COMPITO di ARCHITETTURA DEI CALCOLATORI del 29-09-2017

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] Scrivere in assembly MIPS l'implementazione della funzione char *itoa(int n) che trasforma il numero naturale n (ovvero un intero >=0) in una stringa contenente i caratteri ASCII, il cui puntatore e' restituito come parametro di ritorno che rappresenta in lo stesso numero (es. Il naturale 123 dovra' diventare "123"). La funzione dovra' anche provvedere all'allocazione della memoria dinamica per memorizzare la stringa di uscita, inoltre dovra' essere realizzato un semplice programma main che legga da tastiera l'intero di ingresso e stampi la stringa di uscita chiamando la funzione "itoa" (promemoria: i caratteri ASCII delle cifre sono "0"->0x30, "1"-> 0x31, ...).

2) [8] Si consideri una cache di dimensione 96B e a 3 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' LRU. Il processore effettua i seguenti accessi in cache, ad indirizzi al byte: 623, 339, 327, 379, 778, 139, 333, 754, 725, 354, 322, 354, 739, 1, 26, 754, 324, 354, 729, 354, 328, 354. 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.

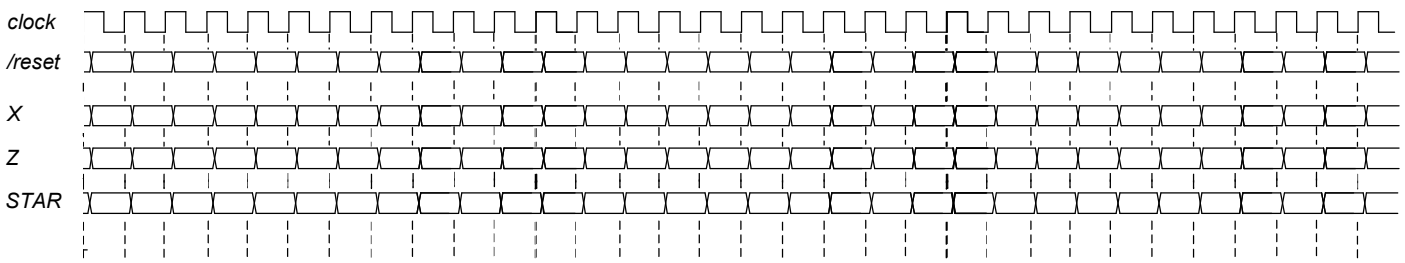
3) [6] In un processore a 64 bit con memoria virtuale supportata da paginazione a 3 livelli con dimensione della pagina di 4KiB, vengono riservati 12 bit per indirizzare il primo livello, 20 bit per indirizzare il secondo livello e altri 20 bit per indirizzare il terzo livello. Rappresentare uno schema architetturale che implementi tale meccanismo di paginazione e istanziare numericamente il valore di un indirizzo virtuale, un indirizzo fisico corrispondente e valori realistici contenuti nelle celle delle pagine relative al percorso che collega l'indirizzo virtuale a quello fisico.

4) [4] Spiegare la legge di Amdhal e tracciarne il grafico di S (speedup) al variare di p (porzione di parte migliorabile) e .con a (fattore di accelerazione della parte migliorabile) pari a 1, 10, 100.

5) [4] Spiegare la legge di Amdhal e tracciarne il grafico di S (speedup) al variare di a (fattore di accelerazione della parte migliorabile) e con di p (porzione di parte migliorabile) pari a 0.1, 0.5, 0.9, 1.

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 riconosca le sequenze interallacciate 1,1,0,1. 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 29-09-2017 **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 29-09-2017

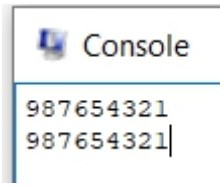
SOLUZIONE

ESERCIZIO 1

```
        .data
nul:    .asciiz ""

        .text
        .globl main

itoa:   add $t0, $0, $a0
      add $t1, $0, $0
      add $t2, $0, 10
div_start:
      slti $t3, $t0, 10
      bne $t3, $0, div_end

      div $t0, $t2
      mfhi $t4
      mflo $t0
      addi $t4, $t4, 0x30
      addi $t1, $t1, 1
      addi $sp, $sp, -1
      sb $t4, 0($sp)

      j div_start
div_end:
      addi $t0, $t0, 0x30
      addi $t1, $t1, 1
      addi $sp, $sp, -1
      sb $t0, 0($sp)
```

```
      add $a0, $0, $t1
      addi $a0, $a0, 1
      ori $v0, $0, 9
      syscall

      add $t5, $0, $v0
sb_start:
      lb $t6, 0($sp)
      addi $sp, $sp, 1
      addi $t1, $t1, -1
      sb $t6, 0($t5)
      addi $t5, $t5, 1
      beq $t1, $0, sb_end
      j sb_start
sb_end:
      la $t7, nul
      lb $t8, 0($t7)
      sb $t8, 0($t5)
      jr $ra

main:   ori $v0, $0, 5
        syscall

      add $a0, $0, $v0

      jal itoa
```

```
      add $a0, $0, $v0
      ori $v0, $0, 4
        syscall

      ori $v0, $0, 10
        syscall
```



ESERCIZIO 2

Sia X il generico riferimento, A=associativita', B=dimensione del blocco, C=capacita' della cache.

Si ricava S=C/B/A=# di set della cache=96/8/3=4, XM=X/B, XS=XM%S, XT=XM/S:

```
A=3, B=8, C=96, RP=LRU, Thit=4, Tpen=40, 22 references:
=== T    X   XM  XT  XS  XB  H [SET]:USAGE [SET]:MODIF  [SET]:TAG
=== R  623   77  19   1   7  0 [1]:2,0,0   [1]:0,0,0  [1]:19,-,-
=== W  339   42  10   2   3  0 [2]:2,0,0   [2]:0,0,0  [2]:10,-,-
=== R  327   40  10   0   7  0 [0]:2,0,0   [0]:0,0,0  [0]:10,-,-
=== W  379   47  11   3   3  0 [3]:2,0,0   [3]:0,0,0  [3]:11,-,-
=== R  778   97  24   1   2  0 [1]:1,2,0   [1]:0,0,0  [1]:19,24,-
=== W  139   17   4   1   3  0 [1]:0,1,2   [1]:0,0,0  [1]:19,24,4
=== R  333   41  10   1   5  0 [1]:2,0,1   [1]:0,0,0  [1]:10,24,4 (out: XM=77 XT=19 XS=1 )
=== W  754   94  23   2   2  0 [2]:1,2,0   [2]:0,0,0  [2]:10,23,-
=== R  725   90  22   2   5  0 [2]:0,1,2   [2]:0,0,0  [2]:10,23,22
=== W  354   44  11   0   2  0 [0]:1,2,0   [0]:0,0,0  [0]:10,11,-
=== R  322   40  10   0   2  1 [0]:2,1,0   [0]:0,0,0  [0]:10,11,-
=== W  354   44  11   0   2  1 [0]:1,2,0   [0]:0,1,0  [0]:10,11,-
=== R  739   92  23   0   3  0 [0]:0,1,2   [0]:0,1,0  [0]:10,11,23
=== W    1    0   0   0   1  0 [0]:2,0,1   [0]:0,1,0  [0]:0,11,23 (out: XM=40 XT=10 XS=0 )
=== R   26    3   0   3   2  0 [3]:1,2,0   [3]:0,0,0  [3]:11,0,-
=== W  754   94  23   2   2  1 [2]:0,2,1   [2]:0,1,0  [2]:10,23,22
=== R  324   40  10   0   4  0 [0]:1,2,0   [0]:0,0,0  [0]:0,10,23 (out: XM=44 XT=11 XS=0 )
=== W  354   44  11   0   2  0 [0]:0,1,2   [0]:0,0,0  [0]:0,10,11 (out: XM=92 XT=23 XS=0 )
=== R  729   91  22   3   1  0 [3]:0,1,2   [3]:0,0,0  [3]:11,0,22
=== W  354   44  11   0   2  1 [0]:0,1,2   [0]:0,0,1  [0]:0,10,11
=== R  328   41  10   1   0  1 [1]:2,0,1   [1]:0,0,0  [1]:10,24,4
=== W  354   44  11   0   2  1 [0]:0,1,2   [0]:0,0,1  [0]:0,10,11
-------------------------------------
P1 Nmiss=16   Nhit=6  Nref=22  mrate=0.727273   AMAT=th+mrate*tpen=33.0909
```

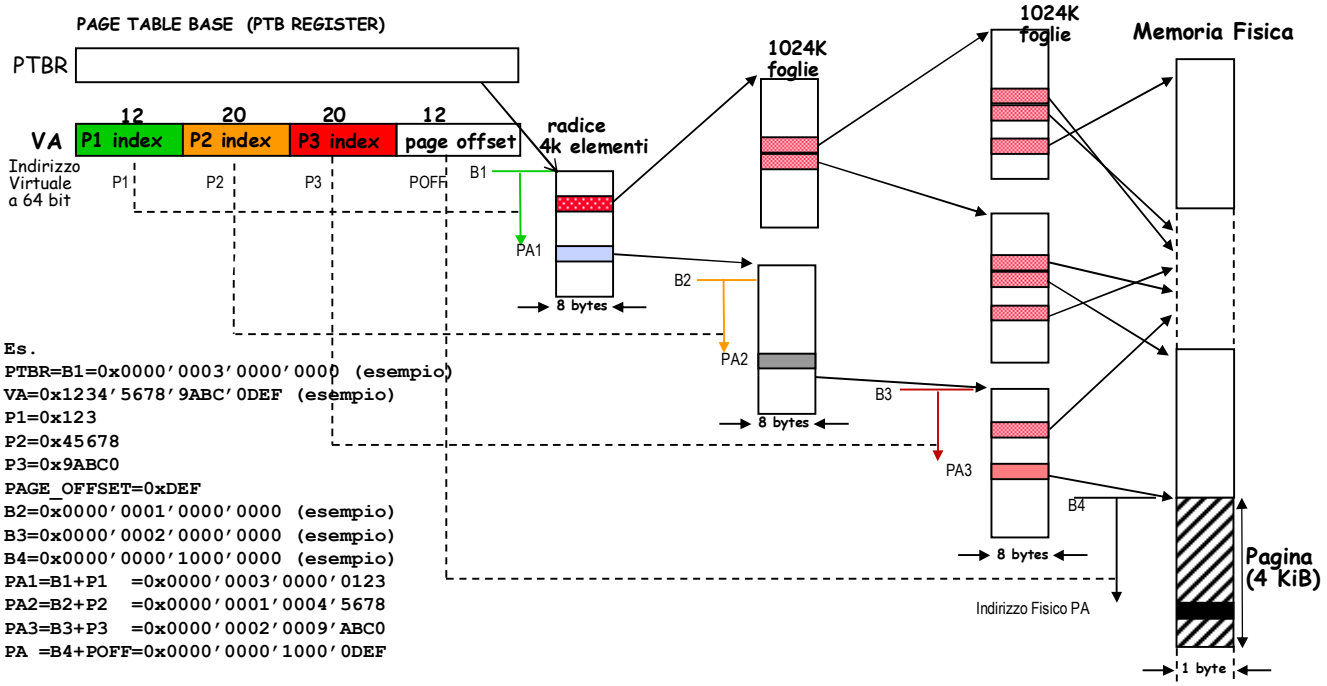
LISTA BLOCCHI USCENTI:

CONTENTUTI dei 4 SET al termine:

COMPITO di ARCHITETTURA DEI CALCOLATORI del 29-09-2017

SOLUZIONE

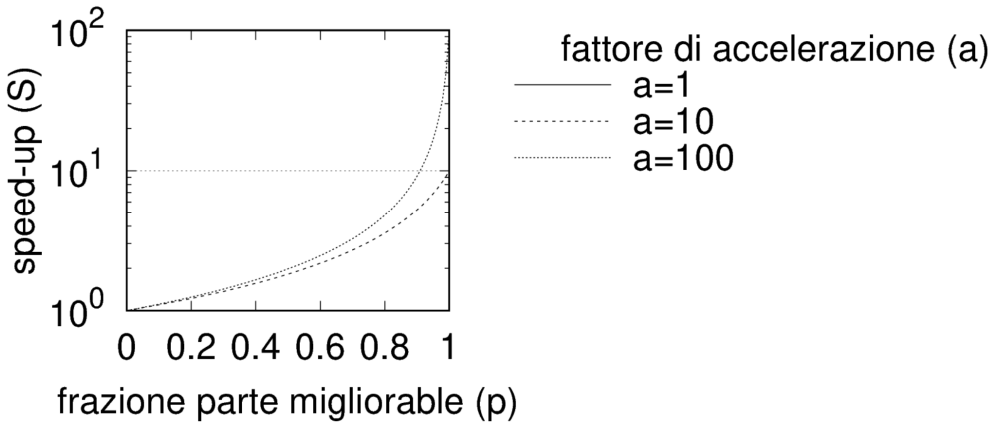
ESERCIZIO 3



ESERCIZIO 4.

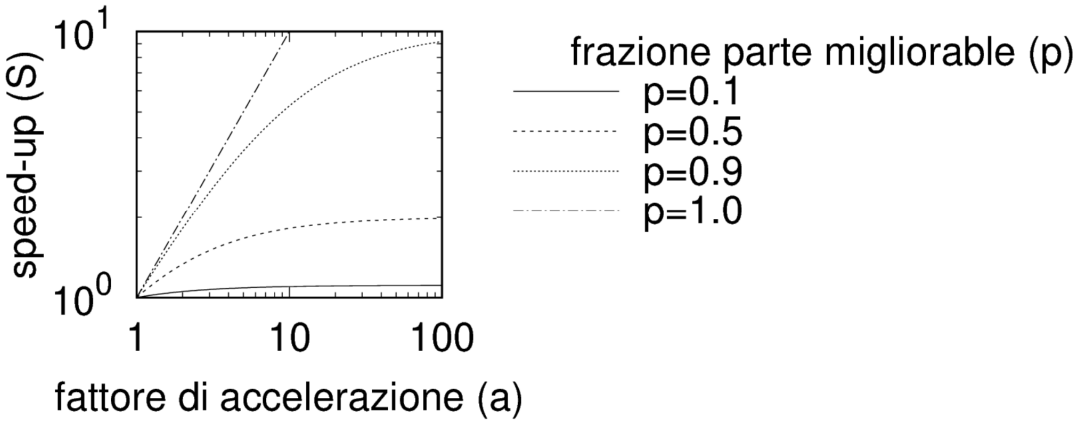
La legge di Amdhal mette in relazione lo speedup di una parte del sistema con lo speedup dell'intero sistema.
S(p,a)=1/((1-p)+p/a)

Per a=1, lo speed-up S e' costante e pari a 1 (ovviamente).



ESERCIZIO 5.

La legge di Amdhal mette in relazione lo speedup di una parte del sistema con lo speedup dell'intero sistema.
S(p,a)=1/((1-p)+p/a)

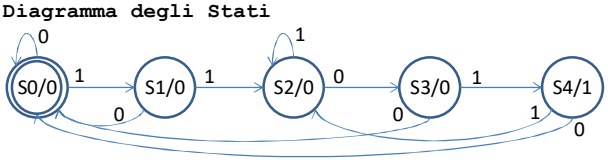


COMPITO di ARCHITETTURA DEI CALCOLATORI del 29-09-2017

SOLUZIONE

ESERCIZIO 6

Diagramma degli Stati



Schema di riferimento per Moore

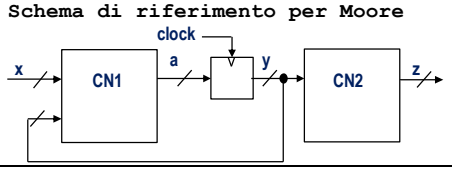


Tabella delle Transizioni (Tabella degli Stati)

TABELLA DELLE TRANSIZIONI

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

(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	S0	S2
01	S0	S2	X	X
11	S3	S2	X	X
10	S0	S4	X	X

STATO SUCCESSIVO / USCITA

OVVERO

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

a2,a1,a0

(segue)

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

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

a_2

a2=y1/y0x

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

a_1

a1=y1y0+y2/x+y0x

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

a_0

a0=/y2/y1x+y2/x+y0x

Sintesi della variable 'z' (uscita):

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

z

z=y2

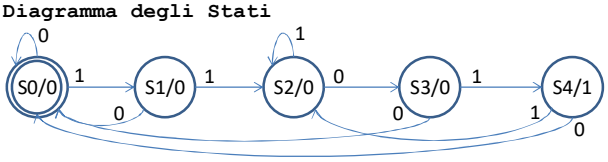
COMPITO di ARCHITETTURA DEI CALCOLATORI del 29-09-2017

SOLUZIONE

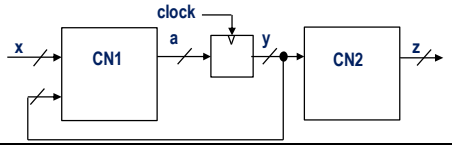
ESERCIZIO 7

E' utile rappresenttare il diagramma a stati del riconoscitore di sequenza richiesto:

Diagramma degli Stati



Schema di riferimento per Moore



Codice VERILOG:

```
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 [2:0] 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<=1;
    @(posedge clock); X<=1;@(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<=0;@(posedge clock); X<=1;@(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<=0;
      $finish;
    end
  XXX Xxx(X,Z,clock,reset_);
endmodule

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

Diagramma temporale: