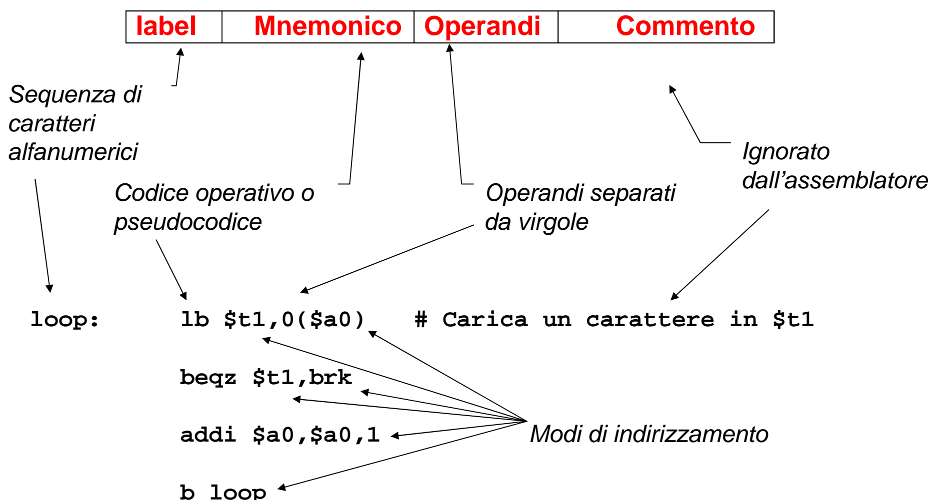
Il linguaggio Assembly del MIPS

Linguaggio macchina
Linguaggio definito da un insieme di istruzioni, codificate come stringhe di bit, che il processore può interpretare ed eseguire direttamente

Linguaggio Assembly
Linguaggio simbolico, vicino al linguaggio macchina, che definisce:

- Uno *mnemonico* per ogni istruzione in L.M.
- Un *formato* per le linee di programma
- Formati per la specifica *della modalità di indirizzamento*
- *Direttive*

Formato istruzioni



Classi di Istruzioni

- Istruzioni aritmetiche
- Istruzioni logiche
- Istruzioni di movimento dati
- Istruzioni di confronto
- Istruzioni per il controllo di flusso

Istruzioni aritmetiche

Istruzione	*Significato*	
add $1,$2,$3	$1 = $2 + $3	*eccezione possibile*
addi $1,$2,100	$1 = $2 + 100	*eccezione possibile*
addu $1,$2,$3	$1 = $2 + $3	*nessuna eccezione*
addiu $1,$2,100	$1 = $2 + 100	*nessuna eccezione*
sub $1,$2,$3	$1 = $2 – $3	*eccezione possibile*
subu $1,$2,$3	$1 = $2 – $3	*nessuna eccezione*
mult $2,$3	Hi, Lo = $2 x $3	*64-bit con segno*
multu $2,$3	Hi, Lo = $2 x $3	*64-bit senza segno*
div $2,$3	Lo = $2 ÷ $3 Hi = $2 mod $3	*Lo = quoziente, Hi = resto*
divu $2,$3	Lo = $2 ÷ $3 Hi = $2 mod $3	*Quoziente & resto senza segno*
sra $1,$2,10	$1 = $2 >> 10	*Shift aritmetico a destra*
srav $1,$2, $3	$1 = $2 >> $3	*Shift aritm. a destra di # bit variabile*

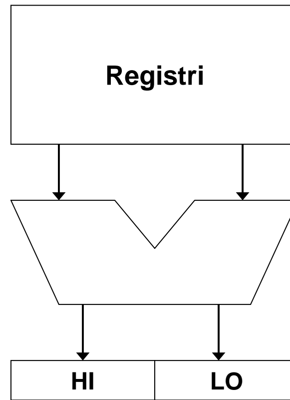
Nota: gli immediati sono valori a 16 bit (con sign extension)

