COMPITO di ARCHITETTURA DEI CALCOLATORI del 11-04-2019

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, 18/19": 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) [11/30] 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).

```
int a[10] = {51, 11, 32, 34, 11, 51, 46, 5, 62, 27};

void shellsort(int a[], int n)
{
  int i, j, increment, tmp;
  for(increment = n/2; increment > 0; increment /= 2) {
    for(i = increment; i < n; i++) {
      tmp = a[i];
      for(j = i; j >= increment; j -= increment) {
        if(tmp < a[j-increment])
          a[j] = a[j-increment];
        else
          break;
      }
      a[j] = tmp;
    }
  }
}
```

```
int main()
{
  int i, n = 10;
  shellsort(a,n);
  for(i = 0; i < n; i++) {
    print_int(a[i]);
    print_string(" ");
  }
  exit(0);
}
```

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)
00+04	**shift left logical**	`sllv $1,$2,10`	$1 = $2 << $3	Shift left by variable
00+06/00+07	**shift right (l=logical,a=arithmetic)**	`srlv/srav $1,$2,10`	$1 = $2 >> $3	Shift right by variable (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 / $f1=(single)$f0	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	Serv.No.($v0)	INPUT Arguments	OUTPUT Args
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

Service Name	Serv.No.($v0)	INPUT Arguments	OUTPUT Arguments
read_float	6	---	$f0=float
read_double	7	---	$f0-f1=double
read_string	8	$a0=address of input buffer, $a1=max chars to read	---
sbrk	9	$a0=Number of bytes to be allocated	$v0=pointer to allocated memory
exit	10	---	---

COMPITO di ARCHITETTURA DEI CALCOLATORI del 11-04-2019

2) [5/30] Si consideri una cache di dimensione 320B e a 5 vie di tipo write-back/write-non-allocate. La dimensione del blocco e' 32 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) [4/30] Spiegare il funzionamento del FLIP-FLOP-D negative edge-triggered: riportare tabella della verita' e sintesi a livello di porte e per ogni riga della tabella della verita' dimostrare perché si ottiene quella data uscita usando il digramma temporale (in particolare per quanto riguarda la sensibilità sui fronti in discesa).

7) [10/30] Descrivere e sintetizzare l'Unità XXX che emette un byte generato in accordo alla legge di cui sotto. Il byte deve permanere all'uscita out di XXX per un numero di clock esattamente pari a numero_clock = byte * 2 e deve essere notificato (contestulamente alla presentazione del risultato in uscita) dal fatto che la variabile go passa da 0 ad 1 per un ciclo di clock. I byte generati soddisfano la condizione di essere numeri dispari. Tracciare il diagramma di temporizzazione come verifica della correttezza dell'unità XXX (il modulo Testbench e' riportato). 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.

```
module testbench;
 reg reset_; initial begin reset_=0; #22 reset_=1; #600; $stop; end
 reg clock ; initial clock =0; always #5 clock <=(!clock);
 wire[7:0] COUNT=Xxx.COUNT;
 wire[7:0] BYTE =Xxx.BYTE;
 wire      STAR =Xxx.STAR;
 wire      go;
 wire[7:0] out;
 XXX Xxx(go,out, clock,reset_);
endmodule
```

