COMPITO di ARCHITETTURA DEI CALCOLATORI del 24-01-2024

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/WATCH, ETC

NOTA: dovrà essere consegnato l'elaborato dell'es.1 come file **<COGNOME>.s** e quelli dell'es. 4 come files **<COGNOME>.v** e **<COGNOME>.png**

1) [10/30] Trovare il codice assembly RISC-V corrispondente al seguente micro-benchmark (**utilizzando solo e unicamente istruzioni dalla tabella sottostante**), rispettando le convenzioni di uso dei registri dell'assembly (riportate qua sotto, per riferimento).

```
int arr[7] = {84, 44, 35, 22, 52, 81, 80};
void swap(int arr[], int i) {
  int temp = arr[i];
  arr[i] = arr[i+1];
  arr[i+1] = temp;
}
void bubbleSort(int arr[], int size) {
  for (int i = 0; i < size - 1; i++)
    for (int j = 0; j < size - i - 1; j++)
      if (arr[j] > arr[j+1]) swap(arr, j);
}
```

Nota: 'int' è un intero a 64 bit.

```
int main() {
  int size = sizeof(arr) / sizeof(arr[0]);
  bubbleSort(arr, size);
  print_int(arr[3]);
  exit(0);
}
```

RISCV Instructions (RV64IMFD) **v230703**

Instruction coding (hexadecimal) funct7/imm	funct3	opcode	Instruction	Example	Register operation	Meaning (** instructions available only in RV64, i.e. 64-bit case)
00	0	33/3b	add	`add/addw x5,x6,x7`	x5 ← x6 + x7	Add two operands; exception possible (addw**)
20	0	33/3b	subtract	`sub/subw x5,x6,x7`	x5 ← x6 – x7	Subtracts two operands; exception possible (subw**)
imm	0	13/1b	add immediate	`addi/addiw x5,x6,100`	x5 ← x6 + 100	Add a constant ; exception possible (addiw**)
01	0	33/3b	multiply	`mul/mulw x5,x6, x7`	x5 ← x6 * x7	(signed/word) Lower 64 bits of 128-bits product (mulw**)
01	1	33	multiply high	`mulh x5,x6,x7`	x5 ← x6 * x7	Higher 64bits of 128-bits product
01	4	33/3b	division	`div/divw x5,x6,x7`	x5 ← x6/x7	(signed/word) division (divw**)
01	6	33/3b	reminder	`rem/remw x5,x6,x7`	x5 ← x6 % x7	Reminder of the division (remw**)
00	2/3	33	set on less than	`slt/sltu x5,x6,x7`	if (x6 < x7) x5 ← 1; else x5 ← 0	signed compare x6 and x7 (less than)
imm	2/3	13	set on less than immediate	`slti/sltiu x5,x6,100`	if (x6 < 100) x5← 1; else x5 ← 0	unsigned compare x6 and 100 (less than)
00	7/6/4	33	and / or / xor	`and/or/xor x5,x6,x7`	x5← x6&x7 / x6|x7 / x6^ x7	Logical AND/OR/XOR register operand
imm	7/6/4	13	and /or / xor immediate	`andi/ori/xori x5,x6,100`	x5 ← x6&100 / x6|100 / x6^100	Logical AND/OR/XOR constant operand
0	1	33/3b	shift left logical	`sll/sllw x5,x6,x7`	x5 ← x6 << x7	Shift left by register (sllw**)
imm	1	13/1b	shift left logical immediate	`slli/slliw x5,x6,10`	x5 ← x6 << 10	Shift left by the immediate value (slliw**)
0	5	33/3b	shift right logical	`srl/srlw x5,x6,x7`	x5 ← x6 >> x7	Shift right by register (srlw**)
imm	5	13/1b	shift right logical immediate	`srli/srliw x5,x6,10`	x5 ← x6 >> 10	Shift left by immediate value (srliw**)
20	5	33/3b	shift right arithmetic	`sra/sraw x5,x6,x7`	x5 ← x6 >> x7 (arith.)	Shift right by register (sign is preserved) (sraw**)
imm	5	13/1b	shift right arithmetic immediate	`srai/sraiw x5,x6,10`	x5 ← x6 >> 10 (arith.)	Shift right by immediate value (sraiw**)
imm	3/2/0	03	load dword / word / byte	`ld/lw/lb x5,100(x6)`	x5 ← MEM[x6+100]	Data from memory to register
imm	6/4	03	load word / byte unsigned	`lwu/lbu x5,100(x6)`	x5 ← MEM[x6+100]	Data from mem. To reg.; no sign extension (lwu**)
imm	3/2	23	store dword / word / byte	`sd/sw/sb x5,100(x6)`	MEM[x6+100] ← x5	Data from register to memory (sw**)
