

Architettura dei Calcolatori e Sistemi Operativi

MIPS - Linguaggio Assembly

Chair

Politecnico di Milano

Prof. C. Brandolese

e-mail: carlo.brandolese@polimi.it
phone: +39 02 2399 3492
web: home.dei.polimi.it/brandolese

Teaching Assistant

D. Iezzi

e-mail: [domenico.iezzi \[at\] polimi \[dot\] it](mailto:domenico.iezzi@polimi.it)
material: github.com/NoMore201/polimi_cr_acso_2019

Outline

▪ Linguaggio Assembly MIPS

- Simulatore MARS
- Struttura di programma
- Dichiarazione di dati
- Registri
- Istruzioni
 - Istruzioni standard
 - Pseudoistruzioni standard
 - Pseudoistruzioni estese
- Traduzione di costrutti
 - IF
 - IF - ELSE
 - WHILE
 - DO - WHILE
 - FOR
- Chiamata a funzione
 - Salvataggio di contesto

Simulatore MARS

■ Il simulatore MARS

- IDE per il linguaggio assembly MIPS
- Sviluppato in Java (quindi richiede una JRE) dalla Missouri State University
- Scaricabile da: <http://courses.missouristate.edu/kenvollmar/mars/>

■ Caratteristiche

- Interfaccia grafica, editor integrato
- Registri e memoria editabili
- Visualizza valori in decimale ed esadecimale
- Esecuzione step-by-step

Struttura di un programma

- File testuali con dichiarazione di dati, istruzioni (estensione .asm per MARS)
- Sezione di dichiarazione dati seguita dalla sezione istruzioni
- **Dichiarazione dati (.data, 0x10010000)**
 - Posizionata in una sezione identificata dalla direttiva `.data`
 - Dichiara i nomi delle variabili usate dal programma
 - Allocare in memoria centrale (RAM)
- **Codice (.text, 0x00400000)**
 - Posizionate in una sezione identificata dalla direttiva `.text`
 - Contiene le istruzioni del programma
 - Inizio identificato dall'etichetta `main`
 - Fine dovrebbe utilizzare una exit system call
- **Commenti**
 - Tutto ciò che è seguito da un `#`
`# questo è considerato un commento`

Template di un programma

```
#-----  
# Program      :  
# Written by   :  
# Date        :  
# Description:  
#-----  
  
# DATA Segment  
.data  
value1:        .word 256  
  
# CODE Segment  
.text  
main:  
    # First instruction of the main file  
    li $v0, 10
```

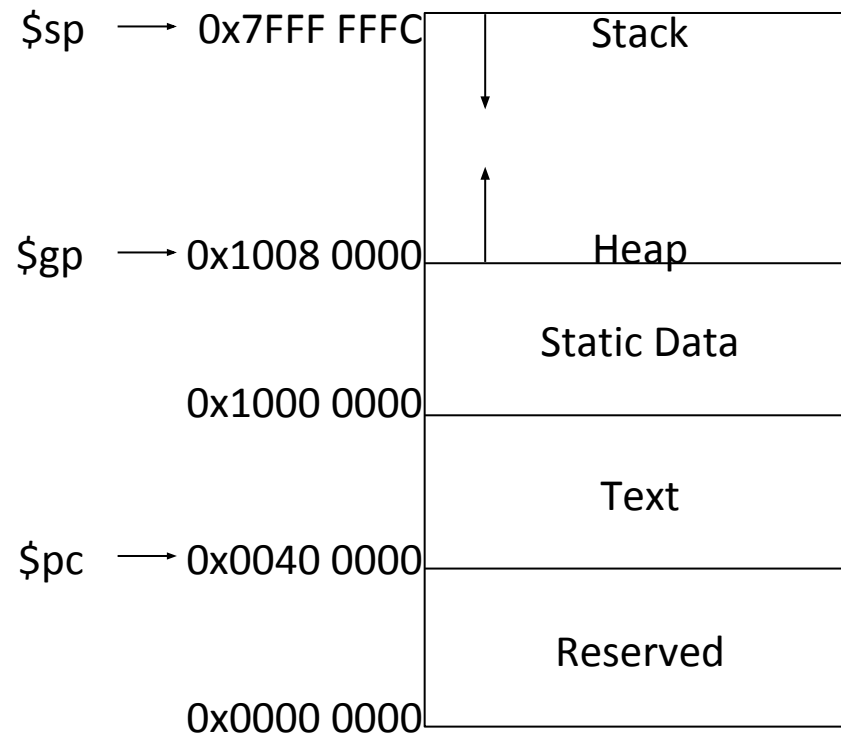
Memoria di un programma

- **Architettura MIPS a 32 bit**

- Memoria indirizzabile □ 4 GB

- **Struttura della memoria**

- Ogni segmento viene allocato in una posizione di memoria predeterminata



Dichiarazione di dati

- Il formato per la dichiarazione di dati è:

nome: tipo_dato valore(i)