clock

COMPITO di ARCHITETTURA DEI CALCOLATORI del 11-04-2019

SOLUZIONE

ESERCIZIO 1

```
.data
a:   .word 51, 11, 32, 34, 11
     .word 51, 46, 5, 62, 27
sp: .asciiz " "

.globl main
.text
#-----------------------------------
--
# a0=a,a1=n
t0=i,t1=j,t2=increm,t3=tmp
shellsort:
   addi $t8,$0,2    #costante=2
   div  $a1,$t8     #n/2
   mflo $t2         #increm=(.)
ss_for1_ini:
    slt  $t9,$0,$t2 #increm>?0
   beq  $t9,$0,ss_for1_end
#-----for1 body start
   add  $t0,$0,$t2
ss_for2_ini:
   slt  $t9,$t0,$a1#i<?n
   beq  $t9,$0,ss_for2_end
#-----for2 body start
   sll  $t3,$t0,2  #i*4
   add  $t3,$t3,$a0#&a[i]
   lw   $t3,0($t3) #tmp=a[i]
   add  $t1,$0,$t0 #j=i
ss_for3_ini:
   slt  $t9,$t1,$t2#j<?increm
   bne  $t9,$0,ss_for3_end
#-----for3 body start
   sub  $t4,$t1,$t2#j-increm
   sll  $t4,$t4,2  #(.)*4
   add  $t4,$t4,$a0#&a[(.)]
   lw   $t4,0($t4) #a[(.)]
    slt  $t9,$t3,$t4#tmp<?(.)
   beq  $t9,$0,ss_else
#-----if body start
   sll  $t5,$t1,2  #j*4
   add  $t5,$t5,$a0#&a[j]
   sw   $t4,0($t5) #a[j]=a[..]
   j    ss_if_end
#-----if body end
ss_else:
   j ss_for3_end #break
ss_if_end:
#-----for3 body end
   sub  $t1,$t1,$t2#j=-increm
   j ss_for3_ini
ss_for3_end:
       sll  $t5,$t1,2  #j*4
   add  $t5,$t5,$a0#&a[j]
   sw   $t3,0($t5) #a[j]=tmp
#-----for2 body end
   addi $t0,$t0,1  #i++
   j ss_for2_ini
ss_for2_end:
#-----for1 body end
   div  $t2,$t8     #increm/2
   mflo $t2         #increm=(.)
   j ss_for1_ini
ss_for1_end:
   jr   $ra

#-----------------------------------
--
main:
   la   $a0,a       # &a
   addi $a1,$0,10   # n
   jal  shellsort
   add  $t0,$0,$0   # i=0
main_forini:
    slt  $t9,$t0,$a1# i<?n
   beq  $t9,$0,main_forend
   sll  $t2,$t0,2   # i*4
   la   $t3,a       # &a
   add  $t2,$t2,$t3# &a[i]
   lw   $a0,0($t2)  # a[i]
   addi $v0,$0,1    # serv.1
   syscall
   la   $a0,sp      # &sp
   addi $v0,$0,4    # serv.4
   syscall
   addi $t0,$t0,1   #i++
   j    main_forini
main_forend:
   addi $v0,$0,10   # serv.10
   syscall
```

```
Console
5 11 11 27 32 34 46 51 51 62
```

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=320/32/5=2, XM=X/B, XS=XM%S, XT=XM/S:

```
A=5, B=32, C=320, RP=LRU, Thit=4, Tpen=40, 20 references:
=== T    X    XM  XT  XS  XB  H [SET]:USAGE [SET]:MODIF  [SET]:TAG
=== R   55    1   0   1  23  0 [1]:4,0,0,0,0 [1]:0,0,0,0,0  [1]:0,-,-,-,-
=== W  173    5   2   1  13  0 [1]:3,4,0,0,0 [1]:0,0,0,0,0  [1]:0,2,-,-,-
=== R  115    3   1   1  19  0 [1]:2,3,4,0,0 [1]:0,0,0,0,0  [1]:0,2,1,-,-
=== W  119    3   1   1  23  1 [1]:2,3,4,0,0 [1]:0,0,1,0,0  [1]:0,2,1,-,-
=== R  222    6   3   0  30  0 [0]:4,0,0,0,0 [0]:0,0,0,0,0  [0]:3,-,-,-,-
=== W  947   29  14   1  19  0 [1]:1,2,3,4,0 [1]:0,0,1,0,0  [1]:0,2,1,14,-
=== R  618   19   9   1  10  0 [1]:0,1,2,3,4 [1]:0,0,1,0,0  [1]:0,2,1,14,9
=== W  449   14   7   0   1  0 [0]:3,4,0,0,0 [0]:0,0,0,0,0  [0]:3,7,-,-,-
=== R  534   16   8   0  22  0 [0]:2,3,4,0,0 [0]:0,0,0,0,0  [0]:3,7,8,-,-
=== W  748   23  11   1  12  0 [1]:4,0,1,2,3 [1]:0,0,1,0,0  [1]:11,2,1,14,9  (out: XM=1  XT=0  XS=1 )
=== R  877   27  13   1  13  0 [1]:3,4,0,1,2 [1]:0,0,1,0,0  [1]:11,13,1,14,9 (out: XM=5  XT=2  XS=1 )
=== W  919   28  14   0  23  0 [0]:1,2,3,4,0 [0]:0,0,0,0,0  [0]:3,7,8,14,-
=== R  283    8   4   0  27  0 [0]:0,1,2,3,4 [0]:0,0,0,0,0  [0]:3,7,8,14,4
=== W  143    4   2   0  15  0 [0]:4,0,1,2,3 [0]:0,0,0,0,0  [0]:2,7,8,14,4   (out: XM=6  XT=3  XS=0 )
=== R  591   18   9   0  15  0 [0]:3,4,0,1,2 [0]:0,0,0,0,0  [0]:2,9,8,14,4   (out: XM=14 XT=7  XS=0 )
=== W  644   20  10   0   4  0 [0]:2,3,4,0,1 [0]:0,0,0,0,0  [0]:2,9,10,14,4  (out: XM=16 XT=8  XS=0 )
=== R  770   24  12   0   2  0 [0]:1,2,3,4,0 [0]:0,0,0,0,0  [0]:2,9,10,12,4  (out: XM=28 XT=14 XS=0 )
=== W  845   26  13   0  13  0 [0]:0,1,2,3,4 [0]:0,0,0,0,0  [0]:2,9,10,12,13 (out: XM=8  XT=4  XS=0 )
=== R  961   30  15   0   1  0 [0]:4,0,1,2,3 [0]:0,0,0,0,0  [0]:15,9,10,12,13 (out: XM=4  XT=2  XS=0 )
=== W  194    6   3   0   2  0 [0]:3,4,0,1,2 [0]:0,0,0,0,0  [0]:15,3,10,12,13 (out: XM=18 XT=9  XS=0 )
-------------------------------------
P1 Nmiss=19   Nhit=1  Nref=20  mrate=0.950000   AMAT=th+mrate*tpen=42
```

CONTENTUTI dei 5 SET al termine:

LISTA BLOCCHI USCENTI:

COMPITO di ARCHITETTURA DEI CALCOLATORI del 11-04-2019

SOLUZIONE

ESERCIZIO 3

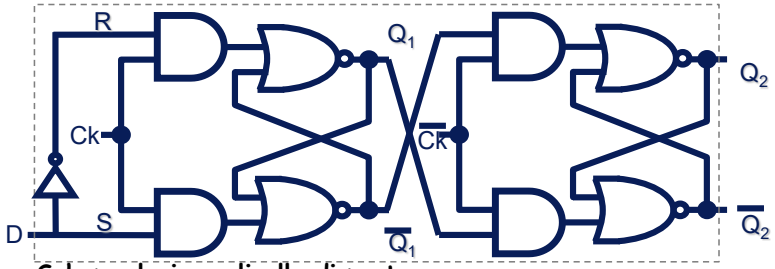
Quando CK=1, il master Clocked-SR latch e' trasparente e lo slave e' opaco. Quindi il valore di D si propaga su Q1. Quando CK=0, il master diventa opaco e lo slave trasparente: il valore di Q1 si propaga su Q, ma Q1 resta scollegato da D. QUINDI: qualunque valore sia presente su D un momento PRIMA del fronte in discesa del clock da 1 a 0, tale valore di D viene copiato su Q immediatamente DOPO la disscesa del clock. In tutti gli altri momenti Q mantiene il suo vecchio valore, perche' c'e' sempre un Clocked-SR latch opaco che blocca il cammino da D a Q.

FLIP FLOP D *negative edge triggered*

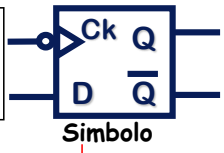
In questo caso vogliamo sensibilita' solo sul fronte in discesa

D	Ck	Q
X	1	Q
X	0	Q
X	⌜	Q
0	⌞	0
1	⌞	1

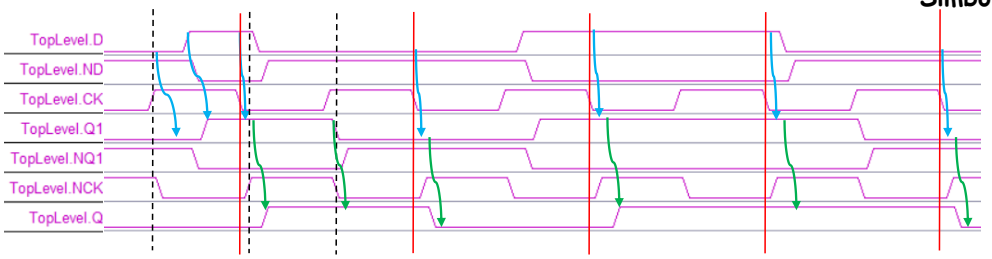
Tabella di Verità



Layout simbolico: 44 transistor
=4*NAND+4*(NAND+NOT)+ 2 per /Ck
+ 2 per la not su D =4*16+4*(4+2)+2+2



Simbolo



ESERCIZIO 7

Codice VERILOG:

```
module XXX(go,out, clock,reset_); //uso modello Mealy-Ritardato
input        clock, reset_;
output       go;
output [7:0] out;                 //go,out fanno le veci di 'z'

reg STAR; parameter S0=0,S1=1;    //STAR,COUNT,BYTE: definizione stato
reg [7:0]    COUNT;               //
reg [7:0]    BYTE;                //
reg          GO;  assign go=GO;   //GO,OUT fanno le veci di 'OUTR'
reg [7:0]    OUT; assign out=OUT; //

always @(reset_==0) begin STAR=S0; GO<=0; BYTE<=3; COUNT<=3; OUT<=0; end //RESET

always @(posedge clock or negedge reset_) if (reset_==1) #0 //evoluzione degli stati
  casex(STAR)
   S0:  begin GO<=1; COUNT<=BYTE; OUT<=BYTE; BYTE<=BYTE+2; STAR<=S1; end
   S1:  begin GO<=0; COUNT<=(COUNT-1); STAR<=(COUNT>2)?S1:S0; end
  endcase

endmodule
```

Diagramma temporale: