Esempi di assembler MIPS

Engineering Department in Ferrara

Esempi in linguaggio assembly MIPS

- Ciascun esempio viene specificato con del codice C
- Il codice assembly non é ottenuto da un compilatore, ma é fatto a mano e con qualche prova sembra funzionare
- Cosa vuole dire verificare che il codice funziona?
- É un problema piuttosto generale e complicato
  - si deve provare il codice con ogni possibile ingresso?
  - non si puó per problemi semplicissimi
  - facciamo un numero di prove che consenta di eseguire ogni istruzione o di seguire ogni path nel control flow
- L'idea é di approfondire lo strumento

Esempio: if

```
int a=8;
int b=-16;
int x=0;
if (a>0) || (b>0) then
  x=1;
else
  x=2;
x=x*2;
```

```
addi $s0, $zero, -16
addi $s1, $zero, -8
addi $s2, $zero, 0
slt $t0, $zero, $s0
slt $t1, $zero, $s1
or $t2, $t0, $t1
beq $t2, $zero, T
addi $s2, $s2, 1
j Join
T: addi $s2, $s2, 2
Join: sll $s2, $s2, 1
```

Esempio: nested if

```
int a=-16;
int b=-8;
int c=4;
int x=10;
if (a>b)
  x=x+1;
else
  if (b>c)
    x=x+2;
  else
    x=x+3;
y=x+1;
```

```
addi $s0, $zero, -16 # a
addi $s1, $zero, -8  # b
addi $s2, $zero, 4   # c
addi $s3, $zero, 10  # x
slt $t0, $s1, $s0    # 1 if a>b else 0
beq $t0, $zero, Test0 # branch if a<=b
addi $s3, $s3, 1
j Join
Test0: slt $t0, $s2, $s1
# 1 if b>c else 0
beq $t0, $zero, Test1 # branch if b<=c
addi $s3, $s3, 2
j Join
Test1: addi $s3, $s3, 3
Join: addi $s4, $s3, 1
```

Tradurre un ITE in assembler sembra un poco macchinoso

Ecco come apparirebbe la traduzione diretta di un if in codice C

```
if (cond)
  x=....;
else
  y=....;
```

```
if (!cond)
  goto label;
x=....;
goto join;
label: y=....;
join:
```

Esempio: loop e array

```
int array0[8]={1,12,4,7,8,9,10
int array1[8];
int i=0;

while (i<8)
  {
    array1[i]=array0[0];
    i=i+1;
  }
```

```
.data
    array0: .word 1,12,4,7,8,9,10,11
    array1: .space 32
    # the two arrays have the same size

    .text

    # index i
    addi $s0, $zero, 0
    # array size in bytes
    addi $t0, $zero, 32

    loop: beq $s0, $t0, exit
    # pseudointructions
    # load in at the base address of
    # the arrays
    lw $t2, array0($s0)
    sw $t2, array1($s0)
    addi $s0, $s0, 4
    j loop
    exit:
```

Esempio: loop e array, una possibile alternativa

```
int array0[8]={1,12,4,7,8,9,10
int array1[8];
int i=0;

while (i<8)
  {
    array1[i]=array0[0];
    i=i+1;
  }
```

```
.data
    array0: .word 1,12,4,7,8,9,10,11
    array1: .space 32
    # the two arrays have the same size

    .text

    # index i
    addi $s0, $zero, 1
    la $t0, array0 # load the base address
    la $t1, array1
    addi $t2, $zero, 9 # 8 words

    loop: beq $s0, $t2, exit
    lw $t3, 0($t0)
    sw $t3, 0($t1)
    addi $s0, $s0, 1 # used for counting
    addi $t0, $t0, 4
    addi $t1, $t1, 4
    j loop
    exit:
```

Esempio: algoritmo di bubble sort

```
int i,j,swap;
const n=8;
int array0[8]={1,12,4,7,8,9,10,1
i=0;
swap=1;
while ((i<n) && swap)
 {
  swap=0;
  /* Comparing array elements */
  j=0;
  while (j<n-i-1)
   {
     if (array0[j]>array0[j+1])
       int temp=arr[j];
       arr[j]=arr[j+1];
       arr[j+1]=temp;
       swap=1; }
     j=j+1;
   }
  i=i+1;
 }
```

```
.data
array0: .word 1,12,4,7,8,9,10,11
n: .word 8
.text
# index i
addi $s0, $zero, 0
# size
lw $t0, n($zero)

loop_ext: beq $s0, $t0, exit

 addi $s2, $zero, 0 # swap variable
 sub $t1, $t0, $s0 # j=n-i-1
 addi $t1, $t1, -1

# index j
 addi $s1, $zero, 0
 loop_int: beq $s1, $t1, exit_int
  sll $t2, $s1, 2
  sll $t3, $s1, 2
```

Esempio: algoritmo di bubble sort

```
int i,j,swap;
const n=8;
int array0[8]={1,12,4,7,8,9,10,1
i=0;
swap=1;
while ((i<n) && swap)
 {
  swap=0;
  /* Comparing array elements */
  j=0;
  while (j<n-i-1)
   {
     if (array0[j]>array0[j+1])
       int temp=arr[j];
       arr[j]=arr[j+1];
       arr[j+1]=temp;
       swap=1; }
     j=j+1;
   }
  i=i+1;
 }
```

```
  addi $t3, $t3, 4   # j+1
  lw $t4, array0($t2)
  lw $t5, array0($t3)
  slt $t6, $t5, $t4
  beq $t6, $zero, noswap
   sw $t4, array0($t3) # swap the elements
   sw $t5, array0($t2)
   addi $s2, $zero, 1 # increment swap
  noswap: addi $s1, $s1, 1
  j loop_int
 exit_int: addi $s0, $s0, 1
# use the swap
beq $s2, $zero, exit

j loop_ext
exit:
```