

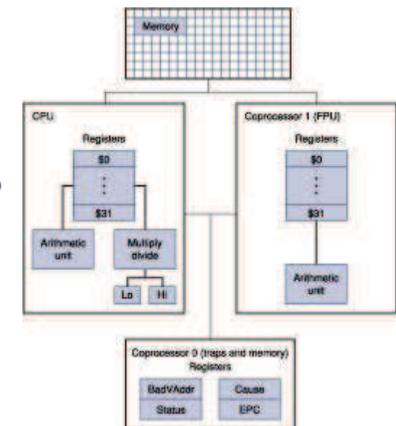
## Architetture dei Calcolatori

### SPIM

Prof. Francesco Lo Presti

## SPIM

- Simulatore che esegue programmi assembler per architetture RISC MIPS R2000/R3000
  - Legge programmi in assembler MIPS e li traduce in linguaggio macchina
    - ✓ Completo di Direttive e Pseudoistruzioni
  - Esegue le istruzioni macchina
  - Mostra il contenuto dei registri e della memoria
  - Permette il debugging dei programmi
  - Fornisce servizi elementari OS-like, come I/O da console



SPIM

2

## Interfaccia Grafica

- Il simulatore SPIM si presenta organizzato in due finestre visuali, una chiamata **console** che funge da terminale per le operazioni di input/output e l'altra che mostra lo stato del **processore MIPS** e della **memoria** organizzata in quattro pannelli:
  - Il display dei registri
  - Il segmento di testo
  - Il segmento di dati e dello stack
  - Il pannello dei messaggi

```
PC = 00000000 EPC = 00000000 Cause = 00000000 BadVAddr= 00000
Status = 00000000 HI = 00000000 LO = 00000000
General Registers
R0 (r0) = 00000000 R8 (t0) = 00000000 R16 (s0) = 00000000 R24 (t8) = 00000000
R1 (at) = 00000000 R9 (t1) = 00000000 R17 (s1) = 00000000 R25 (t9) = 00000000
[0x00400000] 0x8fa40000 lw $4, 0($29) ; 140: lw $a0, 0($sp)
[0x00400004] 0x27a50004 addiu $5, $29, 4 ; 141: addiu $a1, $sp,
[0x00400008] 0x24a60004 addiu $6, $5, 4 ; 142: addiu $a2, $a1,
[0x0040000c] 0x00041080 sll $2, $4, 2 ; 143: sll $v0, $a0, 2
[0x00400010] 0x00c23021 addu $6, $6, $2 ; 144: addu $a2, $a2,
DATA
[0x10000000]...[0x10040000] 0x00000000
STACK
All Rights Reserved.
DOS and Windows ports by David A. Carley (dac@cs.wisc.edu).
Copyright 1997 by Morgan Kaufmann Publishers, Inc.
See the file README for a full copyright notice.
Loaded: C:\Programmi\PCSpim\trap.handler
For Help, press F1 PC=0x00000000 EPC=0x00000000 Cause=0x00000000
```

SPIM

4

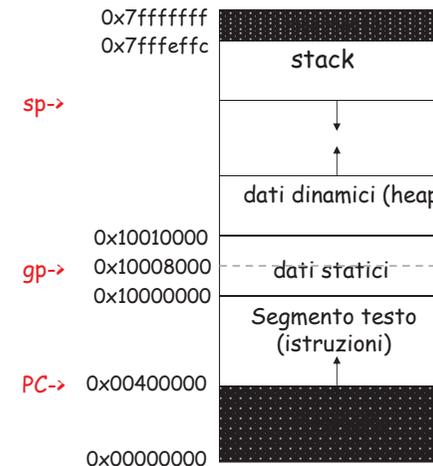
SPIM

3

## Il display dei registri

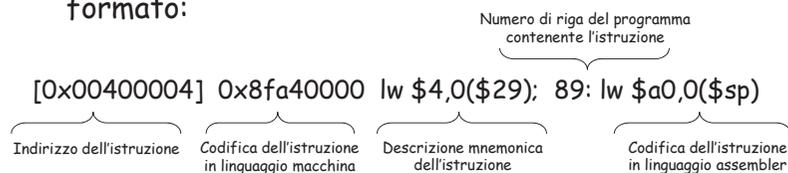
- Mostra lo stato di tutti i registri della CPU e della FPU del MIPS.
- Il contenuto dei registri è codificato in **esadecimale** ed è aggiornato ogni volta che il programma interrompe la sua esecuzione, ovvero quando il programma
  - termina;
  - produce un errore;
  - raggiunge un *breakpoint* definito dall'utente.

