COMPITINO #1 di CALCOLATORI ELETTRONICI 1 del 05-11-03 **MATRICOLA**_____

(versione riveduta) **COGNOME**_____

NOME_____

1) Nel seguente programma, dopo aver trovato il corrispondente codice assembly MIPS (utilizzando solo e unicamente istruzioni dalla tabella riportata qua sotto), calcolare il tempo di esecuzione di tale codice 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 5.

```
char find_op(char *s)
{
   char op = '\0';
   while (*s != '\0') { if (*s < '0' || *s > '9') { op = *s; break; } else { ++s; } }
   return (op);
}

char buff[80] = "2*3+4/5";

main()
{
   int a, b, d; double f, g, h; float w, x, y; char c, *p = buff;

   do {
      switch (c = find_op(p++)) {
         case '+': a = b + d; break;
         case '*': f = g / x; break;
         case '/': w = x * y; break;
      }
   } while (c != '\0')
   h = a * w;
}
```

2) Riportare la Tabella di Rilocazione per il precedente programma.

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 address	`la $1,var`	$1 = &var	Load variable address
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
add.s add.d	`add.x $f0,$f2,$f4`	$f0=$f2+$f4	Single and double precision add
sub.s sub.d	`add.x $f0,$f2,$f4`	$f0=$f2-$f4	Single and double precision subtraction
mul.s mul.d	`mul.x $f0,$f2,$f4`	$f0=$f2*$f4	Single and double precision multiplication
div.s div.d	`div.x $f0,$f2,$f4`	$f0=$f2/$f4	Single and double precision division
mov.s mov.d	`mov.x $f0,$f2`	$f0←$f2	Single and double precision move
abs.s abs.d	`abs.x $f0,$f2`	$f0=ABS($f2)	Single and double precision absolute value
c.lt.s c.lt.d (eq,ne,le,gt,ge)	`c.lt.x $f0,$f2`	Temp=($f0<$f2)	Single and double: compare $f0 and $f2 <,=,!=,<=,>,>=
mtc1	`mtc1 $1,$f2`	$f2=$1	Data from gen. register to C1 register (no conversion)
branch on false	`bc1f label`	If (Temp = = false) go to label	Temp is 'Condition-Code'
branch on true	`bc1t label`	If (Temp = = true) go to label	Temp is 'Condition-Code'
load floating point (32bit)	`lwc1 $f0,0($1)`	$f0←Memory[$1]	
store floating point (32bit)	`swc1 $f0,0($1)`	Memory[$1]←$f0	
convert single into double	`cvt.d.s $f0,$f2`	$f0=(double)$f2	Also cvt.s.d (viceversa)
convert single into integer	`cvt.w.s $f1,$f0`	$f1=(int)$f0	Also cvt.s.w (viceversa)

Register Usage

Name	Register Num.	Usage
$zero	0	The constant value 0
$s0-$s7	16-23	Saved
$t0-$t9	8-15,24-25	Temporaries
$a0-$a3	4-7	Arguments

Name	Register Num.	Usage
$v0-$v1	2-3	Results
$fp, $sp	30,29	Frame pointer, stack pointer
$ra, $gp	31,28	return address, global pointer
$k0-$k1	26,27	Kernel usage

Name	Usage
$f0, $f1, …, $f31	Single precision floating point registers
$f0, $f2, …, $f30	Double precision floating point registers

COMPITINO #1 di CALCOLATORI ELETTRONICI 1 del 05-11-03 **SOLUZIONE**

1) Una possibile soluzione e' riportata nel file parse1.s. Sono riportate in verde le parti necessarie allo svolgimento corretto dell'esercizio.

Tale programma puo' essere provato sul simulatore SPIM: i) aggiungendo delle istruzioni che riempiano fittiziamente i registri s1, s2, s3, f1, f2, f3, f10, f12, f14.

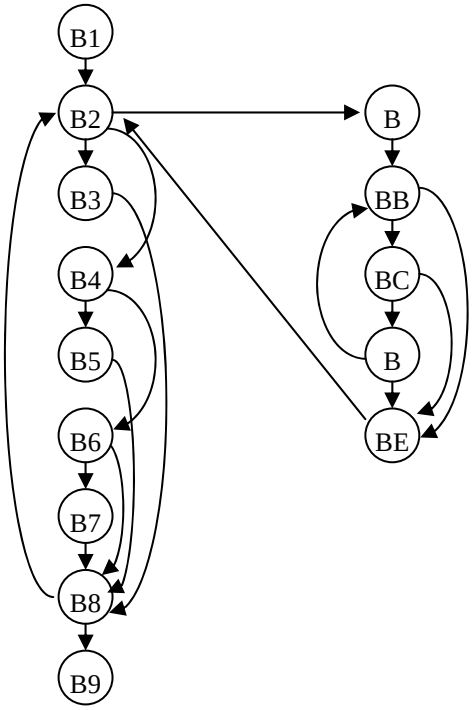
```
.data
buff:    .asciiz "2*3+4/5"
         .space  72

#parametri ingresso: char *s in $a0

.text
.globl main

find_op:
         add     $v0, $0, $0      # op = '\0'
         add     $t9, $a0, $0     # t9 = s (param. ingresso)
while:   lb      $t0, 0($t9)      # carica il carattere *s
         beq     $t0, $0, end2    # se *s==0 termina il ciclo
         slti    $t1, $t0, '0'    # t1 = (t0 <? '0')
         addi    $t2, $0, '9'     #
         slt     $t3, $t2, $t0    # t3 = (t0 >? '9')
         or      $t4, $t1, $t3    # t4 = t1 || t3
         bne     $t4, $0, end1    # se la cond. e' vera ho finito
         addi    $t9, $t9, 1      # ...altrimenti s++
         j       while            # e continuo il ciclo while
end1:    add     $v0, $t0, $0     # scrivo in op il risultato
end2:    jr      $ra              # esco restituendo $v0

# $s3 = a, $s1 = b, $s2 = d
# $f14 = f, $f10 = g, $f12 = h
# $f3 = w, $f1 = x, $f2 = y;
# $v0 = c, $a0=p

main:
         addi    $s1, $0, 1       # s1  = b = 1
         addi    $s2, $0, 2       # s2  = d = 2
         addi    $t0, $0, 3
         mtc1    $t0, $f10
         cvt.s.w $f10,$f10
         cvt.d.s $f10,$f10        # f10 = g = 3
         addi    $t0, $0,4
         mtc1    $t0, $f12
         cvt.s.w $f12,$f12
         cvt.d.s $f12,$f12        # f12 = h = 4
         addi    $t0, $0, 5
         mtc1    $t0, $f1
         cvt.s.w $f1, $f1 # f1  = x = 5
         addi    $t0, $0, 6
         mtc1    $t0, $f2
         cvt.s.w $f2, $f2 # f2  = y = 6

         la      $a0, buff        # p = buff

dowhile: jal     find_op          # chiamo la find_op
         addi    $a0, $a0, 1      # p++

         addi    $t0, $0, '+'     # case '+' ?
         bne     $t0, $v0, sw2    # se no --> caso successivo
         add     $s3, $s1, $s2    # se si', esegui a=b+d
         j       fine_sw          # break

sw2:     addi    $t0, $0, '*'     # case '*' ?
         bne     $t0, $v0, sw3    # se no --> caso successivo
         cvt.d.s $f16, $f1        # se si', esegui f=g/x
         div.d   $f14, $f10, $f16
         j       fine_sw          # break

sw3:     addi    $t0, $0, '/'     # case '/' ?
         bne     $t0, $v0, fine_sw
         mul.s   $f3, $f1, $f2    # se si', esegui w=x*y
                                  # break
fine_sw:
         bne     $v0, $0, dowhile # c !=? '\0', se si': dowhile

         mtc1    $s3, $f7
         cvt.s.w $f4, $f7
         mul.s   $f5, $f4, $f3    # h = a * w
         cvt.d.s $f12, $f5

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

	'2'	'*'	'3'	'+'	'4'	'/'	'5'	'\0'
BA	1	1	1	1	1	1	1	
BB	1 1	1	1 1	1	1 1	1	1 1	
BC	1 1	1	1 1	1	1 1	1	1	
BD	1		1		1		1	
BE	1	1	1	1	1	1		
BF	1	1	1	1	1	1	1	

	'2'	'*'	'3'	'+'	'4'	'/'	'5'	'\0'
B1	1							
B2	1	1	1	1	1	1	1	
B3			1	1				
B4	1	1			1	1	1	
B5	1	1						
B6					1	1	1	
B7					1	1		
B8	1	1	1	1	1	1	1	
B9							1	

Per calcolare il tempo di esecuzione di poteva procedere come segue:

i) suddivisione del programma in basic-block come riportato nella pagina precedente. Puo' essere utile fare uno schema dell'ordine in cui i basic-block vengono eseguiti:



ii) conto le volte in cui ogni blocco e' eseguito nel programma assembly;
iii) in tabella, riporto per ogni basic-block Bi, il numero Ni di volte in cui il blocco e' eseguito, la sua composizione in termini di istruzioni Aritmetico-Logico (AL), Branch-Jump (BJ) e Load-Store (LS), il peso in termini di cicli cBi e il contributo complessivo CBi di quel blocco sui cicli totali del programma:

Bi	Ni	AL	BJ	LS	cBi	CBi=cBi*Ni
B1	1	1	0	0	1	1
B2	7	2	2	0	7	56
B3	2	1	1	0	4	8
B4	5	1	1	0	4	20
B5	2	2	1	0	5	10
B6	3	1	1	0	4	12
B7	2	1	0	0	1	2
B8	7	0	1	0	3	21
B9	1	4	0	0	4	4
BA	7	2	0	0	2	14
BB	11	0	1	1	8	88
BC	10	4	1	0	7	70
BD	4	1	1	0	4	16
BE	6	1	1	0	1	6
BF	7	0	1	0	3	21

Chiaramente:

$T_{CPU}=C_{CPU}/fc$ $C_{CPU}=\sum_i C_{Bi}=349$ $fc=2*10^{-9}$ ===> $T_{CPU}=349/2*10^{-9}=172.5$ ns

2) Per il secondo esercizio, nell'ipotesi di trascurare le istruzioni "di riempimento iniziale" dei registri, la tabella di rilocazione per il codice proposto risulta:

I	R	
56	0	buff
44	12	while
60	0	find_op
80	116	fine_sw
100	116	fine_sw

Si ricorda che I indica il valore dell'offset dell'istruzione contenente il simbolo da rilocare rispetto all'inizio dell'area codice. R indica il valore dell'offset del simbolo da rilocare rispetto all'inizio dell'area (dati o codice) ove si trova.