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/brandole

Teaching Assistant

D. Iezzi

e-mail: domenico.iezzi [at] polimi [dot] 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
```

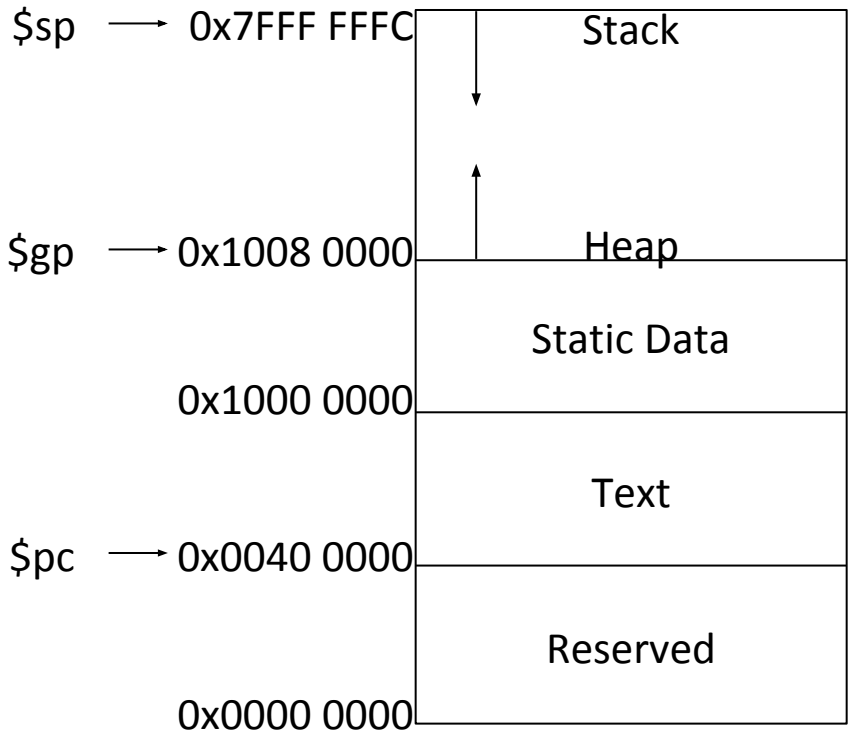
Template di un programma

```
#---------------------------------------------------------------
# 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
- `.asciiz` 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

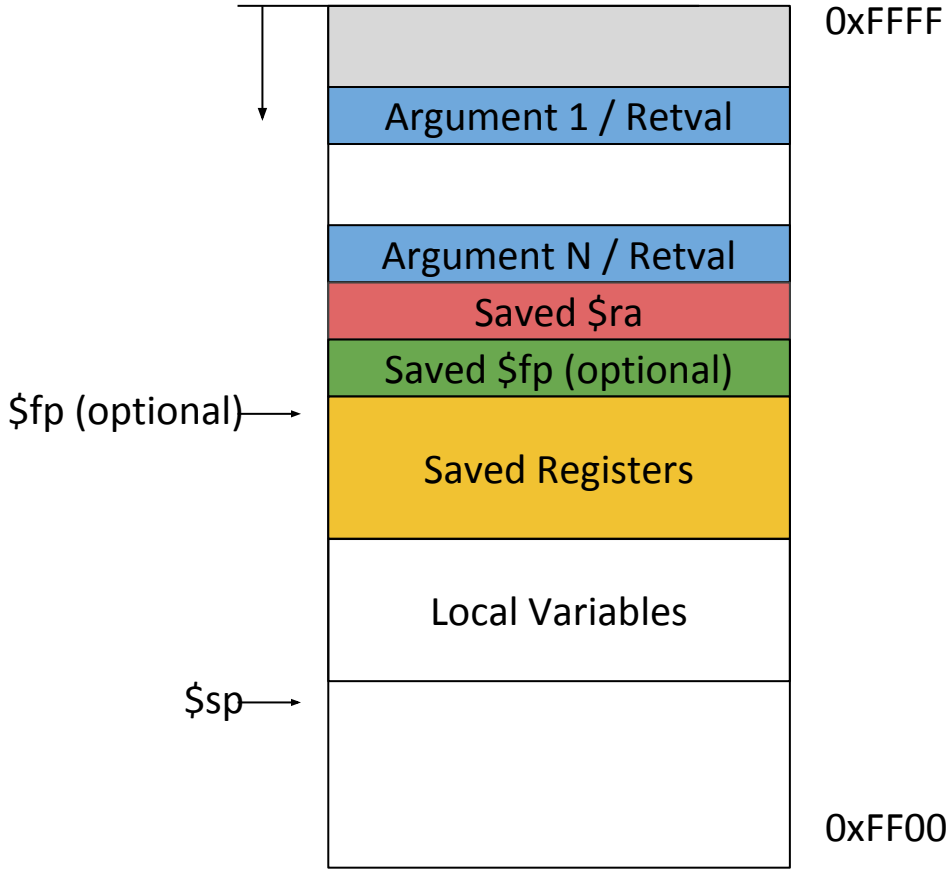
```
main:
    AA
init:
    INIT
cond:
    COND
    beq $s0, $zero, end
    BB
inc:
    INC
    j cond
end:
   CC
```

Chiamata a funzione

- **Esistono delle strutture hardware a supporto delle chiamate a funzione**
  - JAL **Jump and link**: effettua il jump all'indirizzo e salva l'indirizzo di ritorno nel registro $ra
  - JR Istruzione per saltare incondizionatamente all'indirizzo fornito
  - $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 si che il chiamante continui a funzionare correttamente



Salvataggio di contesto

- **Si possono distinguere tre tipologie di funzione**
  - ***Simple leaf:*** funzione che non utilizza stack e non chiama altre funzioni
  - ***Leaf with Data:*** è una leaf che richiede l'utilizzo della stack, per salvare variabili locali o registri
  - ***Non-leaf:*** funzione che chiama altre funzioni
- **Si utilizza la seguente convenzione per il salvataggio di contesto**
  - La funzione *chiamante* salva se necessario i registri **$t0-$t9 $a0-$a3 $v0-$v1**
  - La funzione *chiamata* si occuperà di salvare i registri **$s0-$s7**

Syscall

- **È possibile utilizzare servizi di sistema tramite delle apposite system call. Alcuni esempi**

Servizio di sistema	Codice operativo	Argomenti
Stampa stringa	4	$a0 -> indirizzo della stringa terminata da un carattere nullo
Leggi intero	5	$v0 conterrà l'intero letto in input
Exit (termina esecuzione)	10	
Sleep	32	$a0 -> tempo in millisecondi
MIDI output sincrono	33	$a0 = intonazione (0-127) $a1 = durata in millisecondi $a2 = strumento (0-127) $a3 = volume (0-127)

- **Elenco completo:** http://courses.missouristate.edu/kenvollmar/mars/Help/SyscallHelp.html

Syscall

- **Per utilizzare una system call sono necessari i seguenti passi**
  a. Caricare il codice operativo del servizio nel registro $v0.
  b. Se necessari caricare gli argomenti della syscall nei registri $a0, $a1, $a2, $a3
  c. Chiamata istruzione SYSCALL
  d. Prendere il valore di ritorno dal registro $v0 e/o $v1
- **Esempio**

```
.data

array:      .asciiz     "Ciao mondo!"

.text
            la  $a0, array
            addi  $v0, $zero, 4
            syscall
            addi  $v0, $zero, 10
            syscall
```