imm[31:12]	-	37	load upper immediate	`lui x5,0x12345`	x5 ← 0x1234'5000	Load most significant 20 bits
PSEUDOINSTRUCTION			load address	`la x5,var`	x5 ← &var **(PSEUDO INST.)** load address of 'var' in x5	**REAL INST.:** `lui x5,H20(&var);ori x5, L12(&var)` (H20=high 20 bits of &var; L12=low 12 bits of &var)
imm[31:12] (rd=0) / imm[11:0] (rs1=rs2=0)	- / 0	6f/63	jump/branch	`j/b label`	PC+=off (off=PC-&label) **(PS.INST.)**	**REAL INST.:** `jal x0,offset/beq x0,x0,offset`
imm[31:12] (rd=1)	-	6f	jump and link (offset)	`jal label`	x1←(PC+4); PC+=offset **(PS. INST.)**	**REAL INST.:** `jal x1,offset` (offset=PC-&label)
Imm (rd=0,rs=1)	0	67	return from procedure	`ret`	PC←x1 **(PSEUDO INST.)**	**REAL INST.:** `jalr x0,0(x1)`
imm	0	67	jump and link register	`jalr x1, 100(x5)`	x1 ← (PC + 4); PC=x5+100	Procedure return; indirect call
imm÷2	0/1	63	branch on equal / not-equal	`beq/bne x5,x6,100`	if (x5 = =/!= x6) PC=PC+100	Equal / Not-equal test; PC relative branch
00 (rs1=0,rs2=0,rd=0)	0	73	ecall	`ecall`	SEPC←PC;PC←STVEC;save PL/IE;PL=1;IE=0	Call OS (service number in a7); PL= privilege lev; IE=int.en.
08 (rs1=0,rs2=2,rd=0)	0	73	sret	`sret`	PC←SEPC; restore PL/IE	Exit supervisor mode; PL= privilege lev; IE=int.en.
PSEUDOINSTRUCTION			move	`mv x5,x6`	x5 ← x6 **(PSEUDO INST.)**	**REAL INST.:** `add x5,x0,x6`
PSEUDOINSTRUCTION			load immediate	`li x5,100`	x5 ← 100 **(PSEUDO INST.)**	**REAL INST.:** `addi x5,x0,100`
PSEUDOINSTRUCTION			no operation (nop)	`nop`	do nothing **(PSEUDO INST.)**	**REAL INST.:** `addi x0,x0,0`
{0,1} / {4,5}	0	53	floating point add/sub	`fadd/fsub.{s,d} f0,f1,f2`	f0←f1+f2 / f0←f1-f2	Single or double precision add / subtract
{8,9} / {c,d}	0	53	floating point multiplication/division	`fmul/fdiv.{s,d} f0,f1,f2`	f0←f1*f2 / f0←f1/f2	Single or double precision multiplication / division
PSEUDOINSTRUCTION			floating point move between f-regs	`fmv.{s,d} f0,f1`	f0←f1 **(PSEUDO INST.)**	**REAL INST.:** `fsgnj.{s,d} f0,f1,f1`
PSEUDOINSTRUCTION			floating point negate	`fneg.{s,d} f0,f1`	f0← – (f1) **(PSEUDO INST.)**	**REAL INST.:** `fsgnjn.{s,d} f0,f1,f1`
PSEUDOINSTRUCTION			floating point absolute value	`fabs.{s,d} f0,f1`	f0← | f1 | **(PSEUDO INST.)**	**REAL INST.:** `fsgnjx.{s,d} f0,f1,f1`
{50,51}	0/1/2	53	floating point compare	`fle/flt/feq.{s,d} x5,f0,f1`	x5← (f0<f1)	Single and double: compare f0 and f1 <=,<,==
{70,71} (rs2=0)	0	53	move between x (integer) and f regs	`fmv.x.{s,d} x5,f0`	x5←f0 (no conversion)	Copy (no conversion)
{78,79} (rs2=0)	0	53	move between f and x regs	`fmv.{s,d}.x f0,x5`	f0←x5 (no conversion)	Copy (no conversion)
imm	2	7	load/store floating point (32bit)	`flw/fsw f0,0(x5)`	f0←MEM[x5] / MEM[x5]←f0	Data from FP register to memory
imm	3	7	load/store floating point (64bit)	`fld/fsd f0,0(x5)`	f0←MEM[x5] / MEM[x5]←f0	Data from FP register to memory
21/20 (rs2=0)	7	53	convert to/from double from/to single	`fcvt.d.s/fcvt.s.d f0,f1`	f0← (double)f1 / f0← (single)f1	Type conversion
{60,61} (rs2=0)	7	53	convert to integer from {single,double}	`fcvt.w.{s,d} x5,f0`	x5← (int)f0	Type conversion
{68,69} (rs2=0)	7	53	convert to {single,double} from integer	`fcvt.{s,d}.w f0,x5`	f0← ({single,double})x5	Type conversion
{2c,2d} (rs2=0)	0	53	square root	`fsqrt.{s,d} f0,f1`	f0← square root of f1	Single or double square root
{10,11}	0/1/2	53	sign injection	`fsgnj/jn/jx.{s,d} f0,f1,f2`	f0←sgn(f2)|f1| / –sgn(f2)|f1| / sgn(f2)f1	Extract the mantissa and exp. from f1 and sign from f2

Register Usage

Register	ABI Name	Usage
x10-x11	**a0-a1**	arguments and results
x9, x18-x27	**s1, s2-s11**	Saved
x5-7, x28-x31	**t0-t2, t3-t6**	Temporaries
x12-x17	**a2-a7**	Arguments

Register	ABI Name	Usage
x0	**zero**	The constant value 0
x8, x2	**s0/fp, sp**	frame pointer, stack pointer
x1, x3	**ra, gp**	return address, global pointer
x4	**tp**	thread pointer

Register	ABI Name	Usage
f10-f11	**fa0-fa1**	Argument and Return values
f8-f9, f18-f27	**fs0-fs1, fs2-fs11**	Saved registers
f0 – f7, f28-f31	**ft0-ft7, ft8-ft11**	Temporaries registers
f12-17	**fa2-fa7**	Function arguments

System calls

Service Name	Serv.No.(a7)	INPUT Arguments	OUTPUT Args
print_int	1	a0=integer to print	---
print_float	2	fa0=float to print	---
print_double	3	fa0=double to print	---
print_string	4	a0=address of ASCIIZ string to print	---
read_int	5	---	a0=integer

Service Name	Serv.No.(a7)	INPUT Arguments	OUTPUT Arguments
read_float	6	---	fa0=float
read_double	7	---	fa0=double
read_string	8	a0=address of input buffer, a1=max chars to read	---
sbrk	9	a0=Number of bytes to be allocated	a0=pointer to allocated memory
exit	10	---	---