- I tipi di dato principali sono:

- `.word` memorizza il dato in 32 bit (4 bytes)
- `.space` specifica il numero di bytes da utilizzare
- `.ascii` memorizza la stringa e aggiunge il terminatore di stringa (`'\0'`)
- `.float` memorizza il dato come numero a precisione singola (32 bit)
- `.double` memorizza il dato come numero a precisione doppia (64 bit)
- `.byte` memorizza il dato come singolo byte (8 bit)

```
var1:    .word    3          # crea una singola variabile intera con valore
                               # iniziale 3

array1:  .byte    'a','b'    # crea un array di caratteri da due elementi
                               # inizializzato ad a e b

array2:  .space    40        # vengono allocati 40 bytes consecutive, senza inizializzare
                               # lo spazio, che può essere usato come array di
                               # caratteri ma anche come array di interi. E'
                               # consigliato commentare specificando cosa si
                               # dovrebbe memorizzare!
```

Istruzioni

- Le istruzioni supportate da MARS possono essere divise in tre diverse categorie:
 - *Istruzioni standard* istruzioni nativamente supportate dall'architettura MIPS
 - *Pseudoistruzioni standard* istruzioni non supportate nativamente dall'architettura, ma che fanno parte dello standard
 - *Pseudoistruzioni estese* istruzioni non supportate nativamente dall'architettura, definite dal simulatore come utilità

Pseudoistruzioni standard

- Nella tabella qui sotto, la lista delle pseudoistruzioni standard MIPS

Pseudo instruction	
bge	rx,ry,imm
bgt	rx,ry,imm
ble	rx,ry,imm
blt	rx,ry,imm
la	rx,label
li	rx,imm
move	rx,ry
nop	

Ad esempio, la pseudoistruzione

```
blt $8, $9, label
```

verrà tradotta nelle seguenti istruzioni

```
slt $1, $8, $9  
bne $1, $zero, label
```

- **Reference istruzioni MIPS:**

<http://www.mrc.uidaho.edu/mrc/people/jff/digital/MIPSir.html>

https://inst.eecs.berkeley.edu/~cs61c/resources/MIPS_Green_Sheet.pdf

Traduzione - IF

Linguaggio C

```
AA;  
if( COND ) {  
    BB;  
}  
CC;
```

Assembly MIPS

```
main:  
    AA  
    COND  
    beq $s0, $zero, end  
    BB  
end:  
    CC
```

Traduzione - IF ELSE

Linguaggio C

```
AA;  
if( COND ) {  
    BB;  
} else {  
    CC;  
}  
DD;
```

Assembly MIPS

```
main:  
    AA  
    COND  
    beq $s0, $zero, else  
    BB  
    j end  
else:  
    CC  
end:  
    DD
```

Traduzione - WHILE

Linguaggio C

```
AA;  
while( COND ) {  
    BB;  
}  
CC;
```

Assembly MIPS

```
main:  
    AA  
cond:  
    COND  
    beq $s0, $zero, end  
    BB  
    j cond  
end:  
    CC
```

Traduzione - DO WHILE

Linguaggio C

```
AA;  
do {  
    BB;  
} while( COND );  
CC;
```

Assembly MIPS

```
main:  
    AA  
do:  
    BB  
    COND  
    bne $s0, $zero, do  
    CC
```

Traduzione - FOR

Linguaggio C

```
AA;  
for (INIT; COND; INC) {  
    BB;  
}  
CC;
```

Assembly MIPS

```
main:  
    AA  
init:  
    INIT  
cond:  
    COND  
    beq $s0, $zero, end  
    BB  
inc:  
    INC  
    j cond  
end:  
    CC
```

Chiamata a funzione

- **Esistono delle strutture hardware a supporto delle chiamate a funzione**
 - JAL Istruzione per fare il salto a funzione
 - JR Istruzione per saltare all'indirizzo contenuto in un indirizzo
 - \$ra Registro contenente il return address
 - \$a0 - \$a3 Registri argomento
 - \$v0 - \$v1 Registri di ritorno

Salvataggio di contesto

- Durante la chiamata a funzione può rendersi necessario salvare alcuni registri (contesto) per far sì che il chiamante continui a funzionare correttamente

