

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

A. Canidio

e-mail: andrea.canidio@mail.polimi.it

material: github.com/acanidio/polimi_cr_acso_2018

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

Simulatore MARS

- **Il simulatore MARS**

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

- **Caratteristiche**

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

Struttura di un programma

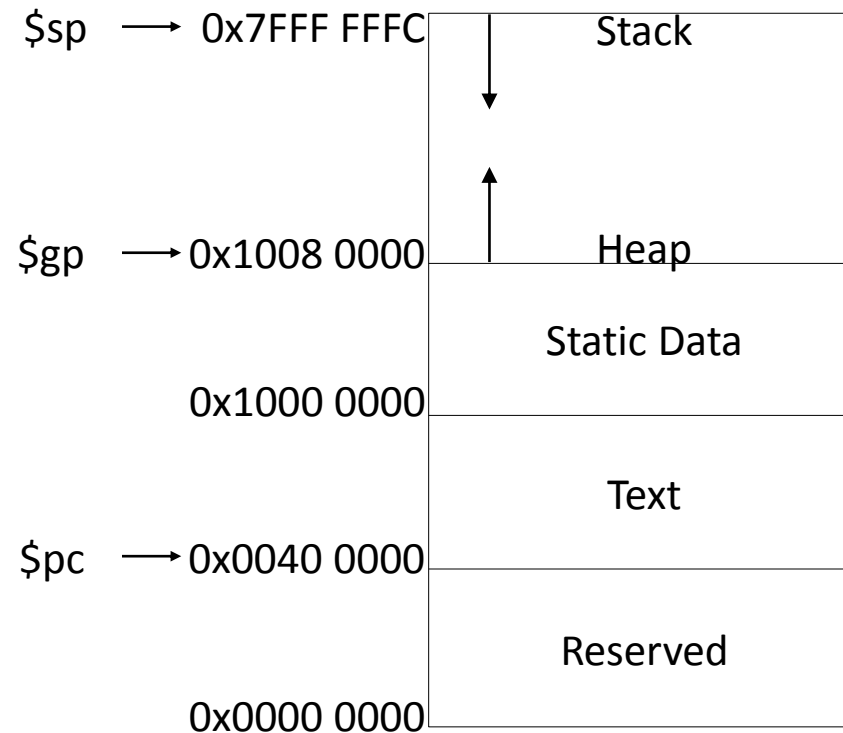
- File testuali con dichiarazione di dati, istruzioni (estensione .asm per MARS)
- Sezione di dichiarazione dati seguita dalla sezione istruzioni
- **Dichiarazione dati (.data, 0x10000000)**
 - Posizionata in una sezione identificata dalla direttiva `.data`
 - Dichiara i nomi delle variabili usate dal 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 cio' che è seguito da un `#`
`# questo è considerato un commento`

Template di un programma

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

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      # alloca 40 bytes consecutive, senza initializzare
                        # 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	

Traduzione - IF

Linguaggio C

```
A
if( cond ) {
    B
}
C
```

Assembly MIPS

```
τ (A)
τ (cond)
BNEZ cond, endif
τ (B)
end: τ (C)
```

Traduzione – IF ... ELSE

Linguaggio C

```
A
if( cond ) {
    B
}
else {
    C
}
D
```

Assembly MIPS

```
τ (A)
τ (cond)
BNEZ cond, else
τ (B)
B      end
else: τ (C)
end:   τ (D)
```

Traduzione – WHILE

Linguaggio C

```
A
while( cond ) {
    B
}
C
```

Assembly MIPS

```
      τ (A)
cond: τ (cond)
      BNEZ  cond, end
      τ (B)
      B     cond
end:   τ (C)
```