COMPITINO di ARCHITETTURA DEI CALCOLATORI del 11-01-2017

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

1) [18] Ricordando che i due bit meno signifcativi del registro FCSR (registro 31 del coprocessore 1 nell'architetura MIPS) determinano la modalita' di arrotondamento (Rounding Mode o RM - 00=to-nearest, 01=toward-zero, 02=toward+inf, 03=toward-inf), scrivere un programma in Assembly MIPS (facendo riferimento solo ed unicamente alla tabella sottostante e rispettando le convenzioni dell'ABI MIPS) tale che: legga 3 numeri interi inseriti da tastiera e stampi a video il quoziente dei primi due numeri sia in formato decimale in virgola mobile che in formato binario in singola precisione (nota: nel caso binario devono essere stampate 1 cifra per il segno, 8 cifre per l'esponente e 23 cifre per la mantissa). Il terzo numero dovrà essere utilizzato per selezionare tramite il registro FCSR la modalità di arrotondamento. L'esercizio potrà essere svolto sia su carta che con l'ausilio del simulatore SPIM salvando una copia dell'output (screenshot della console) e del programma MIPS su USB-drive del docente.

2) [9] Si consideri una cache a 2 livelli in cui il primo livello ha dimensione 256B, ad accesso diretto, di tipo write-back/write-non-allocate. La dimensione del blocco è 64 byte sia per il primo che per il secondo livello, il tempo di accesso alla cache è 2 ns e la penalità in caso di miss (per accedere al livello 2) è pari a 8 ns, la politica di rimpiazzamento è LRU. Il secondo livello ha dimensione 1024B, a 2 vie, ancora di tipo write-back/write-non-allocate e la penalità in caso di miss è 90 ns. Il processore effettua i seguenti accessi in cache (primo livello), ad indirizzi al byte: 190 210 412 710 315 221 71 65 90 143 81 57 133 61 98 75 64 259 130 67 70 25. Tali accessi sono alternativamente letture e scritture. Per la sequenza data, ricavare il tempo medio di accesso finale per ciascun livello e quello effettivo visto dal processore, riportare per ciascun livello di cache al termine: i tag della configurazione finale, 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) [5] Determinare la rappresentazione IEEE-754 in singola precisione del risultato dell'operazione 3/5 in ciascuno dei 4 casi di arrotondamento (to-nearest, toward-zero, toward+inf, toward-inf).

4) [4] Spiegare con un diagramma e con almeno un esempio di una istruzione ciascuna delle 5 modalità di indirizzamento del processore MIPS.

5) [4] Spiegare il funzionamento della paginazione a tre livelli con un diagramma ed un esempio numerico nel caso di spazio di indirizzamento virtuale a 64 bit, spazio di indirizzamento fisico a 42 bit, pagine di 4K e 10 bit per l'offset di ciascuno dei tre livelli.

6) [8] Sintetizzare una rete sequenziale utilizzando il modello di Moore con un ingresso X su un bit e una uscita Z su un bit che funziona nel seguente modo: devono essere riconosciute le sequenze interallacciate 1,1,0 e 1,0,1; l'uscita Z va a 1 se è presente una delle due sequenze. Rappresentare per tale macchina a stati finiti, 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 nel'esercizio 6. Tracciare il diagramma di temporizzazione come verifica della correttezza dell'unità XXX (il modulo TopLevel e' riportato in calce). 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.

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

COMPITINO di ARCHITETTURA DEI CALCOLATORI del 11-01-2017 **MATRICOLA**_____

(SOLUZIONE) (AGGIORNATA 22/20/2018) **COGNOME**_____

NOME_____

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	---	---

```
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 [1:0] Z;
wire [2:0] STAR=Xxx.STAR;
wire Z1=Xxx.z[1];
wire Z0=Xxx.z[0];
initial begin X=0;
wait(reset_==1); #5
@(posedge clock); X<=0; @(posedge clock); X<=0; @(posedge clock); X<=1; @(posedge clock); X<=1;
@(posedge clock); X<=0; @(posedge clock); X<=1; @(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<=1; @(posedge clock); X<=1; @(posedge clock); X<=1; @(posedge clock); X<=0;
@(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
```

COMPITINO di ARCHITETTURA DEI CALCOLATORI del 11-01-2017 **MATRICOLA**_____

(SOLUZIONE) (AGGIORNATA 22/20/2018) **COGNOME**_____

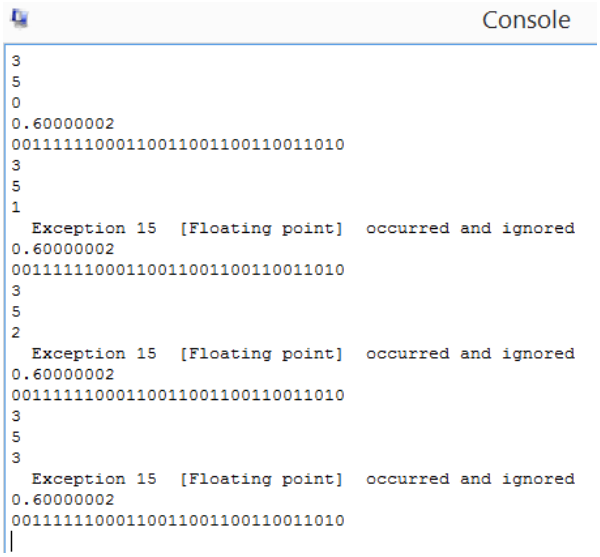
NOME_____

SI RIPORTA NEL SEGUITO UNA SINTETICA TRACCIA DI UNA POSSIBILE SOLUZIONE DEGLI ESERCIZI

ESERCIZIO 1)

```
-------------------------------------------------------------------------
.data
cr: .asciiz "\n"
.text
.globl main
main:
          addi      $v0, $0, 5
          syscall                     # leggo a
          mtc1      $v0, $f0          # f0=a
          cvt.s.w $f0, $f0
          addi      $v0, $0, 5
          syscall                     # leggo b
          mtc1      $v0, $f1          # f1=b
          cvt.s.w $f1, $f1
          addi      $v0, $0, 5
          syscall                     # leggo r (modalita' arrotandamento)

          # setup modalita' di arrotondamento
          andi    $v0, $v0, 3         # prelevo solo i 3 bit meno significativi (lsb)
          cfc1    $v1, $31            # leggo FCSR
          srl     $v1, $v1, 2
          sll     $v1, $v1, 2         # pulisco i 2 lsb
          add     $v1, $v1, $v0       # imposto i 2 lsb (RM)
          ctc1    $v1, $31            # scrivo FCSR

          # calcolo quoziente a/b
          div.s     $f12, $f0, $f1    # a/b nella FPU
          mfc1      $s0, $f12         # a/b nella CPU

          addi      $v0, $0, 2
          syscall                     # stampo a/b in decimale
          la                  $a0, cr
          addi    $v0, $0, 4
          syscall                     # carriage return

          # stampo a/b in binario (single-precision=<1-bit=sign,8-bits=exponent,23-bit=mantissa>)
          addi      $s1, $0, 32       # bit to shift
ciclo:
          addi      $s1, $s1, -1
          srlv      $a0, $s0, $s1     # metto il bit s1 in posizione 0
          andi      $a0, $a0, 1       # isolo il bit
          addi      $v0, $0, 1
          syscall                     # stampo tale bit
          bne       $s1, $0, ciclo

          la                  $a0, cr
          addi    $v0, $0, 4
          syscall                     # carriage return
          addi      $v0, $0, 10
          syscall                     # exit
-------------------------------------------------------------------------
```

Output generato dal simulatore SPIM (notare che mentre nel caso RM=0 l'output e'corretto, nei casi RM=1,2,3 l'output generato non e' corretto per un difetto del simulatore - Ovviamente non e' considerato errore dell'esercizio produrre questa stampa!)



ESERCIZIO 3)
(per una spiegazione piu' dettagliata vedere per es. soluzione del compito del 02-11-07)

3/5=0.6 → 1.2 * 2 ^ -1
RM=0 (NEAREST) → SEGNO=0 ESPONENTE=01111110 MANTISSA=0011 0011 0011 0011 0011 010
RM=1 (TOWARD-0) → SEGNO=0 ESPONENTE=01111110 MANTISSA=0011 0011 0011 0011 0011 001
RM=2 (TOWARD+INF) → SEGNO=0 ESPONENTE=01111110 MANTISSA=0011 0011 0011 0011 0011 010
RM=3 (TOWARD-INF) → SEGNO=0 ESPONENTE=01111110 MANTISSA=0011 0011 0011 0011 0011 001

ESERCIZIO 5)
Soluzione disponibile dal docente su richiesta.

COMPITINO di ARCHITETTURA DEI CALCOLATORI del 11-01-2017 **MATRICOLA**_____

(SOLUZIONE) (AGGIORNATA 22/20/2018) **COGNOME**_____

NOME_____

ESERCIZIO 2)

```
A = 1
B = 64
C = 256
RP = LRU
Thit = 2
Tpen = 8
File: c1170111-2L.sh_001000.din
Read 22 references.
=== T    X   XM  XT  XS  XB  H [SET]:USAGE [SET]:MODIF  [SET]:TAG
=== R  190    2   0   2  62  0 [2]:0 [2]:0   [2]:0
=== W  210    3   0   3  18  0 [3]:0 [3]:0   [3]:0
=== R  412    6   1   2  28  0 [2]:0 [2]:0   [2]:1 (out: XM=2  XT=0  XS=2 )
=== W  710   11   2   3   6  0 [3]:0 [3]:0   [3]:2 (out: XM=3  XT=0  XS=3 )
=== R  315    4   1   0  59  0 [0]:0 [0]:0   [0]:1
=== W  221    3   0   3  29  0 [3]:0 [3]:0   [3]:0 (out: XM=11 XT=2  XS=3 )
=== R   71    1   0   1   7  0 [1]:0 [1]:0   [1]:0
=== W   65    1   0   1   1  1 [1]:0 [1]:1   [1]:0
=== R   90    1   0   1  26  1 [1]:0 [1]:1   [1]:0
=== W  143    2   0   2  15  0 [2]:0 [2]:0   [2]:0 (out: XM=6  XT=1  XS=2 )
=== R   81    1   0   1  17  1 [1]:0 [1]:1   [1]:0
=== W   57    0   0   0  57  0 [0]:0 [0]:0   [0]:0 (out: XM=4  XT=1  XS=0 )
=== R  133    2   0   2   5  1 [2]:0 [2]:0   [2]:0
=== W   61    0   0   0  61  1 [0]:0 [0]:1   [0]:0
=== R   98    1   0   1  34  1 [1]:0 [1]:1   [1]:0
=== W   75    1   0   1  11  1 [1]:0 [1]:1   [1]:0
=== R   64    1   0   1   0  1 [1]:0 [1]:1   [1]:0
=== W  259    4   1   0   3  0 [0]:0 [0]:0   [0]:1 (out: XM=0  XT=0  XS=0 )
=== R  130    2   0   2   2  1 [2]:0 [2]:0   [2]:0
=== W   67    1   0   1   3  1 [1]:0 [1]:1   [1]:0
=== R   70    1   0   1   6  1 [1]:0 [1]:1   [1]:0
=== W   25    0   0   0  25  0 [0]:0 [0]:0   [0]:0 (out: XM=4  XT=1  XS=0 )
-------------------------------------
P1 Nmiss=11   Nhit=11  Nref=22  mrate=0.500000   AMAT=6
```

TAG DELLA CONFIGURAZIONE FINALE

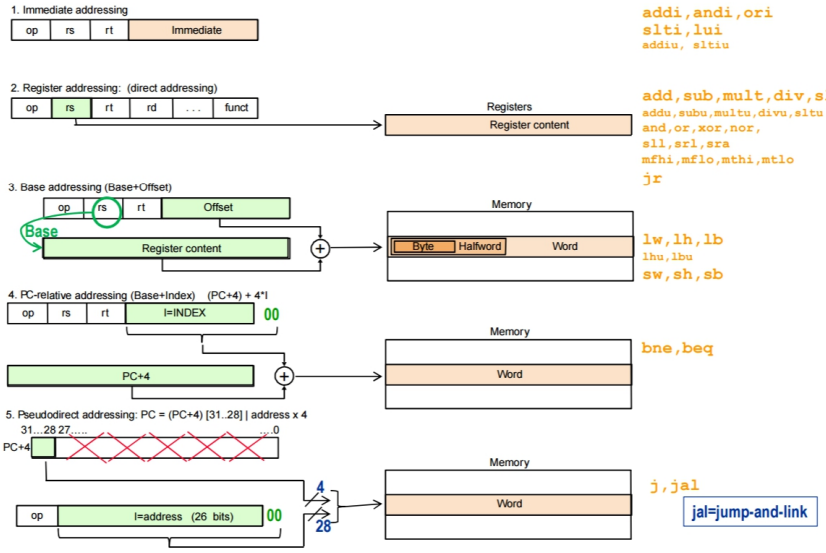
SET \ VIA	0
S0	0
S1	0
S2	0
S3	0

```
A = 2
B = 64
C = 1024
RP = LRU
Thit = 8
Tpen = 90
File: c1170111-2L.shL2_001000.din
Read 11 references.
=== T    X   XM  XT  XS  XB  H [SET]:USAGE [SET]:MODIF  [SET]:TAG
=== R  190    2   0   2  62  0 [2]:1,0 [2]:0,0   [2]:0,-
=== W  210    3   0   3  18  0 [3]:1,0 [3]:0,0   [3]:0,-
=== R  412    6   0   6  28  0 [6]:1,0 [6]:0,0   [6]:0,-
=== W  710   11   1   3   6  0 [3]:0,1 [3]:0,0   [3]:0,1
=== R  315    4   0   4  59  0 [4]:1,0 [4]:0,0   [4]:0,-
=== W  221    3   0   3  29  1 [3]:1,0 [3]:1,0   [3]:0,1
=== R   71    1   0   1   7  0 [1]:1,0 [1]:0,0   [1]:0,-
=== W  143    2   0   2  15  1 [2]:1,0 [2]:1,0   [2]:0,-
=== W   57    0   0   0  57  0 [0]:1,0 [0]:0,0   [0]:0,-
=== W  259    4   0   4   3  1 [4]:1,0 [4]:1,0   [4]:0,-
=== W   25    0   0   0  25  1 [0]:1,0 [0]:1,0   [0]:0,-
-------------------------------------
P1 Nmiss=7    Nhit=4   Nref=11   mrate=0.636364    AMAT=65.2727
---
TEMPO di accesso per ciascun livello:
L1: P1 Nmiss=11    Nhit=11   Nref=22   mrate=0.500000    AMAT=6   ns
L2: P1 Nmiss=7     Nhit=4    Nref=11   mrate=0.636364    AMAT=65.2727 ns
---
TEMPO di accesso visto complessivamente dal processore:
AMAT=H1+m1*(H2+m2*P2)=2+0.500000*(8+0.636364*90)=34.636380   ns
```

TAG DELLA CONFIGURAZIONE FINALE

SET \ VIA	0	1
S0	0	-
S1	0	-
S2	0	-
S3	0	-
S4	0	1
S5	-	-
S6	0	-
S7	-	-

ESERCIZIO 4)



COMPITINO di ARCHITETTURA DEI CALCOLATORI del 11-01-2017 MATRICOLA_____

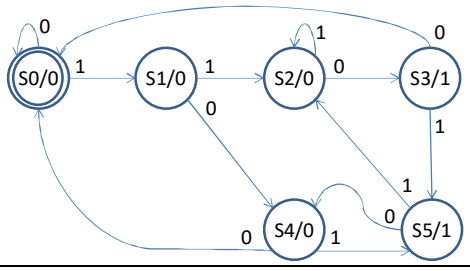
(SOLUZIONE) (AGGIORNATA 22/20/2018) COGNOME_____

NOME_____

ESERCIZIO 6)

In corrispondenza del pattern X_{t-2},X_{t-1},X_t = 1,1,0 oppure 1,0,1 ottengo → Z_{t+1} = 1; (ricordare che e' richiesto Moore).

Diagramma degli Stati



Schema di riferimento per Moore

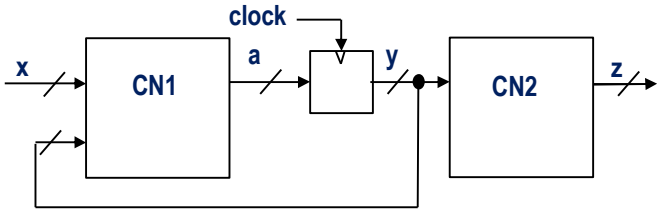


Tabella delle Transizioni (Tabella degli Stati)

TABELLA DELLE TRANSIZIONI

STATO ATTUALE \ x	0	1	z
S0	S0	S1	0
S1	S4	S2	0
S2	S3	S2	0
S3	S0	S5	1
S4	S0	S5	0
S5	S4	S2	1
S6	X	X	X
S7	X	X	X

STATO SUCCES-SIVO/USCITA

Scelta della codifica degli Stati

STATO \ CODIFICA $y_2y_1y_0$
S0 000
S1 001
S2 011
S3 010
S4 100
S5 101
S6 111
S7 110

(completare il n. di stati a una potenza di 2)

Rappresentazione della Tabella degli Stati in Formato più comodo per la sintesi

OVVERO

y_1y_0 \ y_2x	00	01	11	10
00	S0	S1	S5	S0
01	S4	S2	S2	S4
11	S3	S2	X/X	X/X
10	S0	S5	X/X	X/X

STATO SUCCES-SIVO/USCITA

OVVERO

y_1y_0 \ y_2x	00	01	11	10
00	000	001	101	000
01	100	011	011	100
11	010	011	XXX	XXX
10	000	101	XXX	XXX

a2,a1,a0

Sintesi della variable 'a' (stato successivo o ingresso dello STATUS REGISTER -- STAR):

y_1y_0 \ y_2x	00	01	11	10
00	0	0	1	0
01	1	0	0	1
11	0	0	X	X
10	0	1	X	X

a_2

a2=/x/y1y0+/xy2/y0+/x/y1y0+xy1/y0

y_1y_0 \ y_2x	00	01	11	10
00	0	0	0	0
01	0	1	1	0
11	1	1	X	X
10	0	0	X	X

a_1

a1=xy0+y1y0

y_1y_0 \ y_2x	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	0	1	X	X
10	0	1	X	X

a_0

a0=x

Sintesi della variable 'z' (uscita):

y_2 \ y_1y_0	00	01	11	10
0	0	0	X	1
1	0	1	X	0

z

z=/y2y1+y2y0

ESERCIZIO 7)

(diagramma degli stati come nell'esercizio 6)

```
module XXX(x,z,clock,reset_);
input         clock,reset_,x;
output [1:0] z;
reg    [2:0] STAR;
parameter
S0='B000,S1='B001,S2='B010,S3='B011,S4='B100,S5='B101;
always @(reset_==0) #1 begin STAR<=S0; end
assign z=(STAR==S3||STAR==S5)?1:0;

always @(posedge clock) if(reset_==1) #3
casex(STAR)
        S0:begin STAR<=(x==1)?S1:S0; end
        S1:begin STAR<=(x==1)?S2:S4; end
        S2:begin STAR<=(x==1)?S2:S3; end
        S3:begin STAR<=(x==1)?S5:S0; end
        S4:begin STAR<=(x==1)?S5:S0; end
        S5:begin STAR<=(x==1)?S2:S4; end
endcase
endmodule
```