COMPITO di ARCHITETTURA DEI CALCOLATORI del 04-11-2016

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": 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) [12] Scrivere in assembly MIPS l'implementazione della funzione **`int atoi(char *s)`** che trasforma la stringa puntata dal parametro di ingresso s in un intero a 32 bit restituito da tale funzione (utilizzando solo e unicamente istruzioni dalla tabella sottostante e rispettando le convenzioni di utilizzazione dei registri e l'ABI dell'assembly MIPS riportate qua sotto, per riferimento). Nota: svolgere direttamente su carta (senza il simulatore).

2) [6] Per la funzione realizzata al punto precedente calcolare il tempo di esecuzione supponendo che tale funzione sia in esecuzione su un processore MIPS con frequenza di clock di 1GHz, e assumendo che le istruzioni aritmetico-logiche-jump (ALJ) richiedano un ciclo di clock, le istruzioni di tipo branch (B) 3 cicli di clock e le istruzioni load-store (LS) 90 cicli di clock

3) [8] Si consideri una cache di dimensione 128B e a 4 vie di tipo write-back/write-non-allocate. La dimensione del blocco e' 16 byte, il tempo di accesso alla cache e' 1 ns e la penalita' in caso di miss e' pari a 99 ns, la politica di rimpiazzamento e' LRU. Il processore effettua i seguenti accessi in cache, ad indirizzi al byte: 3123, 7456, 4567, 1190, 7264, 2789, 2893, 9088, 2019, 1290, 2227, 3902, 8903, 8890, 3160, 3189, 3197, 3201, 3207, 3208, 3212. 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.

4) [4] Rappresentare in double precision IEEE-754, il valore 2ε essendo ε il valore dell'epsilon di macchina (nota bene: 64 bit) opportunamente normalizzato secondo quanto previsto dallo standard.

5) [4] Indicare come deve essere modificata la macchina a stati finiti che rappresenta il processore, in modo che venga supportata una eccezione relativa ad una lettura ad un indirizzo dati non permesso (valore di 'cause' pari a 4). Indicare inoltre quali registri speciali vengono modificati.

6) [8] Sintetizzare una rete sequenziale utilizzando il modello di Mealy-Ritardato con un ingresso X su un bit e una uscita Z su due bit che funziona nel seguente modo: devono essere riconosciute le sequenze non interallacciate 1,1,0,0 e 0,1,0,1; l'uscita Z[1] va a 1 se si presente una delle due sequenze mentre Z[0] dice quale sequenza si e' presentata (Z[0]=1 se si presenta 1,1,0,0; Z[0]=0 altrimenti. Rappresentare la macchina a stati finiti per tale rete di Mealy-Ritardato, 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. 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 con ausilio del simulatore oppure su carta e salvare una copia dell'output e del programma nella cartella ARCAL1161104.

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

COMPITO di ARCHITETTURA DEI CALCOLATORI del 04-11-2016

SOLUZIONE (TRACCIA aggiornata versione 1.2)

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	mtc1	$1,$f2	$f2=$1	Data from gen.reg. $1 to C1 reg. $f2 (no conversion)
mfc1	mfc1	$f1,$1	$1=$f2	Data from gen.reg. to C1 reg. (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]	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<=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; @(posedge clock); X<=0;
@(posedge clock); X<=1; @(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<=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
```

COMPITO di ARCHITETTURA DEI CALCOLATORI del 04-11-2016

SOLUZIONE (TRACCIA aggiornata versione 1.2)

ESERCIZIO 1

Una possibile implementazione e' la seguente:

```
atoi:
   # a0 = puntatore alla stringa da convertire in intero
   # v0 = risultato
   # nota: in caso di errore atoi da'risultato indeterminato
   addi $t0, $a0, 0      # copio il puntatore s in t0
   addi $v0, $zero, 0    # azzero v0 (risultato)
   addi $t3, $zero, 10   # costante 10 (usata successivamente)
   addi $t4, $zero, 0    # azzero t4 (segno positivo)
   addi $t5, $zero, 0x2B # codifica ASCII del segno '+'
   addi $t6, $zero, 0x2D # codifica ASCII del segno '-'
   lb   $t1, 0($t0)      # carica il char attualmente puntato
   beq  $t1, $t5, ok1    # segno piu' vai avanti
   bne  $t1, $t6, ok2    # altro char vai avanti
   addi $t4, $t4, 1      # registra segno negativo
ok1:
   addi $t0, $t0, 1      # punta carattere successivo
loop:
   lb   $t1, 0($t0)      # carica il char attualmente puntato
ok2:
   beq  $t1, $zero, done # test per fine stringa (char '\0')
   addi $t1, $t1, -48    # aggiusta il valore da ASCII '0' a valore 0
   slti $t2, $t1, 0      # il valore e' <? 0
   bne  $t2, $zero, fine # si', allora c'e' un errore ? fine
   slti $t2, $t1, 10     # il valore e' <? 10
   beq  $t2, $zero, fine # no, allora c'e' un errore ? fine
   mult $v0, $t3         # risultato *= 10
   mflo $v0              #
   add  $v0, $v0, $t1    # risultato += cifra
   addi $t0, $t0, 1      # punta carattere successivo
   j loop                # elabora tale carattere
done:
   beq  $t4, $0, fine    # check segno positivo
   nor  $v0, $v0, $0     # change sign - step1
   addi $v0, $v0, 1      # change sign - step2
fine:
   jr $ra                # return
```

ESERCIZIO 2

Il risultato dipende dal numero da convertire: supponiamo per semplicita' che la stringa fornita sia "1".
In tal caso:

N_CPU_ALJ=15, N_CPU_LS=2, N_CPU_B=7

TOT_CPI_ALJ= N_CPU_ALJ * 1=15, TOT_CPI_LS= N_CPU_LS *90=180, TOT_CPU_B=N_CPU_B*3=21

T_CPU=(TOT_CPI_ALJ+TOT_CPI_LS+ TOT_CPU_B)/FC=216/10E9=216 ns

ESERCIZIO 3

```
A = 4
B = 16
C = 128
RP = LRU
Thit = 1
Tpen = 99
File: c1161104.sh_001000.din
Read 21 references.
=== T    X   XM  XT  XS  XB  H [SET]:USAGE [SET]:MODIF  [SET]:TAG
=== R 3123  195  97   1   3  0 [1]:3,0,0,0 [1]:0,0,0,0  [1]:97,-,-,-
=== W 7456  466 233   0   0  0 [0]:3,0,0,0 [0]:0,0,0,0  [0]:233,-,-,-
=== R 4567  285 142   1   7  0 [1]:2,3,0,0 [1]:0,0,0,0  [1]:97,142,-,-
=== W 1190   74  37   0   6  0 [0]:2,3,0,0 [0]:0,0,0,0  [0]:233,37,-,-
=== R 7264  454 227   0   0  0 [0]:1,2,3,0 [0]:0,0,0,0  [0]:233,37,227,-
=== W 2789  174  87   0   5  0 [0]:0,1,2,3 [0]:0,0,0,0  [0]:233,37,227,87
=== R 2893  180  90   0  13  0 [0]:3,0,1,2 [0]:0,0,0,0  [0]:90,37,227,87   (out: XM=466 XT=233 XS=0 )
=== W 9088  568 284   0   0  0 [0]:2,3,0,1 [0]:0,0,0,0  [0]:90,284,227,87  (out: XM=74 XT=37 XS=0 )
=== R 2019  126  63   0   3  0 [0]:1,2,3,0 [0]:0,0,0,0  [0]:90,284,63,87   (out: XM=454 XT=227 XS=0 )
=== W 1290   80  40   0  10  0 [0]:0,1,2,3 [0]:0,0,0,0  [0]:90,284,63,40   (out: XM=174 XT=87 XS=0 )
=== R 2227  139  69   1   3  0 [1]:1,2,3,0 [1]:0,0,0,0  [1]:97,142,69,-
=== W 3902  243 121   1  14  0 [1]:0,1,2,3 [1]:0,0,0,0  [1]:97,142,69,121
=== R 8903  556 278   0   7  0 [0]:3,0,1,2 [0]:0,0,0,0  [0]:278,284,63,40  (out: XM=180 XT=90 XS=0 )
=== W 8890  555 277   1  10  0 [1]:3,0,1,2 [1]:0,0,0,0  [1]:277,142,69,121 (out: XM=195 XT=97 XS=1 )
=== R 3160  197  98   1   8  0 [1]:2,3,0,1 [1]:0,0,0,0  [1]:277,98,69,121  (out: XM=285 XT=142 XS=1 )
=== W 3189  199  99   1   5  0 [1]:1,2,3,0 [1]:0,0,0,0  [1]:277,98,99,121  (out: XM=139 XT=69 XS=1 )
=== R 3197  199  99   1  13  1 [1]:1,2,3,0 [1]:0,0,0,0  [1]:277,98,99,121
=== W 3201  200 100   0   1  0 [0]:2,3,0,1 [0]:0,0,0,0  [0]:278,100,63,40  (out: XM=568 XT=284 XS=0 )
=== R 3207  200 100   0   7  1 [0]:2,3,0,1 [0]:0,0,0,0  [0]:278,100,63,40
=== W 3208  200 100   0   8  1 [0]:2,3,0,1 [0]:0,1,0,0  [0]:278,100,63,40
=== R 3212  200 100   0  12  1 [0]:2,3,0,1 [0]:0,1,0,0  [0]:278,100,63,40
-------------------------------------
P1 Nmiss=17   Nhit=4  Nref=21  mrate=0.809524   AMAT=81.1429
```

COMPITO di ARCHITETTURA DEI CALCOLATORI del 04-11-2016

SOLUZIONE (TRACCIA aggiornata versione 1.2)

ESERCIZIO 4

Per definizione l'ε di macchina e' "la differenza fra 1 e '1+' essendo '1+' il piu' piccolo numero rappresentabile maggiore di 1". Ovvero e' quello numero la cui mantissa ha l'ultimo bit della mantissa a 1 e la parte a sinistra della virgola pari a zero. Per normalizzare tale numero si deve quindi traslare verso sinistra tale bit meno significativo di un numero pari al numero di cifre m della mantissa e sottrarre quindi m all'esponente. In altri termini l'ε di macchina: 2^{-m}. Nel caso di questo esercizio, essendo m=52 per la rappresentazione in doppia precisione: $2\varepsilon=2^{-(m-1)}=2^{-51}$. Dunque: Segno S= 0, Esponente E=polarizzazione+esponente=1023-51=972=011'1100'1100 e Mantissa M=0…0 (52 zeri). Ovvero: 0011 1100 1100 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000.

ESERCIZIO 5

La macchina a stati finiti che rappresenta il processore viene modificata con la parte in rosso di figura1 (si e' scelto uno stato nuovo pari a $(1101)_2$=13). Notare in particolare il punto in cui l'eccezione viene rilevata.

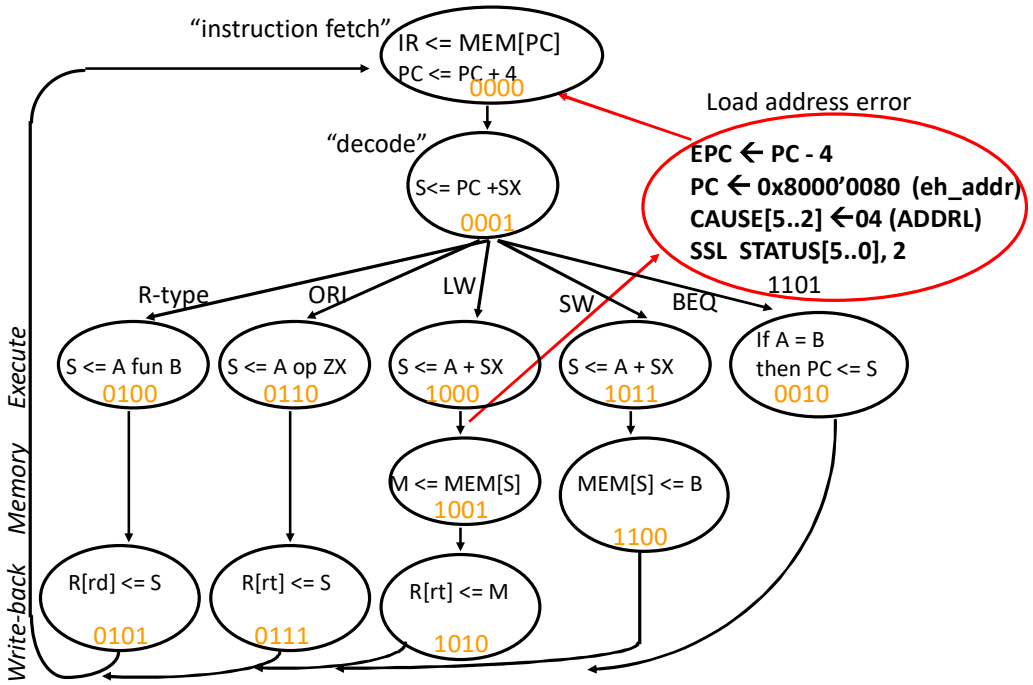


Figura 1

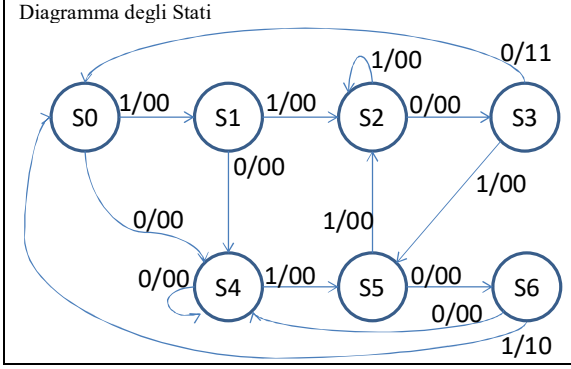
COMPITO di ARCHITETTURA DEI CALCOLATORI del 04-11-2016

SOLUZIONE (TRACCIA aggiornata versione 1.2)

ESERCIZIO 6

In corrispondenza del pattern $X_{t-3},X_{t-2},X_{t-1},X_t$ = 1,1,0,0 ottengo → Z_{t+1} = 11; (ricordare che e' richiesto Mealy ritardato). In corrispondenza del pattern $X_{t-3},X_{t-2},X_{t-1},X_t$ = 0,1,0,1 ottengo → Z_{t+1} = 10.

Diagramma degli Stati



Schema di riferimento per Mealy Ritardato

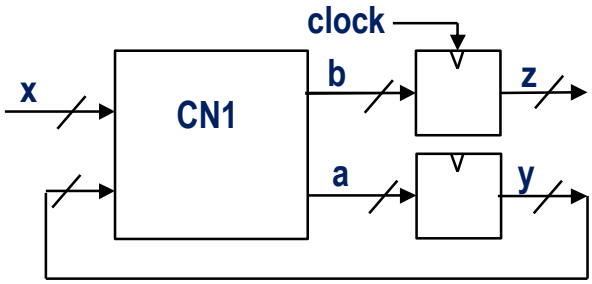


Tabella delle Transizioni (Tabella degli Stati)

TABELLA DELLE TRANSIZIONI

STATO ATTUALE \ x	0	1
S0	S4/00	S1/00
S1	S4/00	S2/00
S2	S3/00	S2/00
S3	S0/11	S5/00
S4	S4/00	S5/00
S5	S6/00	S2/00
S6	S4/00	S0/10

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
X	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	S4/00	S1/00	S5/00	S4/00
01	S4/00	S2/00	S2/00	S6/00
11	S3/00	S2/00	S0/10	S4/00
10	S0/11	S5/00	X/XX	X/XX

STATO SUCCES-SIVO/USCITA

OVVERO

y_1y_0 \ y_2x	00	01	11	10
00	100/00	001/00	101/00	100/00
01	100/00	011/00	011/00	111/00
11	010/00	011/00	000/10	100/00
10	000/11	101/00	XXX/XX	XXX/XX

a2,a1,a0,z1,z0

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

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

a_2

a2=/y1/x+y2/x+y2/y1/y0+y1/y0x

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

a_1

a1=/y2y1y0+y2/y1y0+/y1y0x

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

a_0

a0=/y2x+/y1x+y2/y1y0

Sintesi della variable 'b' (uscita successiva o ingresso dell'OUTPUT REGISTER -- OUTR):

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

b_1

b1=y2y1x+y1/y0/x

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

b_0

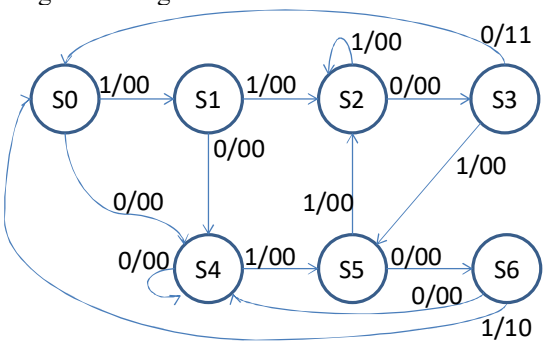
b0=y1/y0/x

COMPITO di ARCHITETTURA DEI CALCOLATORI del 04-11-2016

SOLUZIONE (TRACCIA aggiornata versione 1.2)

ESERCIZIO 7

Diagramma degli stati:



Codice Verilog del modulo da realizzare (possibile soluzione con Mealy-Ritardato):

Implementazione basta sul template di Mealy-Ritardato	Implementazione basata sulle equazioni booleane

```
module XXX(x,z,clock,reset_);
input        clock,reset_,x;
output [1:0] z;
reg    [1:0] OUTR;
reg    [2:0] STAR;
parameter S0='B000,S1='B001,S2='B010,S3='B011,
   S4='B100,S5='B101,S6='B110;
always @(reset_==0) #1 begin STAR<=S0; OUTR<=0; end
assign z=OUTR;

always @(posedge clock) if(reset_==1) #3
casex(STAR)
   S0:begin STAR<=(x==0)?S4:S1; OUTR<=0; end
   S1:begin STAR<=(x==0)?S4:S2; OUTR<=0; end
   S2:begin STAR<=(x==0)?S3:S2; OUTR<=0; end
   S3:begin STAR<=(x==0)?S0:S5; OUTR<=(x==0)?'B11:'B00;
end
   S4:begin STAR<=(x==0)?S4:S5; OUTR<=0; end
   S5:begin STAR<=(x==0)?S6:S2; OUTR<=0; end
   S6:begin STAR<=(x==0)?S4:S0; OUTR<=(x==1)?'B10:'B00;
end
endcase
endmodule
```

```
module XXX(x,z,clock,reset_);
input        clock,reset_,x;
output [1:0] z;
wire   [2:0] y;
wire   [1:0] b;
wire   [2:0] a;
reg    [1:0] OUTR;
reg    [2:0] STAR;
always @(reset_==0) #1 begin STAR<=0; OUTR<=0; end
assign z=OUTR; assign y=STAR;

assign a[2]=((~y[1])&(~x))|(y[2]&(~x))|(y[2]&(~y[1])&(~y[0]))
       |(y[1]&(~y[0])&x);
assign
a[1]=((~y[2])&y[1]&y[0])|(y[2]&(~y[1])&y[0])|((~y[1])&y[0]&x);
assign a[0]=((~y[2])&x)|((~y[1])&x)|(y[2]&(~y[1])&y[0]);

assign b[1]=(y[2]&y[1]&x)|(y[1]&(~y[0])&(~x));
assign b[0]=y[1]&(~y[0])&(~x);

always @(posedge clock) if(reset_==1) #3
begin OUTR<=b; STAR<=a; end

endmodule
```

Diagramma di Temporizzazione: (template)

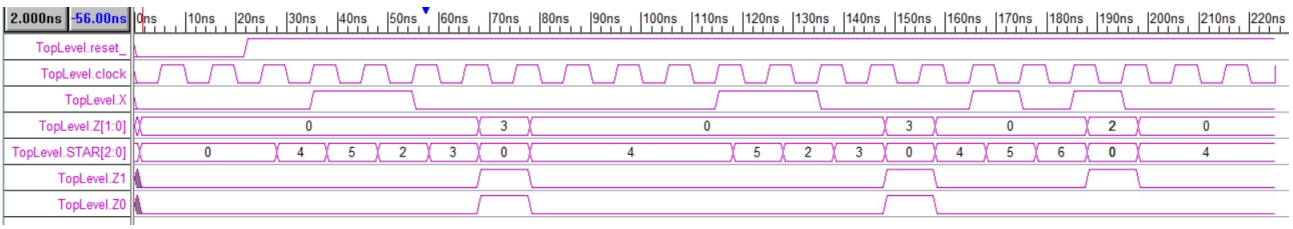


Diagramma di Temporizzazione: (equazioni booleane – nota 2→S3,3→S2,7→S6)