

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 Rilocalazione 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	bclf label	If (Temp == false) go to label	Temp is 'Condition-Code'
branch on true	bclt 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