## Uso della memoria in SPIIM



## Il segmento di testo

- Mostra la porzione di memoria in cui è memorizzato il programma da eseguire (a partire dalla locazione 0x00400000).
- Tale programma è costituito dal codice utente più una parte di codice di sistema che si occupa della gestione della riga di comando e della terminazione.
- Una generica riga di questo segmento ha il seguente formato:



## Segmento di dati e stack

- Mostra lo stato della memoria riservata ai dati (segmento dati) e allo stack del programma.
  - Per facilitarne la lettura, ogni rigo di tali segmenti mostra il contenuto (in esadecimale) di 4 locazioni di memoria consecutive (1 locazione ~ 1 word ~ 4 byte ~ 1 registro)
    - Queste locazioni sono ordinate in maniera crescente da sinistra verso destra.
    - Ogni locazione, essendo questa una word e poiché il MIPS indirizza il singolo byte, rappresenta a sua volta 4 locazioni consecutive che sono invece ordinate da destra verso sinistra
- ✓ endianness dipende dall'architettura Intel su cui gira PCSPIM

## SPIM: Comandi

- Apre un file sorgente
  - File -> Open
- Esecuzione Programmi
  - Simulator -> Set Value... : per impostare il valore di registri/memoria
    - ✓ Per far partire un programma impostare PC= "0x00400000"
      - inizio segmento testo
  - Simulator -> Go : esegue il programma caricato
  - Simulator -> Break : interrompe l'esecuzione
  - Simulator -> Clear Registers e Reinitialize : reset registri/memoria per nuovi run
  - Simulator -> Reload : Rilegge file sorgente
  - Simulator -> Single Step/Multiple Step : per esecuzione passo/passo
  - Simulator -> Breakpoints : imposta breakpoints
- Nota:
  - Se Load trap file e' impostato in Simulator -> Settings... e' presente anche il codice per la gestione delle eccezioni

SPIM

9

## Struttura di un programma SPIM

### □ Formato tipico

```
.text          #code section
               #starting point: must be global
.globl main
main:
# user program code
.data         #data section
# user program data
```

SPIM

10

## Direttive

- Le direttive forniscono all'assemblatore delle indicazioni sul contenuto di un file (istruzioni, strutture dati, ...)
- Sintatticamente, le direttive iniziano con il carattere "."
  - .text <addr>
    - ✓ Memorizza gli elementi successivi nel segmento testo dell'utente, a partire dall'indirizzo addr (questi elementi possono essere solo istruzioni)
  - .data
    - ✓ Gli elementi successivi alla direttiva sono memorizzati nel segmento dati
  - .globl sym
    - ✓ Dichiaro sym come etichetta globale (ad essa è possibile fare riferimento da altri file)

SPIM

11

## Direttive

### □ Principali direttive per l'allocazione di dati inizializzati

- .word w1 ... wn
  - ✓ Memorizza gli n valori su 32 bit w1, ..., wn in parole consecutive
- .byte b1 ... bn
  - ✓ Memorizza gli n valori b1, ..., bn in byte consecutivi
- .ascii str
  - ✓ Memorizza la stringa str terminandola con il carattere Null (ascii str ha lo stesso effetto, ma non aggiunge Null)
- .space n
  - ✓ Alloca uno spazio di n byte nel segmento dati
- .align n
  - ✓ Allinea il dato successivo a blocchi di 2<sup>n</sup> byte

SPIM

12

## Etichette

- Introduce un identificatore e lo associa al punto del programma (indirizzo) a cui si riferisce
  - Consiste di un identificatore seguito dal simbolo :
    - ✓ Main, loop, Array
- Può avere visibilità
  - Locale: referenziabile solo all'interno del file in cui è definita
  - Globale: referenziabile anche da altri file
    - ✓ Richiede .globl
- Gli identificatori possono essere usati nelle istruzioni/dati per fare riferimento alla posizione di programma associata

SPIIM 13

## MIPS Assembler: Modalità di indirizzamento memoria per operazioni load/store

- ISA MIPS supporta solo: c(rx) (lw \$t0, 4(\$s0))
  - L'assembler MIPS supporta modalità d'indirizzamento più flessibili tramite pseudoistruzioni:
    - Formato
    - Indirizzo
- |                                  |  |
|----------------------------------|--|
| 1. (registro)                    | 1. Contenuto registro  |
| 2. Immediato                     | 2. Valore immediato  |
| 3. Immediato(registro)           | 3. Valore immediato+contenuto registro                       |
| 4. Etichetta                     | 4. Indirizzo etichetta                                       |
| 5. Etichetta±immediato           | 5. Indirizzo etichetta±valore immediato                      |
| 6. Etichetta±immediato(registro) | 6. Indirizzo etichetta±(valore immediato+contenuto registro) |

SPIIM 15

## Direttive/Etichette: esempi

```
# Sommare le variabili x ed y

.text
.globl main

main: lw $t1, x           # legge x in $t1
      lw $t2, y           # legge y in $t2
      add $t0, $t1, $t2   # calcola somma
      sw $t0, z           # salva il risultato in z
      jr $ra              # ritorna

.data
x: .word 5                # variabile x
y: .word 7                # variabile y
z: .word 0                # variabile z
```

SPIIM 14

## Direttive/Etichette: esempi

```
# Somma valori in un array

.text
.globl main

main: li $s0,10            # load immediate (pseudoistruzione)
      la $s1, array       # load address (pseudoistruzione)
      add $s2, $zero, $zero # contatore ciclo
      add $t2, $zero, $zero # somma=0

loop: lw $t1, 0($s1)      # accesso all'array
      add $t2, $t1, $t2   # incremento somma
      addi $s1, $s1, 4    # $s1: indice del prossimo elemento
      addi $s2, $s2, 1    # incremento contatore ciclo
      bne $s2, $s0, loop  # test termine ciclo
      jr $ra              # ritorna

.data
array: .word 1 2 3 4 5 6 7 8 9 10 # dichiarazione array
      # array rappresenta l'indirizzo del primo elemento
```

SPIIM 16

## Chiamate di Sistema SPIM

### □ Chiamate di Sistema: syscall

- Servizi OS-like

### □ Invocazione

- Caricare il codice chiamato nel registro \$v0 (vedi tabella prossimo lucido)
- Caricare gli argomenti nei registri \$a0, ..., \$a3
- Eseguire l'istruzione `syscall`
- La chiamata restituisce un valore nel registro \$v0, o \$f0 nel caso di risultati floating point

SPIM 17

## Codici Chiamate di Sistema SPIM

Service	Code (put in \$v0)	Arguments	Result
print_int	1	\$a0=integer	
print_float	2	\$f12=float	
print_double	3	\$f12=double	
print_string	4	\$a0=addr. of string	
read_int	5		int in \$v0
read_float	6		float in \$f0
read_double	7		double in \$f0
read_string	8	\$a0=buffer, \$a1=length	
sbrk	9	\$a0=amount	addr in \$v0
exit	10		

SPIM 18

## Chiamate di Sistema: Esempio

```
# Lettura intero da tastiera

.text
.globl main
main:
    li $v0,4          # codice per print_string
    la $a0, Stringa  # indirizzo stringa
    syscall          # chiamata di sistema
    li $v0,5          # codice per read_int
    syscall          # chiamata di sistema
    jr $ra           # ritorna

.data
Stringa: .asciiz "Inserisci un numero" # Stringa da stampare
```

SPIM 19