COMPITO di ARCHITETTURA DEI CALCOLATORI del 12-07-2018

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 FOGLI PERSONALI, NO TELEFONI, SMARTPHONE, ETC

SVOLGIMENTO DELLA PROVA:
☐ PER GLI STUDENTI DI "ARCHITETTURA DEI CALCOLATORI – A.A. 2015/16, 16/17, 17/18": es. N.1+2+3+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) [19/38] Trovare il codice assembly MIPS corrispondente al seguente programma **(usando solo e unicamente istruzioni della tabella sottostante e rispettando le convenzioni di utilizzazione dei registri dell'assembly MIPS** riportate qua sotto per riferimento). Nota: la funzione "fabs" puo' essere mappata direttamente sull'istruzione "abs.s".

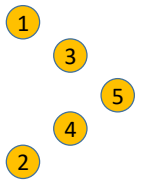
```
int idamax(int n, float *dx, int incx) {
    float dmax;
    int i, ix, itemp;

    if (n < 1) return(-1);
    if (n == 1) return(0);
    if (incx != 1) {
        ix = 1;
        dmax = fabs(dx[0]);
        ix = ix + incx;
        for (i = 1; i < n; i++) {
            if(dmax < fabs(dx[ix])) {
                itemp = i;
                dmax = fabs(dx[ix]);
             }
            ix = ix + incx;
        }
    } else {
        itemp = 0;
        dmax = fabs(dx[0]);
        for (i = 1; i < n; i++) {
            if(dmax < fabs(dx[i])) {
                itemp = i;
                dmax = fabs(dx[i]);
            }
        }
    }
    return (itemp);
}

int main() {
    float a[100];
    int l, k, n=100, lda=10;

    for (k=0; k<n; ++k) a[k] = (float)((k*k*k)%100);
    k=4;
    l = idamax(n-lda*k-k,&a[lda*k+k],1) + k;
    print_int(l);
    exit;
}
```

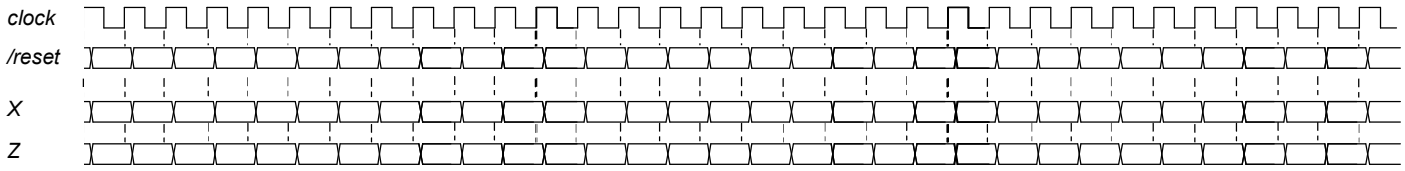
2) [7/38] Si consideri una cache di dimensione 96B e a 3 vie di tipo write-back/write-non-allocate. La dimensione del blocco e' 8 byte, il tempo di accesso alla cache e' 4 ns e la penalita' in caso di miss e' pari a 40 ns, la politica di rimpiazzamento e' LRU. Il processore effettua i seguenti accessi in cache, ad indirizzi al byte: 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) [4/38] Si supponga che un programma richieda l'esecuzione di $50x10^6$ istruzioni FP, $110x10^6$ istruzioni INT, $80x10^6$ istruzioni di lettura/scrittura e $16x10^6$ istruzioni di salto condizionato. Il CPI per ciascun tipo di istruzione e' rispettivamente di 1, 1, 4 e 2. Si supponga che il processore abbia una frequenza di clock di 2 GHz. A) di quanto occorre aumentare il CPI delle istruzioni FP se si vuole che il programma venga eseguito al doppio della velocita'? B) di quanto occorre aumentare il CPI delle istruzioni lettura/scrittura se si vuole che il programma venga eseguito al doppio della velocita'?

4) Non assegnato
5) Non assegnato
6) Non assegnato

7) [8/38] **Realizzare** in Verilog una rete sequenziale secondo il modello di Mealy che accenda 5 led di una "freccia a destra" nel modo seguente: durante il primo ciclo tutte i led sono spenti; nel ciclo successivo si accendono i led 1 e 2, nel secondo ciclo si accendono i led 1,2,3,4; nel ciclo successivo tutti i 5 led sono accesi; poi la sequenza si ripete, cioe' al ciclo successivo led tutti spenti, poi 1,2, poi 1,2,3,4 e cosi'via. L'ingresso X su un bit e'un interruttore generale che indica con X=1 che i led (governati dall'uscita Z su 3 bit) si accendono secondo la sequenza descritta, se X=0 tutti i led devono stare spenti. **Tracciare il diagramma di temporizzazione** come verifica della correttezza dell'unità riportando i segnali clock, /reset, uscita Z. Nota: si può 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.





Testbench:

```
`timescale 100ms/1ms
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; initial begin x=0; #40 x=1; end
wire[1:0] STAR = Xxx.STAR;
wire[2:0] z=Xxx.z;
XXX Xxx(x,z,clock,reset_);
endmodule
```

COMPITO di ARCHITETTURA DEI CALCOLATORI del 12-07-2018

Instructions

Opcode+Funct (hexadecimal)	Instruction	Example	Meaning	Comments
00+20/00+21	**add**	`add/addu $1,$2,$3`	$1 = $2 + $3	(signed/unsigned) 3 operands; exception possible
00+22/00+23	**subtract**	`sub/subu $1,$2,$3`	$1 = $2 - $3	(signed/unsigned) 3 operands; exception possible
08/09	**add immediate**	`addi/addiu $1,$2,100`	$1 = $2 + 100	(signed/unsigned) + constant ; exception possible
00+18/00+19	**multiplication**	`mult/multu $1, $2`	Hi,Lo= $1 x $2	(signed/unsigned) 64-bit Product ; result in Hi,Lo
00+1A/00+1B	**division**	`div/divu $1, $2`	Hi= $1 % $2, Lo = $1 / $2	(signed/unsigned) division
00+10/00+12	**move from Hi / move from Lo**	`mfhi/mflo $1`	$1 = Hi ($1 = Lo)	Create copy of Hi (Create a copy of Lo)
00+2A/00+2B	**set on less than**	`slt/sltu $1,$2,$3`	if ($2 < $3) $1 = 1; else $1 = 0	(signed/unsigned) compare $2 and $3 (less than)
0A/0B	**set on less than immediate**	`slti/sltiu $1,$2,100`	if ($2 < 100) $1 = 1; else $1 = 0	(signed/unsigned) compare $2 and constant (less than)
00+24/25/26/27	**and / or / xor / nor**	`and/or/xor/nor $1,$2,$3`	$1=$2&$3 / $2|$3 / $2^$3 / !($2|$3)	3 register operands; Logical AND/OR/XOR/NOR
0C/0D/0E	**and /or / xor immediate**	`andi/ori/xori $1,$2,100`	$1 = $2 & 100 / $2 | 100 / $2 ^100	Logical AND/OR/XOR register, constant
00+00	**shift left logical**	`sll $1,$2,10`	$1 = $2 << 10	Shift left by constant
00+02/00+03	**shift right (l=logical,a=arithmetic)**	`srl/sra $1,$2,10`	$1 = $2 >> 10	Shift right by constant (for arithmetic: sign is preserved)
23/20	**load word / load byte**	`lw/lb $1,100($2)`	$1 = Memory[$2+100]	Data from memory to register
24	**load byte unsigned**	`lbu $1,100($2)`	$1 = Memory[$2+100]	Data from mem. To reg.; no sign extension
2B/28	**store word / store byte**	`sw/sb $1,100($2)`	Memory[$2+100] = $1	Data from register to memory
0F	**load upper immediate**	`lui $1,0x1234`	$1=0x1234'0000	load most significant 16 bits
PSEUDOINSTRUCTION	**load address**	`la $1,var`	$1 = &var	Load address of var (lui $1,H16(&var);ori $1, L16(&var)) H16/L16=high/low 16 bits of &var
02	**jump**	`j 10000`	go to 10000	Jump to target address
00+08	**jump register**	`jr $31`	go to $31	For switch, procedure return
03	**jump and link**	`jal 10000`	$31 = PC + 4;go to 10000	For procedure call
04	**branch on equal**	`beq $1,$2,100`	if ($1 = = $2) go to PC+4+100	Equal test; PC relative branch
05	**branch on not equal**	`bne $1,$2,100`	if ($1 != $2) go to PC+4+100	Not equal test; PC relative
00+0C	**syscall**	`syscall`	call OS service $v0	See table of system calls below
10+10,rs=10	**rfe**	`rfe`	shift right (k,e) bits in STATUS reg	Exit Kernel Mode, Enable Interrupts
PSEUDOINSTRUCTION	**branch unconditional**	`b 100`	go to PC+4+100	PC relative branch (e.g., beq $0,$0,100)
PSEUDOINSTRUCTION	**no operation**	`nop`	do nothing	Do nothing (e.g. sll $0,$0,0)
30	**load-linked**	`ll $1,100($2)`	$1=Memory[$2+100]	Read and start to monitor the given memory location
38	**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)
11+00 fmt=10/11	**add.s / add.d**	`add.x $f0,$f2,$f4`	$f0=$f2+$f4	Single and double precision add
11+01 fmt=10/11	**sub.s / sub.d**	`sub.x $f0,$f2,$f4`	$f0=$f2-$f4	Single and double precision subtraction
11+02 fmt=10/11	**mul.s / mul.d**	`mul.x $f0,$f2,$f4`	$f0=$f2*$f4	Single and double precision multiplication
11+03 fmt=10/11	**div.s / div.d**	`div.x $f0,$f2,$f4`	$f0=$f2/$f4	Single and double precision division
11+05 fmt=10/11	**abs.s / abs.d**	`abs.x $f0,$f2`	$f0=ABS($f2)	Single and double precision absolute value
11+06 fmt=10/11	**mov.s / mov.d**	`mov.x $f0,$f2`	$f0←$f2	Single and double precision move
11+07 fmt=10/11	**neg.s / neg.d**	`neg.x $f0,$f2`	$f0= – ($f2)	Single and double precision opposite value
11+3C(31,32,3D,3E,3F) fmt=10/11	**c.lt.s / c.lt.d (ne,eq,gt,le,ge)**	`c.lt.x $f0,$f2`	Temp=($f0<$f2)	Single and double: compare $f0 and $f2 <,=,!=,>,<=>=
11+00 fmt=4/0	**move to/from coprocessor 1**	`mtc1/mfc1 $1,$f2`	$f2=$1 / $1=$f2	Move $1 to/from C1reg. $f2 (no conversion)
10+00 fmt=4/0	**move to/from coprocessor 0**	`mtc0/mfc0 $1,$2`	$c2=$1 / $1=$c2	Move $1 to/from C0 reg. $f2 (no conversion)
11+00 fmt=6/2	**move to/from control reg of cop.1**	`ctc1/cfc1 $1,$cf2`	$cf2=$1 / $1=$cf2	Move $1 to/from C1-CONTROL register
11 fmt=8,ft=1/0	**branch on true/false**	`bc1t/bc1f label`	If (Temp = = true/false) go to label	Temp is 'Condition-Code'
31/39	**load/store floating point (32bit)**	`lwc1/swc1 $f0,0($1)`	$f0←Memory[$1] / Memory[$1]←$f0	Data from FP (C1) register to memory
11+21,fmt=10/11+22,fmt=11	**convert from/to single to/from double**	`cvt.d.s/cvt s.d $f0,$f2`	$f0=(double)$f2/$f0=(single)$f2	Type conversion
11+24,fmt=11/11+20	**convert from/to single to/from integer**	`cvt.w.s/cvt s.w $f1,$f0`	$f1=(int)$f0 / $f0=(single)$f2	Type conversion

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

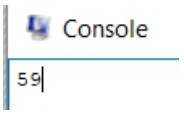
Service Name	Service Num. ($v0)	INPUT Arguments	OUTPUT Arguments
print_int	1	$a0=integer to print	---
print_float	2	$f12=float to print	---
print_double	3	($f12,$f13)=double to print	---
print_string	4	$a0=address of ASCIIZ string to print	---
read_int	5	---	$v0=integer
read_float	6	---	$f0=float
read_double	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 12-07-2018

SOLUZIONE

ESERCIZIO 1

```
.text
.text
.globl main

idamax:
#______NO_CALL_FRAME______
# dmax in f4
# i in t0, ix in t1, itemp in t2

   addi $t4,$0,1       #t4=1
   addi $v0,$0,-1      #retval=-1
   slti $t5,$a0,1      #n<?1
   bne  $t5,$0,idamax_ret
   add  $v0,$0,$0      #retval=0
   beq  $a0,$t4,idamax_ret #n==?1

   beq  $a2,$t4,idamax_else1

   addi $t1,$0,1       #ix=1
   lwc1 $f5,0($a1)     #dx[0]
   abs.s $f4,$f5       #dmax=fabs()
   add  $t1,$t1,$a2
   addi $t0,$0,1       #i=1
idamax_forini1:
   slt  $t5,$t0,$a0    #i<?n
   beq  $t5,$0,idamax_forend1

   sll  $t5,$t1,2      #ix*4
   add  $t5,$t5,$a1    #&dx[ix]
   lwc1 $f5,0($t5)     #dx[ix]
   abs.s $f6,$f5       #=fabs()
   c.lt.s $f4,$f6      #dmax<?()
   bc1f idamax_if1end
   add $t2,$0,$t0      #itemp=i
   mov.s $f4,$f6       #dmax=()
idamax_if1end:
   add  $t1,$t1,$a2    #ix=ix+incx
   addi $t0,$t0,1      #i++
   j idamax_forini1
idamax_forend1:
   j idamax_ifend1

idamax_else1:
   addi $t2,$0,1       #itemp=1
   lwc1 $f5,0($a1)     #dx[0]
   abs.s $f4,$f5       #dmax=fabs()
   add  $t1,$t1,$a2
   addi $t0,$0,1       #i=1
idamax_forini2:
   slt  $t5,$t0,$a0    #i<?n
   beq  $t5,$0,idamax_forend2

   sll  $t5,$t0,2      #i*4
   add  $t5,$t5,$a1    #&dx[i]
   lwc1 $f5,0($t5)     #dx[i]
   abs.s $f6,$f5       #=fabs()
   c.lt.s $f4,$f6      #dmax<?()
   bc1f idamax_if2end
   add $t2,$0,$t0      #itemp=i
   mov.s $f4,$f6       #dmax=()
idamax_if2end:
   addi $t0,$t0,1      #i++
   j idamax_forini2
idamax_forend2:

idamax_ifend1:
   add $v0,$0,$t2      #v0=itemp

idamax_ret:
   jal $ra


main:
#______CALL_FRAME______
# 100 float:       400B
#______Totale      400B
   addi $sp,$sp,-400
   add  $t9,$sp,$0     #&a
   addi $t0, $0, 100   #n=100
   addi $t1, $0, 10    #lda=10
#l in t2, k in t3

   add  $t3, $0, $0    #k=0
main_forini:
   slt  $t5,$t3,$t0    #k<?n
   beq  $t5,$0,main_forend

   mult $t3, $t3       #k*k
   mflo $t5
   mult $t3, $t5
   mflo $t5            #k*k*k
   div  $t5,$t0        #()%n
   mfhi $t5

   mtc1 $t5,$f0
   cvt.s.w $f1,$f0     #(float)()

   sll  $t5,$t3,2      #k*4
   add  $t5,$t5,$t9    #&a[k]
   swc1 $f1,0($t5)     #a[k]=()

   addi $t3, $t3, 1    #++k
   j main_forini
main_forend:

   addi $t3,$0,4       #k=4
   mult $t1,$t3        #lda*k
   mflo $t5
   add  $t5,$t5,$t3    #lda*k+k
   sub  $a0,$t0,$t5    #a0=n-lda*k-k
   sll  $t5,$t5,2
   add  $a1,$t5,$t9    #a1=&a[lda*k+k]
   addi $a2,$0,1       #a2=1
   jal  idamax
   addi $a0,$v0,4      #a0=l=retval+k
   addi $v0,$0,1       #print_int
   syscall
   addi $v0,$0,10      #exit
   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 (20 references).
=== T    X   XM  XT  XS  XB  H [SET]:USAGE [SET]:MODIF  [SET]:TAG
=== R   55    6   1   2   7  0 [2]:2,0,0 [2]:0,0,0  [2]:1,-,-
=== W  173   21   5   1   5  0 [1]:2,0,0 [1]:0,0,0  [1]:5,-,-
=== R  115   14   3   2   3  0 [2]:1,2,0 [2]:0,0,0  [2]:1,3,-
=== W  119   14   3   2   7  1 [2]:1,2,0 [2]:0,1,0  [2]:1,3,-
=== R  222   27   6   3   6  0 [3]:2,0,0 [3]:0,0,0  [3]:6,-,-
=== W  947  118  29   2   3  0 [2]:0,1,2 [2]:0,1,0  [2]:1,3,29
=== R  618   77  19   1   2  0 [1]:1,2,0 [1]:0,0,0  [1]:5,19,-
=== W  449   56  14   0   1  0 [0]:2,0,0 [0]:0,0,0  [0]:14,-,-
=== R  534   66  16   2   6  0 [2]:2,0,1 [2]:0,1,0  [2]:16,3,29
=== W  748   93  23   1   4  0 [1]:0,1,2 [1]:0,0,0  [1]:5,19,23
=== R  877  109  27   1   5  0 [1]:2,0,1 [1]:0,0,0  [1]:27,19,23
=== W  919  114  28   2   7  0 [2]:1,2,0 [2]:0,0,0  [2]:16,28,29
=== R  283   35   8   3   3  0 [3]:1,2,0 [3]:0,0,0  [3]:6,8,-
=== W  143   17   4   1   7  0 [1]:1,2,0 [1]:0,0,0  [1]:27,4,23
=== R  591   73  18   1   7  0 [1]:0,1,2 [1]:0,0,0  [1]:27,4,18
=== W  644   80  20   0   4  0 [0]:1,2,0 [0]:0,0,0  [0]:14,20,-
=== R  770   96  24   0   2  0 [0]:0,1,2 [0]:0,0,0  [0]:14,20,24
=== W  845  105  26   1   5  0 [1]:2,0,1 [1]:0,0,0  [1]:26,4,18
=== R  961  120  30   0   1  0 [0]:2,0,1 [0]:0,0,0  [0]:30,20,24
=== W  194   24   6   0   2  0 [0]:1,2,0 [0]:0,0,0  [0]:30,6,24
---------------------------------------
P1 Nmiss=19  Nhit=1  Nref=20  mrate=0.950000  AMAT=42
```

LISTA BLOCCHI USCENTI:

```
(out: XM=6   XT=1   XS=2 )

(out: XM=21 XT=5   XS=1 )
(out: XM=14 XT=3   XS=2 )

(out: XM=77 XT=19 XS=1 )
(out: XM=93 XT=23 XS=1 )


(out: XM=109 XT=27 XS=1 )
(out: XM=56 XT=14 XS=0 )
(out: XM=80 XT=20 XS=0 )
```

CONTENTUTI dei 4 SET al termine:

COMPITO di ARCHITETTURA DEI CALCOLATORI del 12-07-2018

SOLUZIONE

ESERCIZIO 3

Dall'equazione fondamentale delle prestazioni:

$T_{CPU} = {C_{CPU}}/{f_C}$ dove $C_{CPU} = \sum_{k=FP,INT,LS,B} CPI_K \cdot N_{INST,K}$

→ $C_{CPU} = 50 \cdot 1 \cdot 10^6 + 110 \cdot 1 \cdot 10^6 + 80 \cdot 4 \cdot 10^6 + 16 \cdot 2 \cdot 10^6 = 512 \cdot 10^6$

Dovendo dimezzare il tempo di esecuzione:

$T'_{CPU} = {T_{CPU}}/{2}$ → $C'_{CPU} = {C_{CPU}}/{2} = 256 \cdot 10^6$

Per intervenire sul CPI_{FP} evidenziamo il termine dipendente dalle istruzioni FP:

$C'_{CPU} = CPI'_{FP} \cdot N_{INST,FP} \mid \sum_{k=INT,LS,B} CPI_K \cdot N_{INST,K}$

dove $\sum_{k=INT,LS,B} CPI_K \cdot N_{INST,K} = (110 + 320 + 32) \cdot 10^6 = 462 \cdot 10^6$

quindi:

$CPI'_{FP} = \frac{C'_{CPU} - \sum_{k=INT,LS,B} CPI_K \cdot N_{INST,K}}{N_{INST,FP}} = \frac{(256-462)\cdot 10^6}{50 \cdot 10^6} < 0$ → IMPOSSIBILE

Per intervenire sul CPI_{LS} evidenziamo il termine dipendente dalle istruzioni LS:

$C'_{CPU} = CPI'_{LS} \cdot N_{INST,LS} + \sum_{k=FP,INT,B} CPI_K \cdot N_{INST,K}$

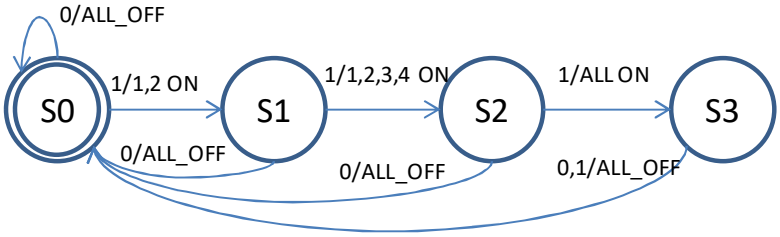
dove $\sum_{k=FP,INT,B} CPI_K \cdot N_{INST,K} = (50 \mid 110 \mid 32) \cdot 10^6 = 192 \cdot 10^6$

quindi:

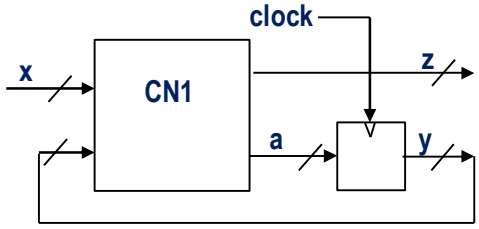
$CPI'_{LS} = \frac{C'_{CPU} - \sum_{k=FP,INT,B} CPI_K \cdot N_{INST,K}}{N_{INST,LS}} = \frac{(256-192)\cdot 10^6}{80 \cdot 10^6} = \frac{64}{80} = 0.8$

Quindi, il nuovo CPI delle istruzioni LS dovra' essere 0.8 per raddoppiare la velocita' del programma.

ESERCIZIO 7



Modello di Mealy:




```
`timescale 100ms/1ms
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; initial begin x=0; #40 x=1; end
wire[1:0] STAR = Xxx.STAR;
wire[2:0] z=Xxx.z;
XXX Xxx(x,z,clock,reset_);
endmodule

module XXX(x,z,clock,reset_);
input clock,reset_;
input x;
output[2:0] z;
reg[1:0] STAR;
```

```
parameter S0='B00,S1='B01,S2='B10,S3='B11;
assign z =(STAR==S0)?((x==0)?'B000:'B001):
          (STAR==S1)?((x==0)?'B000:'B011):
          (STAR==S2)?((x==0)?'B000:'B111):
           'B000;
always @(reset_==0) #1 STAR<=S0;
always @(posedge clock) if(reset_==1) #3
casex(STAR)
        S0:STAR<=(x==0)?S0:S1;
        S1:STAR<=(x==0)?S0:S2;
        S2:STAR<=(x==0)?S0:S3;
        S3:STAR<=(x==0)?S0:S0;
endcase
endmodule
```