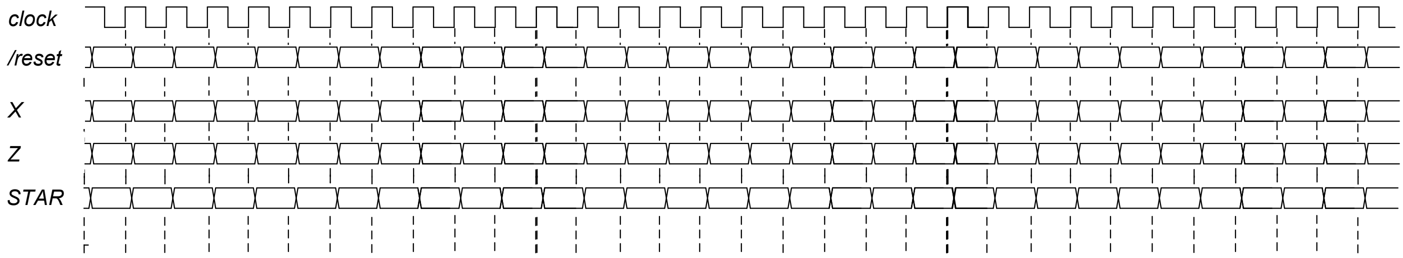
COMPITO di ARCHITETTURA DEI CALCOLATORI del 24-01-2024

2) [5/30] Si consideri una cache di dimensione 32B e a 2 vie di tipo write-back/write-non-allocate. La dimensione del blocco e' 4 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: 277, 163, 123, 181, 300, 221, 275, 184, 182, 201, 176, 173, 176, 183, 251, 176, 201, 180. 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) [5/30] Quale formato viene usato dall'istruzione `ld x5,100(x20)`? Descrivere in dettaglio tale formato indicando la funzione, lunghezza e posizione dei campi che lo compongono. Quindi valorizzare ciascun campo per l'istruzione data.

4) [10/10] Descrivere e sintetizzare in Verilog una rete sequenziale utilizzando il modello di Mealy con un ingresso X su un bit e una uscita Z su un bit che funziona nel seguente modo: devono essere riconosciute le sequenze <u>INTERALLACCIATE</u> 1,1,1,1, e 1,0,0,1; l'uscita Z va a 1 (per 1 ciclo di clock) se è presente una delle due sequenze. Gli stimoli di ingresso sono dati dal seguente modulo Verilog Testbench.

Tracciare il diagramma di temporizzazione [4/10 punti] 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. Modello del diagramma temporale da tracciare:



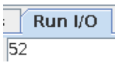

```
module TopLevel;
reg reset_;initial begin reset_=0; #22 reset_=1; #300; $stop; end
reg clock ;initial clock=0; always #5 clock <=(!clock);
reg X;
wire Z;
wire [2:0] STAR=Xxx.STAR;
initial begin X=0;
wait(reset_==1); #5
@(posedge clock); X<=0; @(posedge clock); X<=0; @(posedge clock); X<=1; @(posedge clock); X<=1;
@(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<=1; @(posedge clock); X<=1; @(posedge clock); X<=1; @(posedge clock); X<=1;
@(posedge clock); X<=0; @(posedge clock); X<=0; @(posedge clock); X<=1; @(posedge clock); X<=0;
$finish;
end
XXX Xxx(X,Z,clock,reset_);
endmodule
```

COMPITO di ARCHITETTURA DEI CALCOLATORI del 24-01-2024

SOLUZIONE (rev1.1)

ESERCIZIO 1

```
.data
arr:    .dword 84, 44, 35, 22, 52, 81, 80
arr_end:
.text
.globl main

#-----------------------------------------
swap: # input: a0=arr, a1=i
  # Load arr[i] into temp
  slli t0,a1,3    # offset: i * 8
  la   a2,arr     # &arr
  add  a2,a2,t0   # &arr[8]
  ld   t1,0(a2)   # Load arr[i] into t1
  ld   t2,8(a2)   # Load arr[i+1] into t1
  sd   t2,0(a2)   # Store arr[i+1] into arr[i]
  sd   t1,8(a2)   # Store arr[i] into arr[i+1]
  ret

#-----------------------------------------
bubbleSort:
  addi sp,sp,-32 # allocate frame
  sd   ra,24(sp)
  sd   s0,16(sp)
  sd   s1,8(sp)
  sd   s2,0(sp)

  li   s0,0       # i=0
  addi s2,a1,-1   # size-1
bs_for1_start:
  beq  s0,s2,bs_for1_end

  # Initialize j = 0
  li   s1, 0
bs_for2_start:
  sub  t0,s2,s0  # size-1-i
  beq  s1,t0,bs_for2_end# j==?size-1-i ->exitfor

  slli t1,s1,3   # j * 8
  la   t2,arr    # &arr
  add  t2,t2,t1  # &arr[j]
  ld   t3,0(t2)  # arr[j]
  ld   t4,8(t2)  # arr[j+1]
  slt  t5,t3,t4  # aj1 <? aj no-->cont
  beq  t5,x0,if_end
  mv   a0,t2     # a0=&arr
  mv   a1,s1     # a1=j
  call swap
if_end:

  addi s1,s1,1   # ++j
  b bs_for2_start
bs_for2_end:

  addi s0,s0,1  # ++i
  b bs_for1_start
bs_for1_end:

  # Restore registers from the stack
  ld ra, 24(sp)
  ld s0, 16(sp)
  ld s1, 8(sp)
  ld s2, 0(sp)
  addi sp, sp, 32
  ret

#-----------------------------------------
main:
  la   a0,arr
  la   a1,arr_end
  sub  a1,a1,a0   # sizeof(arr)
  srai a1,a1,3    # a1=(.)/8
  la   a0,arr     # a0=&arr
  call bubbleSort

  la   t0,arr     # &arr
  addi t0,t0,24   # &arr[3]
  ld   a0,0(t0)   # arr[3]
  li   a7,1       # print_int
  ecall

  li   a7,10      # exit
  ecall
```



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=32/4/2,     XM=X/B,    XS=XM%S,    XT=XM/S.
A=2, B=4, C=32, RP=LRU, Thit=4, Tpen=40, 18 references:
=== T    X   XM  XT  XS  XB  H [SET]:USAGE [SET]:MODIF  [SET]:TAG
=== R  277   69  17   1   1  0 [1]:1,0 [1]:0,0  [1]:17,-
=== W  163   40  10   0   3  0 [0]:1,0 [0]:0,0  [0]:10,-
=== R  123   30   7   2   3  0 [2]:1,0 [2]:0,0  [2]:7,-
=== W  181   45  11   1   1  0 [1]:0,1 [1]:0,0  [1]:17,11
=== R  300   75  18   3   0  0 [3]:1,0 [3]:0,0  [3]:18,-
=== W  221   55  13   3   1  0 [3]:0,1 [3]:0,0  [3]:18,13
=== R  275   68  17   0   3  0 [0]:0,1 [0]:0,0  [0]:10,17
=== W  184   46  11   2   0  0 [2]:0,1 [2]:0,0  [2]:7,11
=== R  182   45  11   1   2  1 [1]:0,1 [1]:0,0  [1]:17,11
=== W  201   50  12   2   1  0 [2]:1,0 [2]:0,0  [2]:12,11
=== R  176   44  11   0   0  0 [0]:1,0 [0]:0,0  [0]:11,17
=== W  173   43  10   3   1  0 [3]:1,0 [3]:0,0  [3]:10,13
=== R  176   44  11   0   0  1 [0]:1,0 [0]:0,0  [0]:11,17
=== W  183   45  11   1   3  1 [1]:0,1 [1]:0,1  [1]:17,11
=== R  251   62  15   2   3  0 [2]:0,1 [2]:0,0  [2]:12,15
=== W  176   44  11   0   0  1 [0]:1,0 [0]:1,0  [0]:11,17
=== R  201   50  12   2   1  1 [2]:1,0 [2]:0,0  [2]:12,15
=== W  180   45  11   1   0  1 [1]:0,1 [1]:0,1  [1]:17,11
-------------------------------------
P1 Nmiss=12   Nhit=6  Nref=18  mrate~=0.666667   AMAT=th+mrate*tpen~=30.6
```

LISTA BLOCCHI USCENTI:

