COMPITINO #1 di CALCOLATORI ELETTRONICI del 08-11-01 **MATRICOLA**_____

COGNOME_____

NOME_____

[Versione adattata nell'esercitazione del 16/02/2005]

Si consideri il seguente kernel-benchmark LLL3 (Lawrence Livermoore Loop #3).

```
static int X[2000], Z[2000];
static int Loop3(int n, int *X, int *Y) {
   int i, Q;
   Q = 0;
   for (i = 0, i < n; ++i) {
      Q = X[i] * Y[i];
   }
   return Q;
}
main()
{
   int n, k, i, V;
   n = 1000; V = 0;
   for (k = 0; k < n; ++k) {
      X[k] = 222; Z[k] = 222;
   }
   for (i = 0; i < n; ++i) {
      V = Loop3(n, X, Z);
   }
}
```

1) Scrivere il corrispondente codice assembly MIPS utilizzando solo e unicamente istruzioni dalla tabella riportata qua sotto, piu' la pseudoistruzione 'la'.
2) Calcolare il tempo di esecuzione del codice tradotto al punto precedente su un processore con frequenza di clock pari a 2 GHz, assumendo i seguenti valori per il CPI di ciascuna categoria di istruzioni: aritmetico-logiche 1, branch 3, load-store 2.

MIPS instructions

Instruction	Example		Meaning	Comments
add	`add`	`$1,$2,$3`	$1 = $2 + $3	3 operands; exception possible
subtract	`sub`	`$1,$2,$3`	$1 = $2 - $3	3 operands; exception possible
add immediate	`addi`	`$1,$2,100`	$1 = $2 + 100	+ constant; exception possible
subtract immediate	`subi`	`$1,$2,100`	$1 = $2 - 100	- constant; exception possible
multiplication	`mult`	`$1, $2`	Hi,Lo= $1 x $2	64-bit Signed Product ; result in Hi,Lo
division	`div`	`$1, $2`	Hi= $1 % $2, Lo = $1 / $2	Signed division
move from Hi	`mfhi`	`$1`	$1 = Hi	Create copy of Hi
move from Lo	`mflo`	`$1`	$1 = Lo	Create copy of Lo
and	`and`	`$1,$2,$3`	$1 = $2 & $3	3 register operands; Logical AND
or	`or`	`$1,$2,$3`	$1 = $2 | $3	3 register operands; Logical OR
nor	`nor`	`$1,$2,$3`	$1 = !($2 | $3)	3 register operands; Logical NOR
xor	`xor`	`$1,$2,$3`	$1 = $2 ^ $3	3 register operands; Logical XOR
and immediate	`andi`	`$1,$2,100`	$1 = $2 & 100	Logical AND register, constant
or immediate	`ori`	`$1,$2,100`	$1 = $2 | 100	Logical OR register, constant
xor immediate	`xori`	`$1,$2,100`	$1 = $2 ^ 100	Logical XOR register, constant
shift left logical	`sll`	`$1,$2,10`	$1 = $2 << 10	Shift left by constant
shift right logical	`srl`	`$1,$2,10`	$1 = $2 >> 10	Shift right by constant
load word	`lw`	`$1,100($2)`	$1 = Memory[$2+100]	Data from memory to register
load byte	`lb`	`$1,100($2)`	$1 = Memory[$2+100]	Data from memory to register
load byte unsigned	`lbu`	`$1,100($2)`	$1 = Memory[$2+100]	Data from mem. to reg.; no sign extension
store word	`sw`	`$1,100($2)`	Memory[$2+100] = $1	Data from register to memory
store byte	`sb`	`$1,100($2)`	Memory[$2+100] = $1	Data from register to memory
load upper immediate	`lui`	`$1,100`	$1 = 100 * 256	Load constant in upper 16bits
branch on equal	`beq`	`$1,$2,100`	if ($1 == $2) go to PC+4+100	Equal test; PC relative branch
branch on not equal	`bne`	`$1,$2,100`	if ($1 != $2) go to PC+4+100	Not equal test; PC relative
set on less than	`slt`	`$1,$2,$3`	if ($2 < $3) $1 = 1; else $1 = 0	Compare less than; 2's complement
set on less than immediate	`Slti`	`$1,$2,100`	if ($2 < 100) $1 = 1; else $1 = 0	Compare < constant; 2's complement
set on less than unsigned	`Sltu`	`$1,$2,$3`	if ($2 < $3) $1 = 1; else $1 = 0	Compare less than; natural number
set on less than imm. unsigned	`sltiu`	`$1,$2,100`	if ($2 < 100) $1 = 1; else $1 = 0	Compare constant; natural number
jump	`j`	`10000`	go to 10000	Jump to target address
jump register	`jr`	`$31`	go to $31	For switch, procedure return
jump and link	`jal`	`10000`	$31 = PC + 4;go to 10000	For procedure call

Register Usage

Name	Register Number	Usage
$zero	0	the constant value 0
$s0-$s7	16-23	Saved
$t0-$t9	8-15,24-25	Temporaries
$a0-$a3	4-7	Arguments
$v0-$v1	2-3	Results
$fp, $sp, $ra, $gp	30,29,31,28	Frame pointer, stack pointer, return address, global pointer

```
.data
X:      .space  8000    # 2000 interi
Z:      .space  8000    # 2000 interi

.text
.globl main

Loop3:
        # $a0=n, $a1=X, $a2=Y
        # $t0=i, $v0=Q
        # $t1=tmp, $t2=tmp
        add     $v0, $0, $0         # Q=0
(L0)    add     $t0, $0, $0         # i=0
l1:
        slt     $t2, $t0, $a0       # $t2=(i<n)
(L1)    beq     $t2, $0, endl1      # Condizione falsa ➔ termine ciclo
        sll     $t1, $t0, 2         # $t1=i<<2 (=i*4)
        add     $t2, $a1, $t1       # $t2=&X[i]
        lw      $t2, 0($t2)         # $t2=X[i]
        add     $t3, $a2, $t1       # $t3=&Y[i]
        lw      $t3, 0($t3)         # $t3=Y[i]
        mult    $t2, $t3            # (hi,lo)=X[i]*Y[i] (su 64 bit)
        mflo    $v0                 # Q=X[i]*Y[i] (su 32 bit)
        addi    $t0, $t0, 1         # i++
(L2)    j       l1                  # Ripeti il ciclo
endl1:
(L3)    jr      $ra                 # Esci da subroutine

main:
        # $s0=n, $s1=k, $s2=i, $s3=V, $s4=X, $s5=Z
        # $t0=tmp, $t2=222
        la      $s4, X              # $s4=&X[0]
        la      $s5, Z              # $s5=&Z[0]
        ori     $s0, $0, 1000       # n=1000
        add     $s3, $0, $0         # V=0
        ori     $t2, $0, 222        # $t2=222
(M0)    add     $s1, $0, $0         # k=0
l2:
        slt     $t0, $s1, $s0       # $t0=(k<n)
(M1)    beq     $t0, $0, endl2      # Condizione falsa ➔ termine ciclo
        sll     $t0, $s1, 2         # $t0=k<<2 (=k*4)
        add     $t1, $s4, $t0       # $t1=&X[k]
        sw      $t2, 0($t1)         # X[k]=222
        add     $t1, $s5, $t0       # $t1=&Z[k]
        sw      $t2, 0($t1)         # Z[k]=222
        addi    $s1, $s1, 1         # k++
(M2)    j       l2                  # Ripeti il ciclo
endl2:
(M3)    add     $s2, $0, $0         # i=0
l3:
        slt     $t0, $s2, $s0       # $t0=i<n
(M4)    beq     $t0, $0, endl3      # Condizione falsa ➔ termine ciclo
        add     $a0, $s0, $0        # I argomento per Loop3
        add     $a1, $s4, $0        # II argomento per Loop3
        add     $a2, $s5, $0        # III argomento per Loop3
        jal     Loop3               # Loop3(n,X,Z)
        add     $s3, $v0, 0         # V=Loop3(n,X,Z)
        addi    $s2, $s2, 1         # i++
(M5)    j       l3                  # Ripeti il ciclo
endl3:
        ori     $v0, $0, 10         # $v0=codice per la exit
        syscall                     # exit()
```

Il partizionamento del programma in basic block e' fatta prendendo sequenze di una o piu' istruzioni consecutive che o terminano con una istruzione di branch condizionale o una jump incondizionata (ma non una jal, si veda M5), oppure

terminano subito prima di un'etichetta di salto. Si noti che l'invocazione della syscall exit() in fondo al main non era strettamente richiesta dalla traccia, e non viene considerata nella soluzione ai fini del calcolo del tempo di esecuzione.

	Note	Ni	AL(1)	BJ(3)	LS(2)	cBi	CBi=cBi*Ni
M0	Inizio main()	1	8	0	0	8	8
M1	Verifica condizione uscita del I for	1001	1	1	0	4	4.004
M2	Corpo del I for	1000	4	1	2	11	11.000
M3	Inizializzazione II for	1	1	0	0	1	1
M4	Verifica condizione uscita del II for	1001	1	1	0	4	4.004
M5	Corpo II for (e invocazioni Loop3)	1000	5	2	0	11	11.000
L0	Inizio Loop3()	1000x1	2	0	0	2	2.000
L1	Verifica condizione uscita del for	1000x1001	1	1	0	4	4.004.000
L2	Corpo del for	1000x1000	6	1	2	13	13.000.000
L3	Uscita da Loop3	1000x1	0	1	0	3	3.000
Ccpu	**(durata in clock)**						**17.039.017**
Tcpu	**Ccpu/fc**						**8,520 ms**

Si noti che la pseudoistruzione 'la' nel blocco M0 equivale a 2 istruzioni aritmetico-logiche (lui+ori).
In definitiva, il tempo di esecuzione del programma risulta di circa 8 ms e 520 us.