**COMPITO di ARCHITETTURA DEI CALCOLATORI del 21-07-2017**

**MATRICOLA**_____

**COGNOME**_____

**NOME**_____

**DA RESTITUIRE INSIEME AGLI ELABORATI e A TUTTI I FOGLI**
**→ NON USARE FOGLI NON TIMBRATI**
**→ ANDARE IN BAGNO PRIMA DELL'INIZIO DELLA PROVA**
**→ NO APPUNTI O FOGLI PERSONALI, NO TELEFONI, SMARTPHONE, ETC**

**SVOLGIMENTO DELLA PROVA (selezionare una delle seguenti 4 opzioni):**
☐ PER GLI STUDENTI DI "ARCHITETTURA DEI CALCOLATORI – A.A. 2015/16 e 16/17": es. N.1+2+3+7.
☐ PER GLI STUDENTI DEGLI ANNI PRECEDENTI che devono svolgere sia il modulo CALCOLATORI che il modulo RETI: es. N.1+2+3+4+6.
☐ PER GLI STUDENTI DEGLI ANNI PRECEDENTI che devono svolgere SOLO il modulo CALCOLATORI es. N.1+2+3+4+5.
☐ PER GLI STUDENTI DEGLI ANNI PRECEDENTI che devono svolgere SOLO il modulo RETI: es. N.6+7.

NOTA: per l'esercizio 7 dovranno essere consegnati due files: il file del programma VERILOG e il file relativo all'output (screenshot o copy/paste)

1) [18] Trovare il codice assembly MIPS corrispondente del seguente programma (**utilizzando solo e unicamente istruzioni dalla tabella sottostante e rispettando le convenzioni di utilizzazione dei registri dell'assembly MIPS** riportate qua sotto, per riferimento).

```
int i = 0, j = 0, k = 0;
void multiply (int m1, int n1, int a[2][2], int m2, int n2,
int b[2][2], int c[2][2]) {
    if (i >= m1) {
        return;
    } else if (i < m1) {
        if (j < n2) {
            if (k < n1) {
                c[i][j] += a[i][k] * b[k][j];
                k++;
                multiply(m1, n1, a, m2, n2, b, c);
            }
            k = 0; j++;
            multiply(m1, n1, a, m2, n2, b, c);
        }
        j = 0; i++;
        multiply(m1, n1, a, m2, n2, b, c);
    }
}
```

```
void display(int m1, int n2, int c[2][2]) {
    int i, j;
    for (i = 0; i < m1; i++) {
        for (j = 0; j < n2; j++) {
            print_int(c[i][j]);
            print_string("  ");
        }
        print_string("\n");
    }
}
int main() {
    int x[2][2] = { 12, 56, 45, 78 };
    int y[2][2] = { 2, 6, 5, 8 };
    int z[2][2] = {0, 0, 0, 0 };
    multiply(2, 2, x, 2, 2, y, z);
    display(2, 2, z);
}
```

2) [8] Si consideri una cache di dimensione 128B e a 4 vie di tipo write-back/write-non-allocate. La dimensione del blocco e' 8 byte, il tempo di accesso alla cache e' 4 ns e la penalita' in caso di miss e' pari a 40 ns, la politica di rimpiazzamento e' FIFO. Il processore effettua i seguenti accessi in cache, ad indirizzi al byte: 55, 173, 115, 119, 222, 947, 618, 449, 534, 748, 877, 919, 283, 143, 591, 644, 770, 845, 961, 194. Tali accessi sono alternativamente letture e scritture. Per la sequenza data, ricavare il tempo medio di accesso alla cache, riportare i tag contenuti in cache al termine, i bit di modifica (se presenti) e la lista dei blocchi (ovvero il loro indirizzo) via via eliminati durante il rimpiazzamento ed inoltre in corrispondenza di quale riferimento il blocco e' eliminato.

3) [6] In un processore MIPS con pipeline determinare i cicli necessari per eseguire due iterazioni per il seguente frammento di codice sia nel caso di propagazione abilitata che di propagazione disabilitata. Nota: e' presente 1 delay-slot e l'accesso ai registri nella fase di decodifica e write-back possono essere sovrapposte.

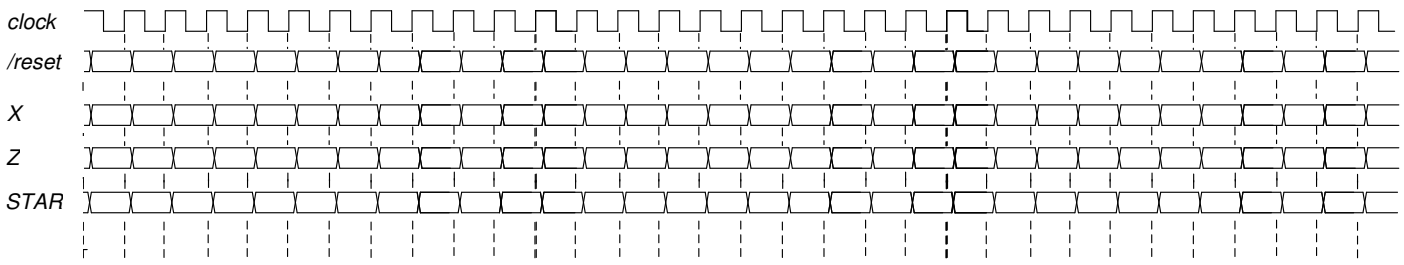
```
     add $1, $2, $3
L1:  lw  $4,0($1)
     lw  $5,0($1)
     bne $4, $5, L1
     nop
```

4) [4] Spiegare tramite un diagramma il funzionamento della tecnica di gestione dell'I/O con interrupt vettorizzato.

5) [4] Spiegare tramite un diagramma e un esempio il funzionamento della gestione dell'I/O a polling

6) [8] Sintetizzare una rete sequenziale utilizzando il modello di Mealy ritardato con un ingresso X su un bit e una uscita Z su un bit che riconosca le sequenze interallacciate 1,0,0,1. Rappresentare la macchina a stati finiti per tale rete logica, la tabella delle transizioni, le equazioni booleane delle reti CN1 e CN2 e il circuito sequenziale sincronizzato basato su flip-flop D.

7) [8] Descrivere e sintetizzare in Verilog la rete sequenziale descritta nell'esercizio 6 e il modulo TopLevel con sequenza di ingresso 0,0,1,1,0,0,1,0,0,1,0,0,1,0,0,0,1,0,0,1,0,0,0,0. **Tracciare il diagramma di temporizzazione** come verifica della correttezza dell'unità. Nota: si puo' svolgere l'esercizio su carta oppure con ausilio del simulatore salvando una copia dell'output (diagramma temporale) e del programma Verilog su USB-drive del docente.



**COMPITO di ARCHITETTURA DEI CALCOLATORI del 21-07-2017** **MATRICOLA**_____

**COGNOME**_____

**NOME**_____

**Instructions**

| Instruction | Example | Meaning | Comments |
|---|---|---|---|
| add | `add/addu $1,$2,$3` | $1 = $2 + $3 | (signed/unsigned) 3 operands; exception possible |
| subtract | `sub/subu $1,$2,$3` | $1 = $2 - $3 | (signed/unsigned) 3 operands; exception possible |
| add immediate | `addi/addiu $1,$2,100` | $1 = $2 + 100 | (signed/unsigned) + constant ; exception possible |
| multiplication | `mult/multu $1, $2` | Hi,Lo= $1 x $2 | (signed/unsigned) 64-bit Product ; result in Hi,Lo |
| division | `div/divu $1, $2` | Hi= $1 % $2, Lo = $1 / $2 | (signed/unsigned) division |
| move from Hi / move from Lo | `mfhi/mflo $1` | $1 = Hi ($1 = Lo) | Create copy of Hi (Create a 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 (l=logical,a=arithmetic) | `srl/sra $1,$2,10` | $1 = $2 >> 10 | Shift right by constant (in the arithmetic case, the sign is always preserved) |
| load word / load byte | `lw/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 / store byte | `sw/sb $1,100($2)` | Memory[$2+100] = $1 | Data from register to memory |
| load address | `la $1,var` | $1 = &var | Load variable address |
| branch unconditional | `b 100` | go to PC+4+100 | PC relative branch |
| 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 |
| no operation | `nop` | Do nothing | Do nothing |
| load-linked | `ll $1,100($2)` | $1=Memory[$2+100] | Read and start to monitor the given memory location |
| store-conditional | `sc $1,100($2)` | Memory[$2+100]=$1 or → | return 0 if a coherence action happens since the previous ll ($1 must be different from 0) |
| 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 |
| neg.s neg.d | `neg.x $f0,$f2` | $f0= – ($f2) | Single and double precision opposite 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/mfc1 | `mtc1/mfc1 $1,$f2` | $f2=$1 / $1=$f2 | Data from gen.reg. $1 to C1 reg. $f2 (no conversion) / and viceversa |
| ctc1/cfc1 | `ctc1/cfc1 $1,$cf2` | $cf2=$1 / $1=$cf2 | Data from gen.reg. to C1 CONTROL reg. (no conversion) ) / and viceversa |
| 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] | Data from FP (C1) register to memory |
| store floating point (32bit) | `swc1 $f0,0($1)` | Memory[$1]←$f0 | Data from memory to FP (C1) register |
| 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 | Reg. Num. | Usage |
|---|---|---|
| **$zero** | 0 | The constant value 0 |
| **$s0-$s7** | 16-23 | Saved |
| **$t0-$t9** | 8-15,24-25 | Temporaires |
| **$a0-$a3** | 4-7 | Arguments |