```
(out:  XM=30 XT=7  XS=2 )
(out:  XM=40 XT=10 XS=0 )
(out:  XM=75 XT=18 XS=3 )

(out:  XM=46 XT=11 XS=2 )
```

CONTENUTI dei SET al termine

ESERCIZIO 3

Formato I – istruzione `ld x5, 100(x20)`

- L'istruzione `ld x5, 100(x20)` utilizza il formato I.
- I vari campi che lo compongono sono illustrate in figura con le posizioni iniziali e finali dei bit di ciascun campo, da cui si deriva facilmente il numero di bit per campo
- Sono inoltre indicati i valori assunti da ciascun campo per l'istruzione data

31 – 20	19 – 15	13 – 12	11 – 7	6 – 0
0000 0110 0100	10100	011	00101	0000011
imm12[11:0]	rs1	funct3	rd	opcode

- rs1 è l'indice del registro sorgente numero-1 (che contiene il registro base da cui leggere l'operando in memoria)
- imm12 è un intero con segno che rappresenta l'offset da aggiungere al contenuto del registro base per ottenere l'indirizzo effettivo da cui leggere in memoria
- rd è l'indice del registro destinazione in cui viene caricato il valore letto dalla memoria
- I campi {opcode,funct3} indicano l'istruzione specifica che nel caso di load-dword (ld) valgono {3,3}

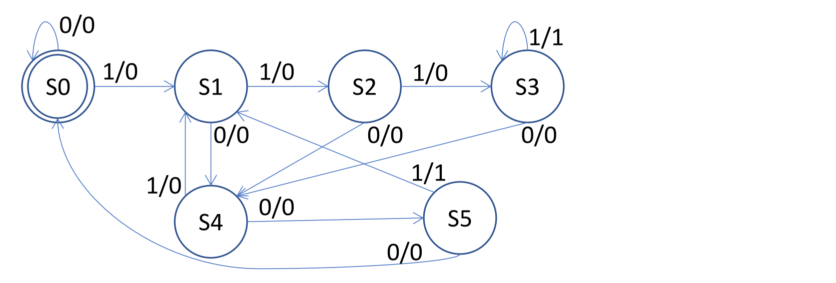
COMPITO di ARCHITETTURA DEI CALCOLATORI del 24-01-2024

SOLUZIONE (rev1.1)

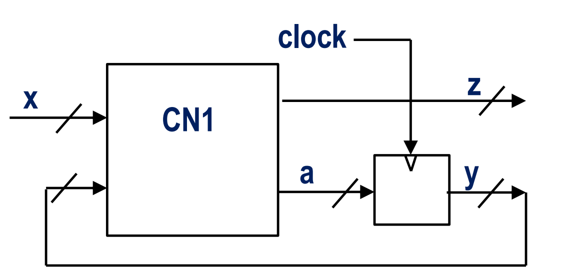
ESERCIZIO 4

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

Diagramma degli Stati



Schema di riferimento per Mealy



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

```
module XXX(x,z,clock,reset_);
input clock,reset_,x;
output z;
reg [2:0] STAR;
parameter S0='B000 , S1='B001 , S2='B010 , S3='B011, S4='B100, S5='B101;
always @(reset_==0) begin STAR<=0 ; end
assign z=(STAR==S3 && x==1) + (STAR==S5 && x==1);
always @(posedge clock) if (reset_==1) #3
  casex(STAR)
    S0: begin STAR<=(x==0)?S0:S1; end
    S1: begin STAR<=(x==0)?S4:S2; end
    S2: begin STAR<=(x==0)?S4:S3; end
    S3: begin STAR<=(x==0)?S4:S3; end
    S4: begin STAR<=(x==0)?S5:S1; end
    S5: begin STAR<=(x==0)?S0:S1; end
  endcase
endmodule
```

Diagramma di Temporizzazione: