) {
    Int f;
    F = (g+h) - (i+j);
    Return f;
}
```

Prima di iniziare con la funzione, dalle nozioni di programmazione sappiamo che i parametri vengono salvati nello stack. In assembly lo stack si chiama $sp

Iniziamo a creare la procedura

```
Leaf_example:
    Addi %sp, %sp, -12 |-> Allora, noi stiamo usando una funzione, e le variabili delle funzioni vanno
                                Dentro lo stack, qui stiamo crendo spazio
                                lo spazio per esse
    Sw $t1, 8($sp) |-> Spazio per $t1
    Sw $t0, 4($sp)
    Sw $s0, 0($sp) |-> Differenza tra t ed s è che t non viene preservata in una chiamata di procedure
    ''' Vari caldoli noiosi che non sto a scrivere '''
    Add $v0, $s0, $zero |-> il registro V è il valore di ritorno di una funzione. Qui stiamo copiando s0
    Lw $s0, 0($sp) |-> Ripuliamo lo stack
```