Multiply/Divide

° Avviano l'operazione
- MULT rs, rt
- MULTU rs, rt
- DIV rs, rt
- DIVU rs, rt

° Trasferiscono il risultato
- MFHI rd
- MFLO rd

° Caricano HI e LO
- MTHI rd
- MTLO rd



- Durano più cicli e vengono eseguite in parallelo con l'esecuzione di altre istruzioni, da un'unità indipendente
- I tentativi di accesso prematuro a Lo e Hi sono interlocked
- DIV non verifica /0 e overflow (min neg/-1)
- Le 2 istruzioni che seguono MFHI e MFLO non devono modificare Hi e Lo

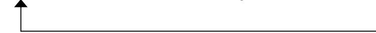
Istruzioni di scorrimento aritmetiche

Uno scorrimento a destra di n bit, può essere visto come una divisione per 2^n:

Es.:

01010110_2 --> 86_{10} con uno scorrimento a destra di 2 bit si ottiene

00010101_2 --> 21_{10} vengono inseriti due 0 a sinistra



Che cosa succede se si considerano numeri signed ?

Es.:

10010110_2 --> -106_{10} con uno scorrimento a destra di 2 bit si ottiene

00100101_2 --> $+37_{10}$ **errato !**

11100101_2 --> -27_{10} vengono inseriti due 1 a sinistra **corretto**

sign extension

Istruzioni logiche

<table>
<tr><th>Istruzione</th><th>Significato</th><th></th></tr>
<tr><td>and $1,$2,$3</td><td>$1 = $2 & $3</td><td>AND bit a bit</td></tr>
<tr><td>andi $1,$2,10</td><td>$1 = $2 & 10</td><td>AND reg, costante</td></tr>
<tr><td>or $1,$2,$3</td><td>$1 = $2 | $3</td><td>OR bit a bit</td></tr>
<tr><td>ori $1,$2,10</td><td>$1 = $2 | 10</td><td>OR reg, costante</td></tr>
<tr><td>xor $1,$2,$3</td><td>$1 = $2 ⊕ $3</td><td>XOR bit a bit</td></tr>
<tr><td>xori $1, $2,10</td><td>$1 = $2 ⊕ 10</td><td>XOR reg, costante</td></tr>
<tr><td>nor $1,$2,$3</td><td>$1 = ~($2 | $3)</td><td>NOR bit a bit</td></tr>
<tr><td>sll $1,$2,10</td><td>$1 = $2 << 10</td><td>Shift a sinistra di # bit costante</td></tr>
<tr><td>sllv $1,$2,$3</td><td>$1 = $2 << $3</td><td>Shift a sinistra di # bit variabile</td></tr>
<tr><td>srl $1,$2,10</td><td>$1 = $2 >> 10</td><td>Shift a destra di # bit costante</td></tr>
<tr><td>srlv $1,$2, $3</td><td>$1 = $2 >> $3</td><td>Shift a destra di # bit variabile</td></tr>
</table>

Uso delle istruzioni logiche

Le istruzioni logiche sono tipicamente utilizzate per accedere e manipolare i singoli bit all'interno delle word.

Es.:

A	1	0	1	0	1	0	0	1	B	0	1	1	1	1	0	1	1	**Dati iniziali**
M	0	0	0	0	1	1	1	1	N	1	1	1	1	0	0	0	0	**Maschere**

A and M --> C

B and N --> D

C or D --> E

C	0	0	0	0	1	0	0	1
D	0	1	1	1	0	0	0	0
E	0	1	1	1	1	0	0	1

Similmente, lo XOR è utile per invertire il valore di singoli bit, mentre le istruzioni di shift possono essere impiegate per le conversioni serie-parallelo.

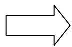
Istruzioni di movimento dati

Il processore MIPS ha due gruppi distinti di istruzioni per i movimenti dati memoria --> registro (load) e registro --> memoria (store).

E' possibile spostare dati di dimensione pari a 1, 2 o 4 byte; in ogni caso, i trasferimenti da/verso la memoria sono sempre da 32 bit.

Per i dati non allineati in memoria esistono apposite istruzioni che permettono il trasferimento tramite trasferimenti parziali successivi, ma lasciando all'utente la responsabilità di costruire la sequenza di istruzioni corretta.

Sono inoltre disponibili istruzioni particolari per lo spostamento da/verso i registri speciali HI, LO.



Non esistono istruzioni per il movimento dati tra registri generali. Perché ? Come si possono realizzare ?

Istruzioni load

Istruzione	***Significato***	
lb $1,*address*	Mem[*address*](8 bit) -> $1	*esteso con segno*
lbu $1,*address*	Mem[*address*](8 bit) -> $1	*esteso con 0*
lh $1,*address*	Mem[*address*](16 bit) -> $1	*esteso con segno*
lhu $1,*address*	Mem[*address*](16 bit) -> $1	*esteso con 0*
lw $1,*address*	Mem[*address*](32 bit) -> $1	
lwl $1,*address*	Mem[*address*] -> $1	*Carica $1 con la parte sinistra della word all'indirizzo non allineato*
lwr $1,*address*	Mem[*address*] -> $1	*Carica $1 con la parte destra della word all'indirizzo non allineato*
lui $1,*costant*	*costant* x 2^{16} -> $1	*Carica una costante da 16 bit nella parte alta del registro, azzerando la parte bassa*

Istruzioni store

Istruzione	*Significato*	
sb $1,*address*	$1(0:8)->Mem[*address*]	
sh $1,*address*	$1(0:15)->Mem[*address*]	
sw $1,*address*	$1(0:31)->Mem[*address*]	
swl $1,*address*	$1->Mem[*address*]	*Carica da $1 la parte sinistra della word all'indirizzo non allineato*
swr $1,*address*	$1->Mem[*address*]	*Carica da $1 la parte destra della word all' indirizzo non allineato*

Caricamento di una costante

Per trasferire il valore di una costante in un registro, esiste l'istruzione lui che assegna alla parte alta di un registro una costante da 16 bit:

Es.:

lui $4,0x1234

$4	1	2	3	4	0	0	0	0

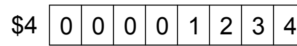
Come fare per caricare una costante da 16 bit nella parte bassa ?

Come fare per caricare una costante da 32 bit nell'intero registro?

Caricamento di una costante (2)

Caricamento di una costante da 16 bit

addi $4,$0,0x1234

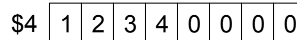


addi $4,$0,-1

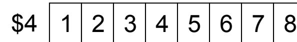
sign extension

Caricamento di una costante da 32 bit (0x12345678)

lui $4,0x1234

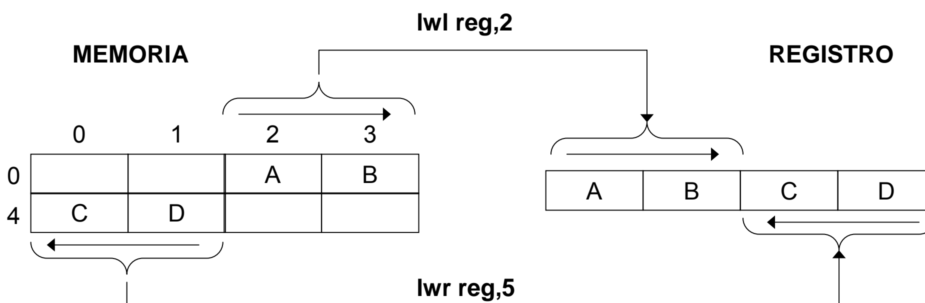


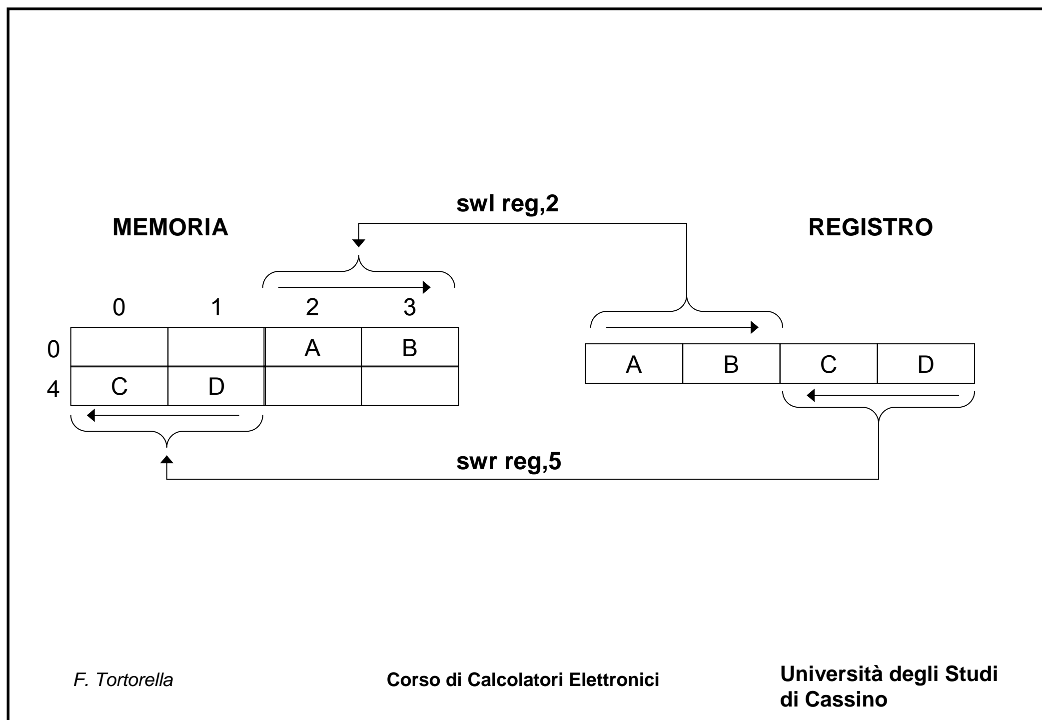
ori $4,$4,0x5678



Istruzioni load/store per dati non allineati

Queste istruzioni si usano in coppia per trasferire un dato da/verso un indirizzo non allineato.





Istruzioni di move da/verso HI,LO

Istruzione	*Significato*	
mfhi $1	$1 = Hi	*Copia Hi in $1*
mflo $1	$1 = Lo	*Copia Lo in $1*
mthi $1	Hi = $1	*Copia $1 in Hi*
mtlo $1	Lo = $1	*Copia $1 in Lo*

Istruzioni di confronto

Istruzione	*Significato*	
slt $1,$2,$3	*if* ($2<$3) $1=1; *else* $1=0	*Confronto tra registri* *(con segno)*
slti $1,$2,100	*if* ($2<100)$1=1; *else* $1=0	*Confronto registro-costante* *(con segno)*
sltu $1,$2,$3	*if* ($2<$3) $1=1; *else* $1=0	*Confronto tra registri* *(senza segno)*
sltiu $1,$2,100	*if* ($2<100) $1=1; *else* $1=0	*Confronto registro-costante* *(senza segno)*

Istruzioni per il controllo di flusso

Sono istruzioni che permettono di alterare il flusso sequenziale di esecuzione delle istruzioni del programma, trasferendo il controllo ad una istruzione diversa da quella che segue immediatamente nel programma ("salto").

Se il salto è realizzato sulla base del verificarsi di una condizione, si parla di *istruzioni di salto condizionato*, altrimenti si definiscono *istruzioni di salto incondizionato.*

A seconda di come viene specificata la destinazione del salto nella codifica in linguaggio macchina dell'istruzione, le istruzioni di salto si dividono in due classi:

- Istruzioni di *branch*
  - la destinazione del salto è specificata tramite uno spiazzamento rispetto al valore del PC.
- Istruzioni di *jump*
  - la destinazione del salto è specificata tramite un indirizzo assoluto.

Istruzioni di branch

Istruzione	*Significato*	
beq $1,$2,label	*if* ($1 == $2) *branch* label	*Test di uguaglianza*
bne $1,$2,label	*if* ($1!= $2) *branch* label	*Test di disuguaglianza*
bgez $1,label	*if* ($1>= $0) *branch* label	*Verifica se il valore in $1 è non negativo*
bgtz $1,label	*if* ($1> $0) *branch* label	*Verifica se il valore in $1 è positivo*
blez $1,label	*if* ($1<= $0) *branch* label	*Verifica se il valore in $1 è non positivo*
bltz $1,label	*if* ($1< $0) *branch* label	*Verifica se il valore in $1 è negativo*

Istruzioni di jump

Istruzione	*Significato*	
j label	*jump* label	*Salta all'istruzione all'indirizzo* label *(indirizzo da 26 bit)*
jr $31	*jump* $31	*Per i ritorni da procedura (indirizzo da 32 bit)*

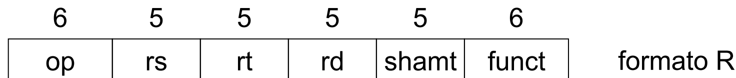
Istruzioni per la chiamata di sottoprogramma

Sono particolari istruzioni per il controllo di flusso. Prima del salto, l’indirizzo dell’istruzione successiva viene salvato nel registro $31.

bgezal $1,label	*if* ($1>= $0) { $31 = PC + 4; *branch* label }	*salto condizionato*
bltzal $1, label	if ($1< $0) { $31 = PC + 4; *branch* label }	*salto condizionato*
jal label	$31 = PC + 4; *jump* label	*salto incondizionato*
jalr $1	$31 = PC + 4; *jump* $1	*salto incondizionato*

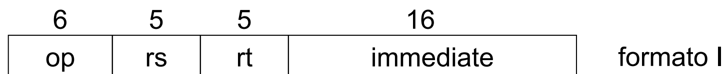
Formati delle istruzioni

La lunghezza delle istruzioni MIPS è fissa (32 bit). Esistono però 3 possibili formati per le istruzioni:



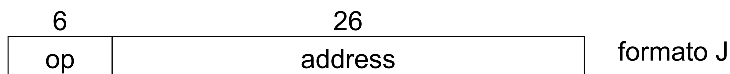
formato R

Usato per le istruzioni aritmetiche



formato I

Usato per le istruzioni di branch



formato J

Usato per le istruzioni di jump

Modi di indirizzamento

Indicano come accedere all'operando di interesse dell'istruzione.
Il MIPS ha 5 modi di indirizzamento:

immediato (immediate)
slti $1,$2,100

registro (register)
slti $1,$2,100

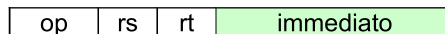
base/spiazzamento (base/displacement)
lw $t0,4($t1)

relativo rispetto al PC (PC-relative)
beq $1,$2,label

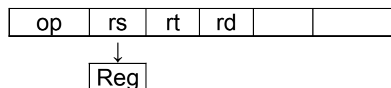
pseudodiretto (pseudo direct)
j label

Modi di indirizzamento

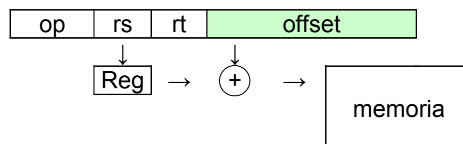
immediato



registro



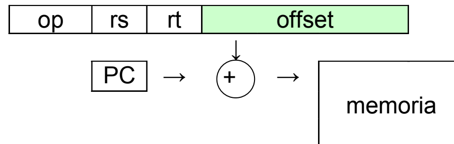
base (+offset)



Modi di indirizzamento

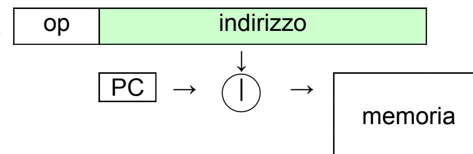
PC-relative (*branch*)

± 128 KB da PC + 4



PC-relative (*jump*)

entro la pagina di 256 MB a partire da PC + 4



Pseudoistruzioni

Sono istruzioni accettate dall'assemblatore MIPS alle quali non corrisponde un effettivo codice operativo in linguaggio macchina.

Prima della traduzione del programma in linguaggio macchina, le pseudo istruzioni vengono espanse dall'assemblatore in sequenze di istruzioni ammesse, usando il registro $1 ($at), riservato a questo scopo.

Una stessa pseudo istruzione può essere espansa in modi diversi, a seconda degli operandi contenuti.

Es.:

li $s0,0x1234	→	ori $s0,$0,0x1234
li $s0,-1	→	lui $1, -1 ori $s0,$1,-1
li $s0,0x12345678	→	lui $1,0x1234 ori $s0,$1,0x5678

Pseudoistruzioni aritmetiche

Pseudoistr.	*Significato*	*Esempio*	
abs *rd,rs*	*rd*= ABS(*rs*)	abs $1,$2	
div *rd,rs,src*	*rd*=*rs* ÷ *src*	div $1,$2,100	*eccezione possibile*
divu *rd,rs,src*	*rd*=*rs* ÷ *src*	divu $1,$2,$3	*nessuna eccezione*
mul *rd,rs,src*	*rd*=*rs* x *src*	mul $1,$2,100	*eccezione possibile*
mulo *rd,rs,src*	*rd*=*rs* x *src*	mulo $1,$2,$3	*eccezione possibile*
mulou *rd,rs,src*	*rd*=*rs* x *src*	mulou $1,$2,$3	*unsigned* *eccezione possibile*
rem *rd,rs,src*	*rd*=*rs* mod *src*	rem $1,$2,100	*signed*
remu *rd,rs,src*	*rd*=*rs* mod *src*	remu $1,$2,100	*unsigned*

Altre pseudoistruzioni

Confronto	*Controllo*	*Load/Store*	*Move*
seq *rd,rs1,rs2*	b *label*	la *rd,address*	move *rd,rs*
sge *rd,rs1,rs2*	beqz *rs,label*	li *rd,costant*	
sgeu *rd,rs1,rs2*	bge *rs1,rs2,label*	ld *rd,address*	
sgt *rd,rs1,rs2*	bgeu *rs1,rs2,label*	ulh *rd,address*	
sgtu *rd,rs1,rs2*	bgt *rs1,rs2,label*	ulhu *rd,address*	
sle *rd,rs1,rs2*	bgtu *rs1,rs2,label*	ulw *rd,address*	
sleu *rd,rs1,rs2*	ble *rs1,rs2,label*	sd *rs,address*	
	bleu *rs1,rs2,label*	ush *rs,address*	
	blt *rs1,rs2,label*	usw *rs,address*	
	bltu *rs1,rs2,label*		
	bnez *rs,label*		

Altri modi di indirizzamento

L'assembler MIPS permette anche altri modi di indirizzamento oltre quelli direttamente implementati in hardware (*pseudo-indirizzamento*).
Vengono realizzati con i modi disponibili in una o più istruzioni.

diretto

lw $t0,vett → lui $1,vett$_{HIGH}$
lw $8,vett$_{LOW}$($1)

registro indiretto

lw $t0,($t1) → lw $8,0($9)

base+registro indiretto

lw $t0, vett+4($t0) → lui $1, vett$_{HIGH}$
addu $1, $1, $8
lw $8, {4+vett$_{LOW}$ }($1)

Direttive

Forniscono all'assemblatore istruzioni relative all'assemblaggio del programma. La loro interpretazione non genera codice, ma provoca lo svolgimento di particolari azioni da parte dell'assemblatore.

Principali direttive

```
.align n                .half h1,…,hn
.ascii str              .kdata <addr>
.asciiz str             .ktext <addr>
.byte b1,...,bn         .set noat
.data <addr>            .set at
.double d1,…,dn         .space n
.extern sym             .text <addr>
.float f1,…,fn          .word w1,...,wn
.globl sym
```

.align n

Allinea il dato successivo a blocchi di 2^n byte.

Es.: .align 2 allinea alla word il dato successivo

.align 0 elimina l'allineamento automatico

.ascii str

.asciiz str

Mette in memoria la stringa str (non) terminata dal carettere null

.byte b1, …, bn

.half h1,…,hn

.word w1,…,wn

Memorizza n byte (halfword,word) in parole consecutive della memoria

.space n

Alloca n byte a partire dall'indirizzo corrente.

.data <addr>

.kdata <addr>

Gli elementi successivi sono memorizzati nel segmento dati utente (kernel)

.text <addr>

.ktext <addr>

Gli elementi successivi sono memorizzati nel segmento testo utente (kernel)

Definizione costanti

Costanti Numeriche

12 decimale
$2F esadecimale

Costanti Carattere

Delimitate da doppi apici. Generano la sequenza di byte corrispondenti ai codici ASCII dei relativi caratteri

mesg: .asciiz "ciao"
mesg: .byte $43,$49,$41,$4F,0

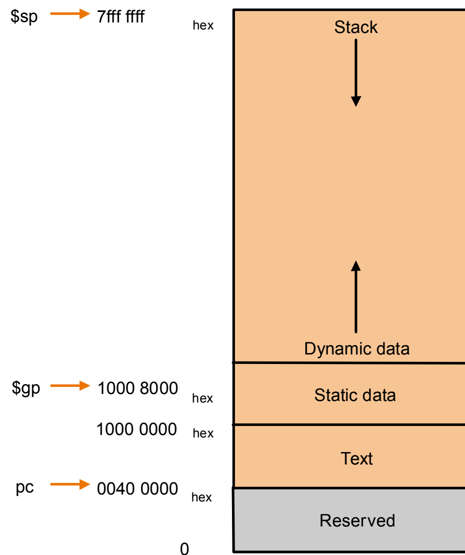
Chiamate di sistema (syscall)

Service	System call code	Arguments	Result
print_int	1	$a0 = integer	
print_float	2	$f12 = float	
print_double	3	$f12 = double	
print_string	4	$a0 = string	
read_int	5		integer (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	address (in $v0)
exit	10		

Chiamate di sistema (syscall)

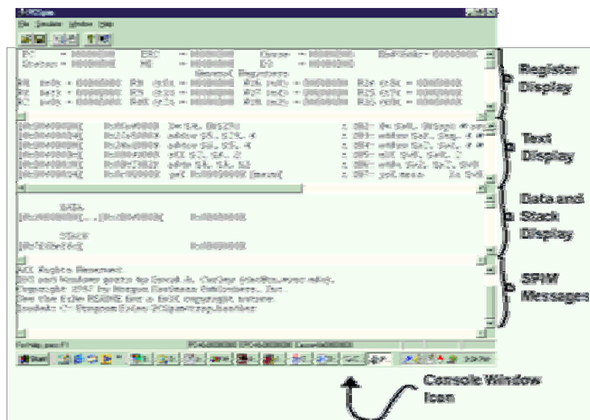
```
      .data
str:
      .asciiz "the answer = "
      .text
      li $v0, 4          # system call code for print_str
      la $a0, str        # address of string to print
      syscall            # print the string
      li $v0, 1          # system call code for print_int
      li $a0, 5          # integer to print
      syscall            # print it
```


Allocazione di memoria del MIPS



Il simulatore SPIM – 1

- Interfaccia grafica suddivisa in quattro pannelli



Il simulatore SPIM – 2

- Messaggi dalla console
- Il file sorgente assembly
- Creare e modificare un file assembly con il Notepad

```
# Inizio del codice

        .text
        .globl main

main:

# Carica vet, isize e l'indice
        la $16, vet
        lw $17, isize
        ori $18, $0, 0

# Ciclo for

for1:
# carica l'elemento del vettore e lo stampa
        lw $a0, 0($16)
        li $v0, 1
        syscall
```

Ciclo di produzione

- Editing del sorgente assembly
- Caricamento
- Esecuzione
  - Esecuzione continua
  - Esecuzione passo passo

Esempio

```
#
# Esercitazioni simulatore SPIM
# Esempio  1
#
#
#    main()
#    {
#         int a,b,c;
#
#         a=12;     a-> $16
#         b=7;      b-> $17
#         c=a+b;    c-> $18
#         printf("%d", c);
#    }
#

    .text
    .globl main
```

```
main:

# Carica i registri
          addi $16, $0, 12
          addi $17, $0, 7

# Esegue la somma
          add $18, $16, $17

# Visualizza il risultato
          add $a0, $0, $18
          li $v0, 1
          syscall

# Termina l'esecuzione
          li $v0, 10
          syscall
```