COMPITO di ARCHITETTURA DEI CALCOLATORI del 21-09-2018

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".

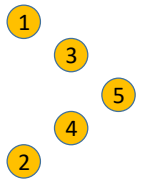
```
float x[2][3] = {1,2,3,4,5,6};
void gep(float c[2][3],int n, float *res) {
   int i,j,k,p,q,m;
   float temp,t[2],sum,max;
   for(j=0;j<n-1;j++) {
      max = fabs(c[j][j]);
      p=j;
      for(m=j+1;m<n;m++) {
         if(fabs(c[m][j]) >= max) {
            max = c[m][j];
            p = m;
         }
      }
      if(p != j) {
         for(q=j;q<n+1;q++) {
            temp = c[j][q];
            c[j][q] = c[p][q];
            c[p][q] = temp;
         }
      }
      for(i=j+1;i<n;i++) {
         temp = c[i][j] / c[j][j];
         for(k=j;k<=n;k++) {
            c[i][k] = c[i][k] - (temp * c[j][k]);
         }
      }
   }
   t[n-1] = c[n-1][n] / c[n-1][n-1];
   for(i=n-2;i>=0;i--) {
      sum = 0;
      for(j=i+1;j<n;j++) {
        sum = sum + (c[i][j] * t[j]);
      }
      t[i] = (c[i][n] - sum)/c[i][i];
   }
   *res = t[0];
}
int main () {
   float out;
   gep(x,2,&out);
   print_float(out);
}
```

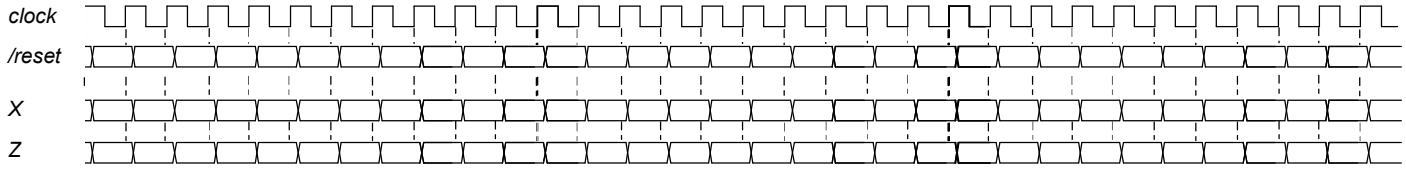
2) [7/38] Si consideri una cache di dimensione 96B e a 3 vie di tipo write-back/write-non-allocate. La dimensione del blocco e' 8 byte, il tempo di accesso alla cache e' 4 ns e la penalita' in caso di miss e' pari a 40 ns, la politica di rimpiazzamento e' FIFO. Il processore effettua i seguenti accessi in cache, ad indirizzi al byte: 127, 113, 163, 111, 140, 161, 115, 224, 222, 241, 216, 313, 416, 523, 691, 716, 831, 910, 1011, 1118. 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] Determinare quale istruzione assembly corrisponde alla seguente stringa binaria 0000 0010 0001 0000 1000 0000 0010 0000.

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

7) [8/38] **Realizzare** in Verilog una rete sequenziale secondo il modello di Moore 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 21-09-2018

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 21-09-2018

SOLUZIONE

ESERCIZIO 1

```
.data
x: .float 1.0,2.0,3.0,4.0,5.0,6.0
.text
.globl main
gep:
#_____CALL_FRAME_____
# float t[2]:      8B
#_____Totale       8B
addi $sp,$sp,-8
# c in a0, n in a1, res in a2
# i in t0, j in t1, k in t2, p in t3
# q in t4, m in t5
# temp in f4, sum in f5, max in f6
addi  $t9,$0,3      #t9=3=dim.riga
#for1
add   $t1,$0,$0     #j=0
gep_for1start:
   addi  $t6,$a1,-1 #t6=n-1
   slt   $t7,$t1,$t6#j<?n-1
   beq   $t7,$0,gep_for1end

   mult  $t1,$t9    #j*3
   mflo  $t7
   add   $t7,$t7,$t1#j*3+j
   sll   $t7,$t7,2  #4*(j*3+j)
   add   $t7,$a0,$t7#&c[j][j]
   lwc1  $f7,0($t7) #c[j][j]
   abs.s $f6,$f7    #max=fabs()

   add   $t3,$t1,$0 #p=j
   #for11
   addi  $t5,$t1,1       #m=j+1
   gep_for11start:
      slt   $t7,$t5,$a1  #m<?n
      beq   $t7,$0,gep_for11end
      mult  $t5,$t9      #m*3
      mflo  $t7
      add   $t7,$t7,$t1  #m*3+j
      sll   $t7,$t7,2    #4*(m*3+j)
      add   $t7,$a0,$t7  #&c[m][j]
      lwc1  $f7,0($t7)   #c[m][j]
      abs.s $f8,$f7      #fabs()
      #if111
      c.lt.s $f8,$f6     #()>=?max
      bc1t gep_if111end
         mov.s $f6,$f7   #max=c[m][j]
         add   $t3,$t5,$0#p=m
      gep_if111end:
      add  $t5,$t5,1     #m++
      j gep_for11start
   gep_for11end:
   #if11
   beq $t3,$t1,gep_if11end#p==?j
      #for111
      add  $t4,$t1,$0     #q=j
      gep_for111start:
         addi  $t6,$a1,1  #n+1
         slt   $t7,$t4,$t6#q<?n+1
         beq   $t7,$0,gep_for111end

         mult  $t1,$t9    #j*3
         mflo  $t7
         add   $t7,$t7,$t4#j*3+q
         sll   $t7,$t7,2  #4*(j*3+q)
         add   $t7,$a0,$t7#&c[j][q]
         lwc1  $f4,0($t7) #temp=c[j][q]

         mult  $t3,$t9    #p*3
         mflo  $t8
         add   $t8,$t8,$t4#p*3+q
         sll   $t8,$t8,2  #4*(p*3+q)
         add   $t8,$a0,$t8#&c[p][q]
         lwc1  $f8,0($t8) #c[p][q]

         swc1  $f8,0($t7) #c[j][q]=c[p][q]
         swc1  $f4,0($t8) #c[p][q]=temp

         add   $t4,$t4,1  #++q
         j gep_for111start
      gep_for111end:
   gep_if11end:
   #for12
   addi $t0,$t1,1         #i=j+1
   gep_for12start:
      slt  $t7,$t0,$a1    #i<?n
      beq  $t7,$0,gep_for12end

      mult $t0,$t9        #i*3
      mflo $t7
      add  $t7,$t7,$t1    #i*3+j
      sll  $t7,$t7,2      #4*(i*3+j)
      add  $t7,$a0,$t7    #&c[i][j]
      lwc1 $f7,0($t7)     #c[i][j]

      mult $t1,$t9        #j*3
      mflo $t8
      add  $t8,$t8,$t1    #j*3+j
      sll  $t8,$t8,2      #4*(j*3+j)
      add  $t8,$a0,$t8    #&c[j][j]
      lwc1 $f8,0($t8)     #c[j][j]

      div.s $f4,$f7,$f8   #temp=cij/cjj

      #for121
      add $t2,$t1,$0        #k=j
      gep_for121start:
         slt   $t7,$a1,$t2  #n<?k
         bne   $t7,$0,gep_for121end

         mult  $t0,$t9      #i*3
         mflo  $t7
         add   $t7,$t7,$t2  #i*3+k
         sll   $t7,$t7,2    #4*(i*3+k)
         add   $t7,$a0,$t7  #&c[i][k]
         lwc1  $f7,0($t7)   #c[i][k]

         mult  $t1,$t9      #j*3
         mflo  $t8
         add   $t8,$t8,$t2  #j*3+k
         sll   $t8,$t8,2    #4*(j*3+k)
         add   $t8,$a0,$t8  #&c[j][k]
         lwc1  $f8,0($t8)   #c[j][k]

         mul.s $f9,$f4,$f8  #temp*cjk
         sub.s $f9,$f7,$f9  #cik-()
         swc1 $f9,0($t7)    #c[i][k]=()

         add $t2,$t2,1      #k++
         j gep_for121start
      gep_for121end:

      addi $t0,$t0,1 #i++
      j gep_for12start
   gep_for12end:

   add  $t1,$t1,1  #j++
   j gep_for1start
gep_for1end:

addi $t8,$a1,-1 #t8=n1=n-1
mult $t8,$t9    #n1*3
mflo $t7
add  $t7,$t7,$a1#n1*3+n
sll  $t7,$t7,2  #4*(n1*3+n)
add  $t7,$a0,$t7#&c[n1][n]
lwc1 $f7,0($t7) #c[n1][n]
mult $t8,$t9    #n1*3
mflo $t7
add  $t7,$t7,$t8#n1*3+n1
sll  $t7,$t7,2  #4*(n1*3+n1)
add  $t7,$a0,$t7#&c[n1][n1]
lwc1 $f8,0($t7) #c[n1][n1]

div.s $f9,$f7,$f8 #cn1n/cn1n1
sll  $t7,$t8,2  #4*n1
add  $t7,$t7,$sp#&t[n-1]
swc1 $f9,0($t7) #t[n-1]=()

#for2
addi $t0,$a1,-2 #i=n-2
gep_for2start:
   slt   $t7,$t0,$0 #i<?0
   bne   $t7,$0,gep_for2end

   mtc1  $0,$f5     #sum=0.0
   add   $t1,$t0,1  #j=i+1
   #for21
   gep_for21start:
      slt   $t7,$t1,$a1#j<?n
      beq   $t7,$0,gep_for21end

      mult  $t0,$t9    #i*3
      mflo  $t7
      add   $t7,$t7,$t1#i*3+j
      sll   $t7,$t7,2  #4*(i*3+j)
      add   $t7,$a0,$t7#&c[i][j]
      lwc1  $f7,0($t7) #c[i][j]

      sll   $t8,$t1,2  #4*j
      add   $t8,$t8,$sp#&t[j]
      lwc1  $f8,0($t8) #t[j]

      mul.s $f9,$f7,$f8#cij*tj
      add.s $f5,$f5,$f9#sum+=()

      addi $t1,$t1,1   #j++
      j gep_for21start
   gep_for21end:

   mult  $t0,$t9       #i*3
   mflo  $t7
   add   $t7,$t7,$t0   #i*3+i
   sll   $t7,$t7,2     #4*(i*3+i)
   add   $t7,$a0,$t7   #&c[i][i]
   lwc1  $f7,0($t7)    #c[i][i]

   mult  $t0,$t9       #i*3
   mflo  $t7
   add   $t7,$t7,$a1   #i*3+n
   sll   $t7,$t7,2     #4*(i*3+n)
   add   $t7,$a0,$t7   #&c[i][n]
   lwc1  $f8,0($t7)    #c[i][n]

   sub.s $f10,$f8,$f5  #cin-sum
   div.s $f11,$f10,$f7 #()/cii
   sll   $t8,$t0,2     #4*i
   add  $t8,$t8,$sp    #&t[i]
   swc1 $f11,0($t8)    #t[i]=()

   add  $t0,$t0,-1     #i--
   j gep_for2start
gep_for2end:

lwc1 $f7,0($sp)   #t[0]
swc1 $f7,0($a2)   #*res=()
addi $sp,$sp,-8
jal  $ra

main:
#_______CALL FRAME_____________
# saved variable:   fp,ra     8B
# local variable:   out=0(fp) 4B
#_____________________Totale   12B
addi $sp,$sp,-12
sw   $fp,8($sp)
sw   $ra,4($sp)
add  $fp,$sp,$0
la   $a0,x      #a0=&x
addi $a1,$0,2   #a1=2
add  $a2,$fp,$0 #a2=&out
jal  gep
add  $t0,$fp,$0
lwc1 $f12,0($t0)#f12=out
addi $v0,$0,2   #print_float
syscall
lw   $ra,4($fp)
lw   $fp,8($fp)
addi $sp,$sp,12
jr   $ra
```

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 = FIFO, Thit = 4, Tpen = 40, 20 references.
=== T    X    XM  XT  XS  XB  H [SET]:USAGE [SET]:MODIF  [SET]:TAG
=== R  127   15   3   3   7  0 [3]:2,0,0 [3]:0,0,0  [3]:3,-,-
=== W  113   14   3   2   1  0 [2]:2,0,0 [2]:0,0,0  [2]:3,-,-
=== R  163   20   5   0   3  0 [0]:2,0,0 [0]:0,0,0  [0]:5,-,-
=== W  111   13   3   1   7  0 [1]:2,0,0 [1]:0,0,0  [1]:3,-,-
=== R  140   17   4   1   4  0 [1]:1,2,0 [1]:0,0,0  [1]:3,4,-
=== W  161   20   5   0   1  1 [0]:2,0,0 [0]:1,0,0  [0]:5,-,-
=== R  115   14   3   2   3  1 [2]:2,0,0 [2]:0,0,0  [2]:3,-,-
=== W  224   28   7   0   0  0 [0]:1,2,0 [0]:1,0,0  [0]:5,7,-
=== R  222   27   6   3   6  0 [3]:1,2,0 [3]:0,0,0  [3]:3,6,-
=== W  241   30   7   2   1  0 [2]:1,2,0 [2]:0,0,0  [2]:3,7,-
```

COMPITO di ARCHITETTURA DEI CALCOLATORI del 21-09-2018

SOLUZIONE

```
=== R  216   27   6   3   0  1 [3]:1,2,0 [3]:0,0,0   [3]:3,6,-
=== W  313   39   9   3   1  0 [3]:0,1,2 [3]:0,0,0   [3]:3,6,9
=== R  416   52  13   0   0  0 [0]:0,1,2 [0]:1,0,0   [0]:5,7,13
=== W  523   65  16   1   3  0 [1]:0,1,2 [1]:0,0,0   [1]:3,4,16
=== R  691   86  21   2   3  0 [2]:0,1,2 [2]:0,0,0   [2]:3,7,21
=== W  716   89  22   1   4  0 [1]:2,0,1 [1]:0,0,0   [1]:22,4,16 (out: XM=13 XT=3  XS=1 )
=== R  831  103  25   3   7  0 [3]:2,0,1 [3]:0,0,0   [3]:25,6,9  (out: XM=15 XT=3  XS=3 )
=== W  910  113  28   1   6  0 [1]:1,2,0 [1]:0,0,0   [1]:22,28,16 (out: XM=17 XT=4  XS=1 )
=== R 1011  126  31   2   3  0 [2]:2,0,1 [2]:0,0,0   [2]:31,7,21 (out: XM=14 XT=3  XS=2 )
=== W 1118  139  34   3   6  0 [3]:1,2,0 [3]:0,0,0   [3]:25,34,9 (out: XM=27 XT=6  XS=3 )
---------------------------------------
P1 Nmiss=17   Nhit=3  Nref=20  mrate=0.850000   AMAT=38
```

LISTA BLOCCHI USCENTI:

CONTENTUTI dei 4 SET al termine:

ESERCIZIO 3

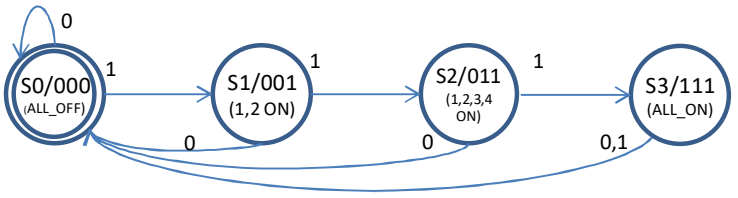
Dai primi 6 bit vediamo che l'opcode e' 0b000000: questo corrisponde ad una istruzione di tipo R come si puo' osservare dalla tabella delle istruzioni MIPS. Quindi i restanti bit possono essere raggruppati come segue:

000000	10000	10000	10000	00000	100000
opcode	**rs**	**rt**	**rd**	**shamt**	**funct**

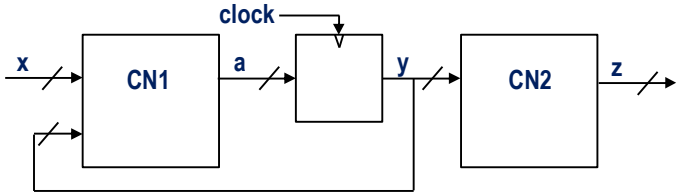
In particolare dato che il campo funct=0b100000=0x20 si vede dalla medesima tabella che l'istruzione corrispondente e' **add**.
A questo punto si possono interpretare i campi rs,rt,rd che in questo caso sono tutti identici e corrispondono al registro 0b10000=16 ovvero al registro s0. Quindi l'istruzione corrispondente è:

```
add $s0, $s0, $s0
```

ESERCIZIO 4



Modello di Moore:




```
`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)?'B000:
          (STAR==S1)?'B001:
          (STAR==S2)?'B011:
          'B111;
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
```