Lezione 13

Assembly III
SPIM: un ambiente di simulazione MIPS

Proff. A. Borghese, F. Pedersini

Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano

- SPIM: A MIPS R2000/R3000 Simulator:
  http://www.cs.wisc.edu/~larus/spim.html
- version 7.4
  - Unix or Linux system / Mac OS/X
  - Microsoft Windows (Windows 95, 98, NT, 2000)
  - Microsoft DOS
- Riferimento:
  - Patterson, Hennessy:
    Appendix A (Assemblers, Linkers, and the SPIM Simulator)

SPIM / XSPIM

Comandi principali:

load | read "FILE"
carica un programma in memoria

run
lancia il programma caricato

step <N>
esegue N istruzioni di programma

clear
reset registri o memoria

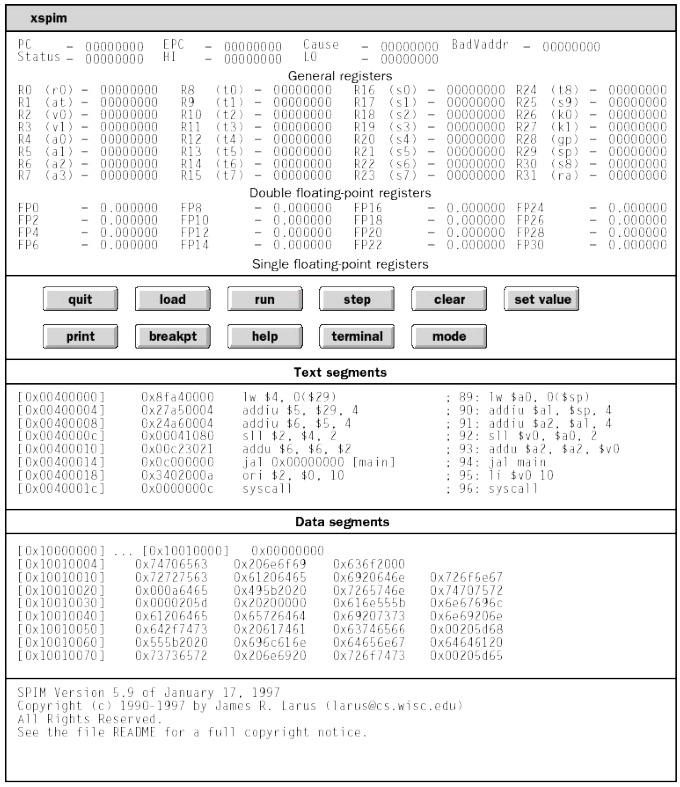
set value
inserisce valori in registri o memoria

print
stampa valore di registri o memoria

breakpoint, delete
inserisce/toglie breakpoints

help

quit | exit



SPIM / XSPIM – comandi

Da "help" di SPIM:

- reinitialize -- *Clear the memory and registers*
- load / read "FILE" -- *Read FILE of assembly code into memory*
- run <ADDR> -- *Start the program at optional ADDRESS*
- step <N> -- *Step the program for N instructions*
- continue -- *continue program execution without stepping*

- print -- *print value of a register or a memory location*
  - print $N | $fN -- *Print register N | floating point register N*
  - print ADDR -- *Print contents of memory at ADDRESS*
  - print_symbols -- *Print all global symbols*
  - print_all_regs -- *Print all MIPS registers*
  - print_all_regs hex -- *Print all MIPS registers in hex*

- breakpoint <ADDR> -- Set a breakpoint at address
- delete <ADDR> -- Delete all breakpoints at address
- list -- List all breakpoints

- exit,quit -- Exit the simulator

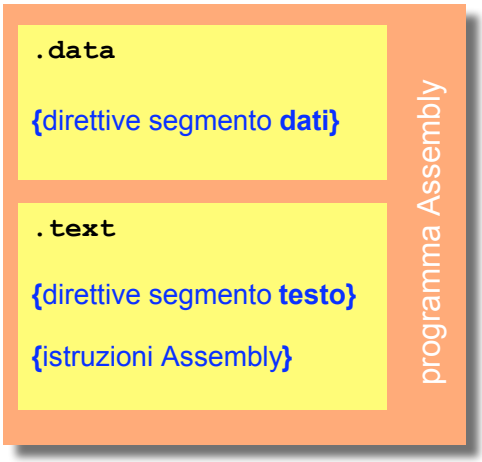
Abbreviazioni: basta il prefisso unico. ex(it), re(ad), l(oad), ru(n), s(tep), p(rint)

Sommario

- Direttive
- Chiamate di sistema (system call)
- Esempi

Direttive

- Definizione:
  le direttive danno delle indicazioni all'assemblatore:
  - su come memorizzare il programma e i dati in memoria
  - su come eseguire il programma Assembly
- Sintassi: le direttive iniziano con il carattere "."
- Classificazione:
  - direttive relative al **segmento dati**
  - direttive relative al **segmento testo**
- Struttura programma Assembly:
  1. **segmento dati:**
     - contiene le dichiarazioni di variabili e costanti
     - inizia con la direttiva: `.data`
  2. **segmento testo:**
     - contiene il programma
     - inizia con la direttiva: `.text`



Direttive

- **Direttive – segmento dati:**
  - Definiscono regole di allocazione di dati in memoria

`.data <addr>`
- Gli elementi successivi sono memorizzati nel segmento dati a partire dall'indirizzo **`addr`** (facoltativo)

`.asciiz str`
- Memorizza la stringa **`str`** terminandola con il carattere **`Null (0x00)`**
- `.ascii str` ha lo stesso effetto, ma non aggiunge alla fine il carattere **`Null`**

`.byte b1, ... ,bn`
- Memorizza gli n valori **`b1, .., bn`** in **bytes consecutivi** di memoria

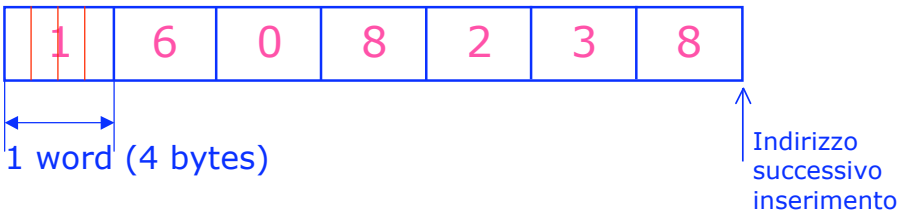
`.word w1, ... ,wn`
- Memorizza gli **`n`** valori su 32-bit **`w1, .., wn`** in parole **(words) consecutive** di memoria

`.half h1, ... ,hn`
- Memorizza gli **`n`** valori su 16-bit **`h1, .., hn`** in **halfword** (mezze parole) **consecutive** di memoria

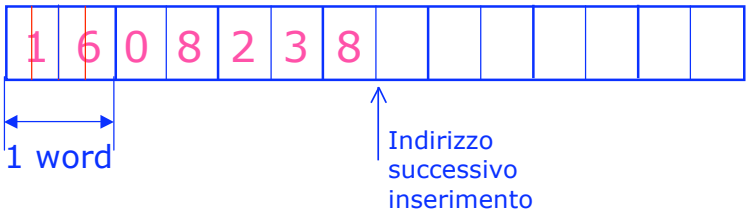
`.space n`
- Alloca uno spazio pari ad **`n`** bytes nel segmento dati

Allineamento: esempio

`.word 1, 6, 0, 8, 2, 3, 8`

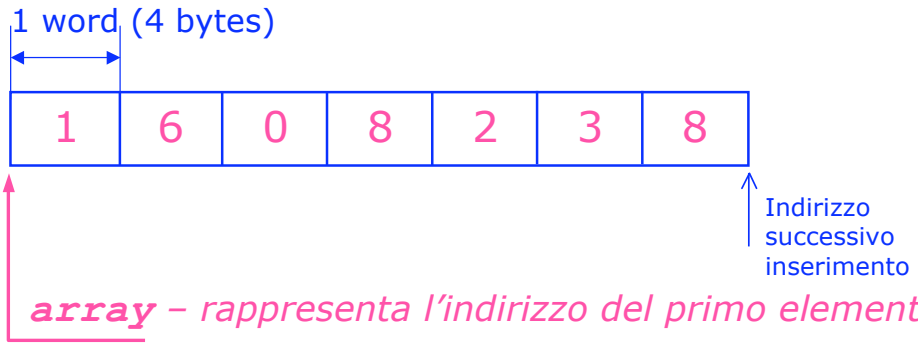


`.half 1, 6, 0, 8, 2, 3, 8`



Allineamento: `.word + label`


```
array: .word 1, 6, 0, 8, 2, 3, 8
```



array *– rappresenta l'indirizzo del primo elemento*

Direttive

- (continua) … direttive: segmento **dati**

`.align n`

- Allinea il dato successivo a blocchi di 2^n byte: ad esempio:
- `.align 2 = .word` : allinea alla parola il valore successivo
- `.align 1 = .half` : allinea alla mezza parola il valore successivo
- `.align 0` : elimina l'allineamento automatico delle direttive `.half, .word, .float, .double`, fino a quando compare la successiva direttiva `.data`

- Direttive: segmento **testo**

`.text <addr>`

- Memorizza gli elementi successivi nel segmento testo dell'utente a partire dall'indirizzo `addr`
  Questi elementi possono essere solo istruzioni o parole.

`.globl sym`

- Dichiara `sym` come etichetta globale (ad essa è possibile fare riferimento da altri file)

Sommario

- Direttive
- Chiamate di sistema (system call)
- Esempi

System call

- Sono disponibili delle **chiamate di sistema (`system call`)** predefinite, che implementano particolari servizi
  - Input/Output (ad esempio: stampa a video)
  - Allocazione dinamica
  - Ritorno al S.O. a fine programma
- Ogni **`system call`** ha:
  - un codice
  - degli argomenti (opzionali)
  - dei valori di ritorno (opzionali)

System call: output

1: print_int:

- stampa sulla console il numero intero che le viene passato come argomento;

2: print_float:

- stampa sulla console il numero in virgola mobile con singola precisione (IEEE 754) che le viene passato come argomento;

3: print_double:

- stampa sulla console il numero in virgola mobile con doppia precisione (IEEE 754) che le viene passato come argomento;

4: print_string:

- stampa sulla console la stringa che le è stata passata come argomento terminandola con il carattere `Null (0)`;

System call: input

5: read_int:

- legge una linea in ingresso fino al carattere a capo incluso (i caratteri che seguono il numero sono ignorati) e la interpreta come numero intero (32 bit);

6: read_float:

- leggono una linea in ingresso fino al carattere a capo incluso (i caratteri che seguono il numero sono ignorati)
  e la interpreta come numero in virgola mobile – singola precisione (32 bit);

7: read_double:

- leggono una linea in ingresso fino al carattere a capo incluso (i caratteri che seguono il numero sono ignorati)
  e la interpreta come numero in virgola mobile – doppia precisione (64 bit);

8: read_string:

- legge una stringa di caratteri di lunghezza `$a1` da una linea in ingresso, scrivendoli in un buffer `($a0)` e terminando la stringa con il carattere `Null`
  (se ci sono meno caratteri sulla linea corrente, li legge fino al carattere a capo incluso e termina la stringa con il carattere `Null`);

System call: **controllo**

9: sbrk

- alloca in memoria dati un blocco di memoria di **n** bytes, restituendo l'indirizzo di inizio blocco;

10: exit

- interrompe l'esecuzione del programma;

System calls

Nome	Codice ($v0)	Argomenti	Risultato
print_int	1	$a0	
print_float	2	$f12	
print_double	3	$f12	
print_string	4	$a0	
read_int	5		$v0
read_float	6		$f0
read_double	7		$f0
read_string	8	$a0,$a1	
sbrk	9	$a0	$v0
exit	10		

System calls

- Per richiedere un servizio ad una chiamata di sistema (**`syscall`**) occorre:
  - Caricare il **codice** della **`syscall`** nel registro **`$v0`**
  - Caricare gli **argomenti** nei registri **`$a0 ÷ $a3`**
    - oppure nei registri **`$f12 ÷ $f15,`** nel caso di valori in virgola mobile
  - Chiamare la procedura: **`syscall`**
  - L'eventuale **valore di ritorno** è caricato nel registro **`$v0`**
    - **`$f0`**, nel caso di valori in virgola mobile

Uso dei registri: convenzioni

Registri "general purpose"

Nome	Numero	Utilizzo
$zero	0	costante zero
$at	1	riservato per l'assemblatore
$v0-$v1	2-3	valori di ritorno di una procedura
$a0-$a3	4-7	argomenti di una procedura
$t0-$t7	8-15	registri temporanei (non salvati)
$s0-$s7	16-23	registri salvati
$t8-$t9	24-25	registri temporanei (non salvati)
$k0-$k1	26-27	gestione delle eccezioni
$gp	28	puntatore alla global area (dati)
$sp	29	stack pointer
$s8	30	registro salvato (fp)
$ra	31	indirizzo di ritorno

Registri usati per le operazioni floating point

Nome	Utilizzo
$f0-$f3	valori di ritorno di una procedura
$f4-$f11	registri temporanei (non salvati)
$f12-$f15	argomenti di una procedura
$f16-$f19	registri temporanei (non salvati)
$f20-$f31	registri salvati

Sommario

- Direttive
- Chiamate di sistema (system call)
- Esempi

Esempio

```
# Programma che stampa "Dammi un intero: "
# e che legge un intero

          .data
prompt:   .asciiz "Dammi un intero: "

          .text
          .globl main

main:     li $v0, 4       # $v0 ← codice della print_string
          la $a0, prompt  # $a0 ←  indirizzo della stringa
          syscall         # stampa la stringa

          li $v0, 5       # $v0 ← codice della read_int
          syscall         # legge un intero e lo carica in $v0

          li $v0, 10      # $v0 ← codice della exit
          syscall         # esce dal programma
```

Esempio

```
# Programma che stampa: "La risposta è 5"
#
        .data
str:    .asciiz "La risposta è "

        .text
        .globl main

Main:   li $v0, 4           # $v0 ← codice della print_string
        la $a0, str         # $a0 ← indirizzo della stringa
        syscall             # stampa della stringa

        li $v0, 1           # $v0 ← codice della print_integer
        li $a0, 5           # $a0 ← intero da stampare
        syscall             # stampa dell'intero

        li $v0, 10          # $v0 ← codice della exit
        syscall             # esce dal programma
```


Esempio

- Calcolo somma dei quadrati da 1 a n:

- Codice C:

```
int main(int argc, char* argv[])
{ int i, n = 10, t = 0;
  for (i=1;i<=n;i++)
        t += i*i;
  printf("La somma vale %d\n", t );
  return(0);
}
```

Esempio: codice Assembly

```
# Programma che somma i quadrati dei primi N numeri
# N e' memorizzato in t1

        .data
str:    .asciiz "La somma vale "

        .text
        .globl main

main:   li      $t1, 10         # N interi di cui calcolare la somma quadrati
        li      $t6, 1          # t6 e' indice di ciclo indice di ciclo
        li      $t8, 0          # t8 contiene la somma dei quadrati
Loop:   mult    $t7, $t6, $t6   # t7 = t6 x t6
        addu    $t8, $t8, $t7   # t8 = t8 + t7
        addi    $t6, $t6, 1
        ble     $t6, $t1, Loop  # if  t6 ≤ N  stay in loop

        la      $a0, str
        li      $v0, 4          # print_string
        syscall

        li      $v0, 1          # print_int
        add     $a0, $t8, $zero
        syscall

        li $v0, 10              # $v0 codice della exit
        syscall                 # esce dal programma
```


Esempio 2: estrazione da array

```
# Programma che copia tramite una procedura
# gli interi positivi contenuti nell'array (elem)
# in un secondo array (pos_el)
```

- Algoritmo:
  1. Inizializzazione
  2. Leggi gli interi
  3. Individua gli interi positivi
  4. Copia gli interi positivi
  5. Stampa gli interi positivi

Esempio 2 – Versione in linguaggio C

```
main()
{
        int i, k = 0, N=10, elem[10], pos_el[10];

        elem = leggi_interi(N);

        for (i=0; i<N;i++)
              if (elem[i] > 0)
              {
                    pos_el[k] = elem[i];
                    k++;
              }
        printf("Il numero di interi positivi e' %d",k);
        for (i=0; i<k; i++)
              printf(" %d",pos_el[i]);
        exit(0);
}
```


Esempio 2 – estrazione di numeri positivi

```
# Programma che copia tramite una procedura gli interi positivi
# contenuti nell'array (elem) in un secondo array (pos_el)
        .data
elem:   .space 40               # alloca 40 byte per array elem
pos_el: .space 40               # alloca 40 byte per array pos_el
prompt: .asciiz "Inserire il successivo elemento \n"
msg:    .asciiz "Gli elementi positivi sono \n"

        .text
        .globl main

main:   la $s0, elem            # $s0 ← indirizzo di elem.
        la $s1, pos_el          # $s1 ← indirizzo di pos_el
        li $s2, 40              # $s2 ← numero di elementi (in byte)

# Ciclo di lettura dei 10 numeri interi
        add $t0, $s0, $zero     # indice di ciclo
        add $t1, $t0, $s2       # $t1 ← posizione finale di elem

loop:   li $v0, 4               # $v0 ← codice della print_string
        la $a0, prompt          # $a0 ←  indirizzo della stringa
        syscall                 # stampa la stringa prompt
```

Esempio 2 (cont.)

```
# Lettura dell'intero
        li $v0, 5               # $v0 ← codice della read_int
        syscall                 # legge l'intero e lo carica in $v0

        sw $v0, 0($t0)          # memorizza l'intero in elem
        addi $t0, $t0, 4
        bne $t0, $t1, loop

# Fine della lettura, ora prepara gli argomenti per la proc.
        add $a0, $s0, $zero     # $a0 ← ind. array elem.
        add $a1, $s1, $zero     # $a1 ← ind. array pos_el
        add $a2, $s2, $zero     # $a2 ← dim. in byte dell'array

# Chiamata della procedura cp_pos
        jal cp_pos              # restituisce il numero di byte
                                # occupati dagli interi pos in $v0
```


Esempio 2 (cont.)

```
# ciclo di stampa degli elementi positivi
# Gli indirizzi in cui e' racchiuso pos_elem vanno da $s2 a $t2

        add $t2, $s1, $v0       # $t2 ← ind. fine ciclo pos_el
        add $t0, $s1, $zero     # $t0 ← ind. di ciclo

loop_w:
        beq $t2, $t0, exit-main
        li $v0, 1               # $v0 ← codice della print_integer
        lw $a0, 0($t0)          # $a0 ← intero da stampare
        syscall                 # stampa dell'intero
        addi $t0, $t0, 4
        j loop_w

exit-main:
        li $v0, 10      # $v0 ← codice della exit
        syscall         # esce dal programma
```

Esempio 2 (procedure)

```
# Questa procedura esamina l'array elem ($a0) contenente
# $a2/4 elementi
# Resituisce in $v0, il numero di byte occupati dagli interi
# positivi e in pos_el gli interi positivi

cp_pos: addi $sp, $sp, -16     # allocazione dello stack
        sw $s0, 0($sp)
        sw $s1, 4($sp)
        sw $s2, 8($sp)
        sw $s3, 12($sp)
        add $s0, $a0, $zero    # $s0 ← ind. ciclo su elem
        add $s1, $a0, $a2      # $s1 ← ind. fine ciclo su elem
        add $s2, $a1, $zero    # $s2 ← ind. ciclo su pos_el
```


Esempio 2 (procedure)

```
next:   beq $s1, $s0, exit     # se esaminato tutti gli elementi di elem
                               # salta alla fine del ciclo (exit).
        lw $s3, 0($s0)         # carica l'elemento da elem
        addi $s0, $s0, 4       # posiziona sull'elemento succ. di elem
        ble $s3, $zero, next   # se $s3 ≤ 0 va all'elemento succ. di elem
        sw $s3, 0($s2)         # Memorizzo il numero in pos_elem
        addi $s2, $s2, 4       # posiziona sull'elemento succ. di pos_el
        j next                 # esamina l'elemento successivo di elem

exit:   sub $v0, $s2, $a1      # salvo lo spazio in byte dei pos_el
        lw $s0, 0($sp)
        lw $s1, 4($sp)
        lw $s2, 8($sp)
        lw $s3, 12($sp)
        addi $sp,$sp,12        # deallocazione dello stack
        jr $ra                 # restituisce il controllo al chiamante
```