| Name | Reg.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 |

| Reg. Num. | Usage |
|---|---|
| **$f0, $f2** | Return values |
| **$f12,$f14** | Function arguments |
| **$f20,$f22,$f24,$f26,$f28,$f30** | Saved registers |
| **$f4,$f6,$f8,$f10,$f16,$f18** | Temporaries registers |

**System calls**

| Service Name | Service Num. ($v0) | INPUT Arguments | OUTPUT Arguments |
|---|---|---|---|
| **print_int** | 1 | $a0=integer to print | --- |
| **print_float** | 2 | $f12=float to print | --- |
| **print_double** | 3 | ($f12,$f13)=double to print | --- |
| **print_string** | 4 | $a0=address of ASCIIZ string to print | --- |
| **read_int** | 5 | --- | $v0=integer |
| **read_float** | 6 | --- | $f0=float |
| **read_double** | 7 | --- | $f0-f1=double |
| **read_string** | 8 | $a0=address of input buffer, $a1=max characters to read | --- |
| **sbrk** | 9 | $a0=Number of bytes to be allocated | $v0=pointer to the allocated memory |
| **exit** | 10 | --- | --- |

**COMPITO di ARCHITETTURA DEI CALCOLATORI del 21-07-2017** **MATRICOLA_____**

**TRACCIA DELLA SOLUZIONE** **COGNOME_____**

**NOME_____**

## ESERCIZIO 1)

```
.data
count: .word 0
vals: .word 12, 56, 45, 78, 2, 6, 5, 8, 0, 0, 0,0
spc2: .asciiz "  "
nl1: .asciiz "\n"
ii:  .word 0
jj:  .word 0
kk:  .word 0

.text
.globl main
#-----------------------------------------------
multiply:
        addi $sp,$sp,-36 # allocate frame,...
        sw  $a3,32($sp)
        sw  $a2,28($sp)
        sw  $a1,24($sp)
        sw  $a0,20($sp)
        sw  $fp,16($sp)  # save old-fp
        sw  $ra,12($sp)  # save old-ra
        sw  $s6,8($sp)   # used for 7th parm
        sw  $s5,4($sp)   # used for 6th parm
        sw  $s4,0($sp)   # used for 5th parm

        lw  $s4,36($sp)  # load 5th parm
        lw  $s5,40($sp)  # load 6th parm
        lw  $s6,44($sp)  # load 7th parm

        # if1
        la  $t0,ii       # &i
        lw  $t6,0($t0)   # i
        slt $t9,$t6,$a0  # i<?m1: else1
        bne $t9,$0,else1
        j endif1
else1:
        slt $t9,$t6,$a0  # i<?m1: if not endif1
        beq $t9,$0,endif1

        # if2
        la  $t0,jj       # &j
        lw  $t7,0($t0)   # j
        slt $t9,$t7,$s4  # j<?n2: if not endif2
        beq $t9,$0,endif2

        # if3
        la  $t0,kk       # &k
        lw  $t8,0($t0)   # k
        slt $t9,$t8,$a1  # k<?n1: if not endif3
        beq $t9,$0,endif3

        # a[i][k] ...
        add  $t0,$t6,$t6 # 2*i
        add  $t0,$t0,$t8 # 2*i+k
        sll  $t0,$t0,2       # 4*(2*i+k)
        add  $t0,$t0,$a2 # &a+4*(2*i+k)
        lw   $t1,0($t0)

        # b[k][j] ...
        add  $t0,$t8,$t8 # 2*k
        add  $t0,$t0,$t7 # 2*k+j
        sll  $t0,$t0,2       # 4*(2*k+j)
        add  $t0,$t0,$s5 # &b+4*(2*k+j)
        lw   $t2,0($t0)

        # c[i][j] ...
        add  $t0,$t6,$t6 # 2*i
        add  $t0,$t0,$t7 # 2*i+j
        sll  $t0,$t0,2       # 4*(2*i+j)
        add  $t0,$t0,$s6 # &c+4*(2*i+j)
        lw   $t3,0($t0)  # load c

        #
        mult $t1,$t2     # a[][]*b[][]
        mflo $t1
        add  $t3,$t3,$t1 # a[][]*b[][]+c[][]
        sw   $t3,0($t0)  # store c[][]
        addi $t8,$t8,1   # k++
        la   $t0,kk      # &k
        sw   $t8,0($t0)  # store k

        addi $sp,$sp,-12 # push extra params
        sw   $s6,8($sp)
        sw   $s5,4($sp)
        sw   $s4,0($sp)
        jal multiply
        addi $sp,$sp,12
endif3:
        la   $t0,kk      # &k
        sw   $0,0($t0)   # k=0
        la   $t0,jj      # &j
        lw   $t1,0($t0)  # load j
        addi $t1,$t1,1   # j++
        sw   $t1,0($t0)  # store j

        addi $sp,$sp,-12 # push extra params
        sw   $s6,8($sp)
        sw   $s5,4($sp)
        sw   $s4,0($sp)
        jal multiply
        addi $sp,$sp,12
endif2:
        la   $t0,jj      # &j
        sw   $0,0($t0)   # store j=0
        la   $t0,ii      # &i
        lw   $t1,0($t0)  # load i
        addi $t1,$t1,1   # i++
        sw   $t1,0($t0)  # store i

        addi $sp,$sp,-12 # push extra params
        sw   $s6,8($sp)
        sw   $s5,4($sp)
        sw   $s4,0($sp)
        jal multiply
        addi $sp,$sp,12
endif1:
        lw   $s4,0($sp)
        lw   $s5,4($sp)
        lw   $s6,8($sp)
        lw   $ra,12($sp)
        lw   $fp,16($sp)
        lw   $a0,20($sp)
        lw   $a1,24($sp)
        lw   $a2,28($sp)
        lw   $a3,32($fp)
        addi $sp,$sp,36
        j $ra
#-----------------------------------------------
display:
        addi $sp,$sp,-28 # allocate frame,
                 # save-s0,save-s1,a0,a1,a2
        sw   $a2,24($sp) # save a2
        sw   $a1,20($sp) # save a1
        sw   $a0,16($sp) # save a0
        sw   $fp,12($sp) # save old-fp
        sw   $ra,8($sp)  # save old-ra
        sw   $s1,4($sp)  # save s1
        sw   $s0,0($sp)  # save s0
        add  $fp,$sp,$0  # fp=sp

        addi $s0,$0,0        # i=0
fori1:
        slt $t9,$s0,$a0  # i<?m1: ifnot endfori
        beq $t9,$0,endfori1
        #----- fori1 body start
        addi $s1,$0,0        # j=0
forj1:
        slt $t9,$s1,$a1  # j<?n2: ifnot endforj
        beq $t9,$0,endforj1
        #----- forj1 body start
        sll $t2,$s0,1    # i*2
        add $t2,$t2,$s1      # i*2+j
        sll $t2,$t2,2    # 4*(i*2+j)
        add $t2,$a2,$t2  # &c+4*(i*2+j)
        lw  $a0,0($t2)   # t3:c[i][j]
        addi $v0,$0,1    # service-1: print_int
        syscall
        la  $a0,spc2     # pointer to string
        addi $v0,$0,4    # service-4: print_str
        syscall
    #----- forj1 body end
nextforj1:
        addi $s1,$s1,1   # ++j
        lw   $a1,20($sp) # restore a1
        j forj1

endforj1:
    #----- fori1 body end
        addi $s0,$s0,1   # i++
        la   $a0,nl1     # print just newlinw
        addi $v0,$0,4    # print_str
        syscall
        lw   $a0,16($sp) # restore a0
        j fori1
endfori1:
        addi $sp,$sp,28
        lw   $s0,0($sp)  # restore s0
        lw   $s1,4($sp)  # restore s1
        lw   $ra,8($fp)  # restore ra
        lw   $fp,12($fp) # restore fp
        j $ra

#-----------------------------------------------
main:
        addi $sp,$sp,-48 # allocate
                 # "int x[2][2], y... z..."
        add  $fp,$sp, $0 # fp=sp
        la   $t0,vals    # initialize x,y,z
        add  $t1,$fp,$0  # pointer to x[0][0]
        add  $t2,$t0,48
loop1:
        lw   $t3,0($t0)  # load val
        sw   $t3,0($t1)  # store val
        add  $t1,$t1,4   # increment pointers
        add  $t0,$t0,4   #
        bne  $t0,$t2,loop1 # for all elements

        addi $a0,$0,2        # m1
        addi $a1,$0,2        # n1
        add  $a2,$fp,$0  # &x
        addi $a3,$0,2        # n1
        addi $sp,$sp,-12  # allocate space
                          # for extra parms
        addi $t0,$fp,32      # &z
        sw   $t0,8($sp)  # store 7th parm
        addi $t0,$fp,16      # &y
        sw   $t0,4($sp)  # store 6th parm
        sw   $a3,0($sp)  # store the 5th parm
        jal multiply

        addi $sp,$sp,12  # restore stack
                         # pointer for parms
        addi $a0,$0,2    # m1
        addi $a1,$0,2    # n2
        addi $a2,$fp,32      # &z
        jal display

        addi $sp,$sp,48  # deallocate "x, y, z"
        addi $v0,$0,10   # service 10: exit
        syscall
```

Console

```
304  520
480  894
```

## ESERCIZIO 2)

```
A = 4, B = 8, C = 128, RP = FIFO, Thit = 4, Tpen = 40
Read 20 references.
=== T X    XM  XT XS XB H [SET]:USAGE [SET]:MODIF [SET]:TAG
=== R 55    6   1  2  7 0 [2]:3,0,0,0 [2]:0,0,0,0 [2]:1,-,-,-
=== W 173  21   5  1  5 0 [1]:3,0,0,0 [1]:0,0,0,0 [1]:5,-,-,-
=== R 115  14   3  2  3 0 [2]:2,3,0,0 [2]:0,0,0,0 [2]:1,3,-,-
=== W 119  14   3  2  7 1 [2]:2,3,0,0 [2]:0,1,0,0 [2]:1,3,-,-
=== R 222  27   6  3  6 0 [3]:3,0,0,0 [3]:0,0,0,0 [3]:6,-,-,-
=== W 947 118  29  2  3 0 [2]:1,2,3,0 [2]:0,1,0,0 [2]:1,3,29,-
=== R 618  77  19  1  2 0 [1]:2,3,0,0 [1]:0,0,0,0 [1]:5,19,-,-
=== W 449  56  14  0  1 0 [0]:3,0,0,0 [0]:0,0,0,0 [0]:14,-,-,-
=== R 534  66  16  2  6 0 [2]:0,1,2,3 [2]:0,1,0,0 [2]:1,3,29,16
=== W 748  93  23  1  4 0 [1]:1,2,3,0 [1]:0,0,0,0 [1]:5,19,23,-
=== R 877 109  27  1  5 0 [1]:0,1,2,3 [1]:0,0,0,0 [1]:5,19,23,27
=== W 919 114  28  2  7 0 [2]:3,0,1,2 [2]:0,1,0,0 [2]:28,3,29,16   (out: XM=6 XT=1 XS=2 )
=== R 283  35   8  3  3 0 [3]:2,3,0,0 [3]:0,0,0,0 [3]:6,8,-,-
=== W 143  17   4  1  7 0 [1]:3,0,1,2 [1]:0,0,0,0 [1]:4,19,23,27   (out: XM=21 XT=5 XS=1 )
=== R 591  73  18  1  7 0 [1]:2,3,0,1 [1]:0,0,0,0 [1]:4,18,23,27   (out: XM=77 XT=19 XS=1 )
=== W 644  80  20  0  4 0 [0]:2,3,0,0 [0]:0,0,0,0 [0]:14,20,-,-
=== R 770  96  24  0  2 0 [0]:1,2,3,0 [0]:0,0,0,0 [0]:14,20,24,-
=== W 845 105  26  1  5 0 [1]:1,2,3,0 [1]:0,0,0,0 [1]:4,18,26,27   (out: XM=93 XT=23 XS=1 )
=== R 961 120  30  0  1 0 [0]:0,1,2,3 [0]:0,0,0,0 [0]:14,20,24,30
=== W 194  24   6  0  2 0 [0]:3,0,1,2 [0]:0,0,0,0 [0]:6,20,24,30   (out: XM=56 XT=14 XS=0 )
-------------------------------------
P1 Nmiss=19 Nhit=1 Nref=20 mrate=0.950000 AMAT=42
```

**COMPITO di ARCHITETTURA DEI CALCOLATORI del 21-07-2017** **MATRICOLA_____**

**TRACCIA DELLA SOLUZIONE** **COGNOME_____**

**NOME_____**

ESERCIZIO 3)

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| add $1, $2, $3 | F | D | X | M | W | | | | | | | | | | | | | | |
| lw $4,0($1) | | F | --- | --- | D | X | M | W | | | | | | | | | | | |
| lw $5,0($1) | | | | | F | D | X | M | W | | | | | | | | | | |
| bne $6, $5, L1 | | | | | | F | --- | --- | D | X | M | W | | | | | | | |
| nop | | | | | | | | | F | D | X | M | W | | | | | | |
| lw $4,0($1) | | | | | | | | | | F | D | X | M | W | | | | | |
| lw $5,0($1) | | | | | | | | | | | F | D | X | M | W | | | | |
| bne $6, $5, L1 | | | | | | | | | | | | F | --- | --- | D | X | M | W | |
| nop | | | | | | | | | | | | | | | F | D | X | M | W |

Senza Forwarding

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| add $1, $2, $3 | F | D | X | M | W | | | | | | | | | | | | | | |
| lw $4,0($1) | | F | D | X | M | W | | | | | | | | | | | | | |
| lw $5,0($1) | | | F | D | X | M | W | | | | | | | | | | | | |
| bne $6, $5, L1 | | | | F | --- | --- | D | X | M | W | | | | | | | | | |
| nop | | | | | | | F | D | X | M | W | | | | | | | | |
| lw $4,0($1) | | | | | | | | F | D | X | M | W | | | | | | | |
| lw $5,0($1) | | | | | | | | | F | D | X | M | W | | | | | | |
| bne $6, $5, L1 | | | | | | | | | | F | --- | --- | D | X | M | W | | | |
| nop | | | | | | | | | | | | | F | D | X | M | W | | |

Forwarding

Senza forwarding: 19 cicli. Con forwarding 17 cicli.

7)

```
module Toplevel;
  reg reset_; initial begin reset_=0; #22 reset_=1; #300; $stop; end
  reg clock; initial clock=0; always #5 clock<=(!clock);
  reg   X;
  wire  z=Xxx.z;
  wire [1:0] STAR=Xxx.STAR;
  initial begin X=0;
    wait(reset_==1);
    @(posedge clock); X<=0;@(posedge clock); X<=0;@(posedge clock); X<=1;
    @(posedge clock); X<=1;@(posedge clock); X<=0;@(posedge clock); X<=0;
    @(posedge clock); X<=1;@(posedge clock); X<=0;@(posedge clock); X<=0;
    @(posedge clock); X<=1;@(posedge clock); X<=0;@(posedge clock); X<=0;
    @(posedge clock); X<=1;@(posedge clock); X<=0;@(posedge clock); X<=0;
    @(posedge clock); X<=0;@(posedge clock); X<=1;@(posedge clock); X<=0;
    @(posedge clock); X<=0;@(posedge clock); X<=1;@(posedge clock); X<=0;
    @(posedge clock); X<=0;@(posedge clock); X<=0;@(posedge clock); X<=0;
      $finish;
    end
  XXX Xxx(X,Z,clock,reset_);
endmodule

module XXX(x,z,clock,reset_);
input clock,reset_,x;
output z;
reg [1:0] STAR;
reg OUTR;
parameter S0='B00 , S1='B01 , S2='B10 , S3='B11;
always @(reset_==0) begin STAR<=0 ; end
assign z=OUTR;
always @(posedge clock) if (reset_==1)
  casex(STAR)
    S0: begin OUTR<=0; STAR<=(x==0)?S0:S1; end
    S1: begin OUTR<=0; STAR<=(x==0)?S2:S1; end
    S2: begin OUTR<=0; STAR<=(x==0)?S3:S1; end
    S3: begin OUTR<=(x==1)?1:0; STAR<=(x==0)?S0:S1; end
  Endcase
```