Architetture dei Calcolatori e Sistemi Operativi
1° Esercitazione, 29-30/09/2020

(Rielaborati da esercitazioni prof. Breveglieri/PhD Zoni)

ESERCIZI BASE:

1) IF-THEN-ELSE [cond1 && cond2]

```
int a = 1;
int b = 2;
int c = 3;

int main ()
{
     if( a==b && a == c)
          a++;
     else
          b++;
     c = 12;
}
```

```
.data
A:   .word 1
B:   .word 2
C:   .word 3

.text
.globl MAIN
MAIN:
     #if( a==b && a==c)
     lw $t0, A($gp)
     lw $t1, B($gp)
     bne $t0, $t1, ELSE  (*)
     lw $t1, C($gp)
     bne $t0, $t1, ELSE
THEN:
     #a++
     lw $t0, A($gp)
     addiu $t0, $t0, 1
     sw $t0, A($gp)
     j END_IF
ELSE:
     #b++
     lw $t0, B($gp)
     addiu $t0,$t0, 1
     sw $t0, B($gp)
END_IF:
     li $t0, 12
     sw $t0, C($gp)

     #syscall to exit (10)
     li $v0, 10
     syscall
```

- **Come cambia il codice assembly se sostituiamo "&&" con "||" ?**
  (*) bne $t0, $t1, ELSE → beq $t0, $t1, THEN
- **Come cambia il codice assembly se sostituiamo a==b con a>=b?**
  blt $t0, $t1, ELSE (Pseudoistruzione)
  → slt $t0, $t0, $t1
  → bne $t0, $zero, ELSE

- **Come cambia il codice assembly se sostituiamo a==b con a>b?**

```
ble $t0, $t1, ELSE
    slt $t0, $t1, $t0
    beq $t0, $zero, ELSE
```

- **NOTA**: in questo esercizio abbiamo esplicitamente scritto, nelle istruzioni di load e store, il base address rispetto a cui si calcola lo spiazzamento. (es. lw $to, A($gp)). Il base address ($gp) può essere omesso. Da qui in avanti sarà omesso. Ricordatevi che l'indirizzo a cui si accede per le istruzioni di lw/sw è sempre $gp + offset.

2) FOR-LOOP E ARRAY (accesso tramite indice)

```
//trova il valore massimo dell'array

int vett[10]={1,2,3,4,5,6,7,8,9,10};
int max;
int i;

int main()
{
     max = vett[0];
     for(i=0;i<10;i++)
     {
          if(vett[i] > max)
               max = vett[i];
     }
}
```

```
.data
VETT: .word 1,2,3,4,5,6,7,8,9,10
MAX:  .space 4
I:    .space 4

.text
.globl MAIN
MAIN:
     lw    $t0, VETT
     sw    $t0, MAX
     #i=0
     sw    $zero, I
LOOP:
     #i<10
     lw    $t0, I
     li    $t1, 10
     bge   $t0, $t1, END_LOOP
     #if(vett[i] > max)
     la    $t1, VETT
     lw    $t0, I
     sll   $t0, $t0, 2
     addu  $t0, $t1, $t0
     lw    $t0, 0($t0)
     lw    $t1, MAX
     ble   $t0, $t1, END_IF
     sw    $t0, MAX
END_IF:
     #i++
     lw    $t0, I
     addiu $t0,$t0, 1
     sw    $t0, I
     j LOOP
END_LOOP:
     # syscall to exit (10)
     li    $v0, 10
     syscall
```

NOTE:

- **<u>Tutte var. globali</u>: vanno salvate ogni volta che vengono aggiornate**
- **`la $t0, VETT` sposta in $t0 l'indirizzo di VETT → Pseudoistruzione**
  - **`lui $t0, %hi(VETT)`**
  - **`ori $t0, %lo(VETT)`**
    %hi(VETT) e %lo(VETT) indicano rispettivamente i 16 bit più significativi e i 16 bit meno significativi di VETT
- **lw $t0, VETT sposta in $t0 il valore all'indirizzo VETT**
  - **`lw $t0, %gp_rel(VETT)($gp)`**
    %gp_rel(VETT) indica lo spiazzamento di ciascun elemento di VETT rispetto al global pointer

3) FOR-LOOP E ARRAY (accesso tramite puntatore)

```
//trova il valore massimo dell'array

int vett[10]={1,2,3,4,5,6,7,8,9,10};
int max;
int i;
int* p;

void main()
{
     max = vett[0];
     p = vett; //p punta a vett
     for(i=0;i<10;i++)
     {
          if(*p > max)
               max = *p;
          p++;
     }
}
```

```
.data
VETT: .word 1,2,3,4,5,6,7,8,9,10
MAX:  .space 4
I:    .space 4
P:    .space 4

.text
.globl MAIN
MAIN:
     lw    $t0, VETT
     sw    $t0, MAX
     la    $t0, VETT
     sw    $t0, P
     #i=0
     sw    $zero, I
LOOP:
     #i<10
     lw    $t0, I
     li    $t1, 10
     bge   $t0, $t1, END_LOOP
     #if(*p > max)
     lw    $t0, P
     lw    $t0, 0($t0)
     lw    $t1, MAX
     bge   $t1, $t0, END_IF
     sw    $t0, MAX
END_IF:
     #p++
     lw $t0, P
     addiu $t0, $t0, 4
     sw $t0, P
     #i++
     lw    $t0, I
     addiu $t0,$t0, 1
     sw    $t0, I

     j LOOP
END_LOOP:
     # syscall to exit (10)
     li    $v0, 10
     syscall
```

3) Valutazione delle espressioni

```
//calcola il risultato
dell'espressione

int a = 5;
int b = 7;
int c = 12;

int res1;
int res2;
int res3;

void main()
{
     res1 = a+b-c-a;
     res2 = a+(b-(c-a));
     res3 = a+b*c-a;
}
```

```
.data
A:    .word 5
B:    .word 7
C:    .word 12
RES1: .space 4
RES2: .space 4
RES3: .space 4

.text
.globl MAIN
MAIN:
     #(a+b)-c-a;
     lw $t0, A
     lw $t1, B
     add $t0, $t0, $t1
     lw $t1, C
     sub $t0,$t0, $t1
     lw $t1, A
     sub $t0, $t0, $t1
     sw $t0, RES1

     #a+(b-(c-a));
     lw $t0, A
     lw $t1, B
     lw $t2, C
     lw $t3, A
     sub  $t2,  $t2,  $t3
     sub  $t1,  $t1,  $t2
     add  $t0,  $t0,  $t1
     sw $t0, RES2

     #a+b*c-a;
     lw $t0, A
     lw $t1, B
     lw $t2, C
     mul $t1, $t1, $t2
     add $t0,$t0, $t1
     lw $t1, A
     sub $t0,$t0, $t1
     sw $t0, RES3


     #exit
     lw $v0, 10
     syscall
```

esercizio n. 1 – linguaggio macchina

traduzione da C ad assembler

Si deve tradurre in linguaggio macchina simbolico (assemblatore) *MIPS* il frammento di programma C riportato sotto. Il modello di memoria è quello **standard *MIPS*** e le variabili intere sono da **32 bit**. Non si tenti di accorpare od ottimizzare insieme istruzioni C indipendenti. Si facciano le ipotesi seguenti:

- il registro "frame pointer" *fp* non è in uso
- le variabili locali sono allocate nei registri, se possibile
- vanno salvati (a cura del chiamante o del chiamato, secondo il caso) solo i registri necessari

Si chiede di svolgere i quattro punti seguenti (usando le varie tabelle predisposte nel seguito):

1. **Si traducano** in linguaggio macchina le dichiarazioni delle variabili globali e **si indichi** l'indirizzo iniziale di ciascuna variabile globale dichiarata.
2. **Si traduca** in linguaggio macchina lo statement riquadrato del programma principale ***main***.
3. **Si descrivano** l'area di attivazione della funzione ***ptrloop*** e l'allocazione delle variabili locali nei registri.
4. **Si traduca** in linguaggio macchina il codice della funzione ***ptrloop***.

```
/* sezione dichiarativa costanti e variabili globali        */
#define  N 3
int val = 5
int * ptrlist [N]
int * ptr

int stampa_dato (int dato)     /* testata funz. stampa_dato */

int * ptrloop (int lim, int elm) {      /* funzione ptrloop */
   int i
   int * p
   p = NULL
   for (i = 0; i != lim; i++) {
      p = ptrlist [i]
      if (*p == elm) break  /* end if */
      stampa_dato (elm)
   } /* end for */
   return p
} /* end ptrloop */

void main ( ) {                    /* programma principale */
   ptr = ptrloop (3, val)
} /* end main */
```

punto 1 – codice MIPS della sezione dichiarativa globale (numero di righe non significativo)				**indirizzo iniz. della variabile**
	.eqv	N, 3		
	.data		// seg. dati statici – 0x 1000 0000	
VAL:	***.word***	*5*	*// spazio varglob VAL (iniz)*	*0x 1000 0000*
PTRLIST:	***.space***	*12*	*// spazio varglob PTRLIST (non iniz)*	*0x 1000 0004*
PTR:	***.space***	*4*	*// spazio varglob PTR (non iniz)*	*0x 1000 0010*

punto 2 – codice MIPS dello statement da tradurre di ***MAIN*** – (numero di righe non significativo)			
	li	*$a0, 3*	*// prepara param LIM **
	lw	*$a1, VAL*	*// prepara param VAL*
	jal	*PTRLOOP*	*// chiama funz PTRLOOP*
	sw	*$v0, PTR*	*// aggiorna varglob PTR*
** altrimenti anche* ***addi*** *$a0, $zer0, 3*			

punto 3 – area e registri di ***PTRLOOP*** (numero di righe non significativo)

area di attivazione di *PTRLOOP*

	spiazz. rispetto a stack pointer	**contenuto simbolico**
indirizzi alti	*8*	*ra (reg ra salvato da PTRLOOP non-foglia)*
	4	*s0 (reg di MAIN salvato da PTRLOOP)*
(fine area) sp →	*0*	*s1 (reg di MAIN salvato da PTRLOOP)*
indirizzi bassi	*–4*	*valore di a0 salvato*

allocazione parametri e variabili locali di *PTRLOOP* nei registri

parametro / variabile locale	**registro**
LIM (tipo intero)	*a0*
ELM (tipo intero)	*a1*
I (tipo intero)	*s0*
P (tipo puntatore)	*s1*

punto 4 – codice MIPS dell'intera funzione ***PTRLOOP*** **-** (numero di righe non significativo)			
PTRLOOP:	***addiu***	*$sp, $sp, -12*	*// crea area di attivazione*
	.equ	*RE, 8*	*// spi di reg ra*
	.equ	*S0, 4*	*// spi di reg s0*
	.equ	*S1, 0*	*// spi di reg s1*
	sw	*$ra, RA($sp)*	*// salva reg ra*
	sw	*$s0, S0($sp)*	*// salva reg s0*
	sw	*$s1, S1($sp)*	*// salva reg s1*
	// p = NULL		
	move	*$s1, $zero*	*// inizializza varloc p*
	// for (i = 0; i != lim; i++) prologo e test condizione		
	move	*$s0, $zero*	*// inizializza varloc i*
FOR:	***beq***	*$s0, $a0, ENDFOR*	*// se i == lim vai a ENDFOR*
	// p = ptrlist [i]		
	la	*$t0, PTRLIST*	*// carica ind di elem ptrlist [0]*
	sll	*$t1, $s0, 2*	*// allinea indice i*
	addu	*$t0, $t0, $t1*	*// calcola ind di elem ptrlist [i]*
	lw	*$s1, ($t0)*	*// aggiorna p con elm ptrlist [i]*
	// if (*p == elm) break		
	lw	*$t2, ($s1)*	*// carica oggetto puntato da p*
	beq	*$t2, $a1, ENDFOR*	*// se oggetto == elm vai a ENDFOR*
	// stampa_dato (elm)		
	addiu	*$sp, $sp, -4*	*// aggiorna sp*
	sw	*$a0, ($sp)*	*// push di arg lim di funz ptrloop*
	move	*$a0, $a1*	*// prepara param di funz stampa*
	jal	*STAMPA_DATO*	*// chiama funz stampa*
	lw	*$a0, ($sp)*	*// pop di arg lim di funz ptrloop*
	addiu	*$sp, $sp, 4*	*// aggiorna sp*
	// epilogo del ciclo		
	addi	*$s0, $s0, 1*	*// incrementa indice ciclo for*
	j	*FOR*	*// salta a condizione ciclo for*
	// return p SCRIVERE SOLO L'ISTRUZIONE/I CHE PREPARA IL VALORE USCITA		
ENDFOR:	***move***	*$v0, $s1*	*// prepara valusc*
Nota: si suppone che la funz stampa_dato non alteri il registro a1.			