COMPITO di ARCHITETTURA DEI CALCOLATORI 1 del 17-04-2015 **MATRICOLA**_____

COGNOME_____

NOME_____

MODULO RETI LOGICHE:
I SEGUENTI ESERCIZI VALGONO 50% DEL VOTO FINALE (40/80) PER GLI INFORMATICI (ARCHITETTURA 1) E (1 E 2) IL 33% DEL VOTO FINALE (20/60) PER GLI ALTRI (ARCHITETTURA 1A)

Esercizio 1

Date le funzioni:

$$f_1 = \Sigma_4(0,2,3,5,7,9,10,11,13,14)$$
$$f_2 = \Pi_4(1,2,3,5,6,8,9,12,13,15)$$

disegnare una rete per la loro esecuzione costituita da un solo multiplexer a quattro ingressi, due bit/ingresso ed una costituita da multiplexer, sempre a quattro ingressi e due bit/ingresso, quanti sono ritenuti necessari.

Valutare le due soluzioni rispetto alla *velocità di risposta* in termini di numero massimo di ritardi porta, sapendo che un multiplexer ha un ritardo pari a due; rispetto alla *complessità* come numero complessivo di porte logiche; alla *flessibilità,* ossia alla possibilità di adattare ciascuna soluzione a funzioni diverse da quelle date con il minimo numero di modifiche da apportare al circuito.

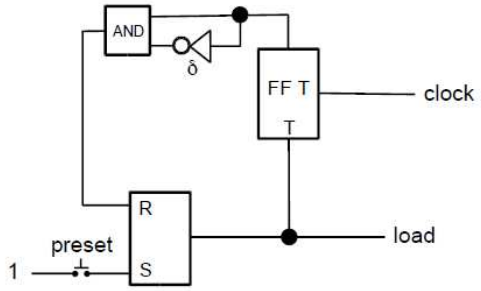
Esercizio 2

Un registro di 4 bit, basato su flip-flop di tipo D e con ingressi e uscite paralleli, deve funzionare alternando i seguenti passi:

1) eseguire il preset di un qualsiasi valore compreso tra 0 e 15 applicato agli ingressi; per questa operazione il segnale di *load* deve rimanere alto per due periodi di clock e a tale scopo viene utilizzato il circuito illustrato in figura.

2) decrementare fino a zero il valore precaricato con il passo 1); questa operazione è avviata attraverso la pressione di un pulsante di *start*. Quando viene raggiunto il valore 0, il contatore si ferma ed ogni ulteriore funzionamento avviene con la ripetizione del passo 1) seguito dal passo 2). Il conteggio deve avvenire in sincronismo con il clock.

Progettare la logica necessaria per il funzionamento del registro richiesto ed analizzare il circuito di figura, tenendo in debita considerazione il ritardo indicato con δ.



MODULO CALCOLATORI ELETTRONICI:
I SEGUENTI ESERCIZI VALGONO 50% DEL VOTO FINALE (40/80) PER ARCHITETTURA 1 E 66% DEL VOTO FINALE (40/60) PER ARCHITETTURA 1A. VALGONO 40/40 PER GLI ALTRI.

1. [9] Si consideri una cache di dimensione 160B e a 5 vie di tipo write-back. 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: 123, 639, 327, 679, 878, 639, 133, 654, 125, 454, 122, 654, 939, 626, 954, 724, 254, 829, 154, 828, 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 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.
2. [5] Rappresentare in double precision IEEE-754, il valore 480/7 arrotondato al valore piu' vicino.
3. [16] Trovare il codice assembly MIPS corrispondente del seguente programma (**utilizzando solo e unicamente istruzioni dalla tabella sottostante**), **rispettando le convenzioni di utilizzazione dei registri dell'assembly MIPS** (riportate in calce). In alternativa, si usi l'assembly x86 anziche' MIPS. Le funzioni non definite sono da considere esterne al programma.

```
void transpose(void *dest, void *src, int src_h, int
    src_w)
{
   int i, j;
   double (*d)[src_h] = dest, (*s)[src_w] = src;
   for (i = 0; i < src_h; i++)
      for (j = 0; j < src_w; j++)
         d[j][i] = s[i][j];
}
```

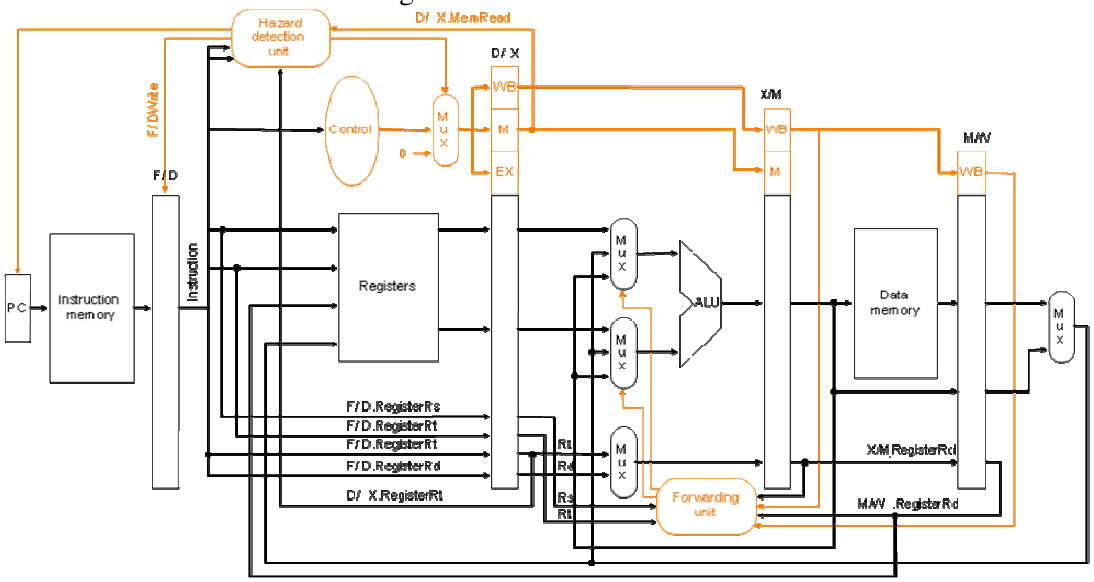
```
int main()
{
   int i, j;
   double a[3][5] = {{ 0, 1, 2, 3, 4 },
       { 5, 6, 7, 8, 9 },
       { 1, 0, 0, 0, 42}};
   double b[5][3];
   transpose(b, a, 3, 5);

   for (i = 0; i < 5; i++)
      for (j = 0; j < 3; j++) {
         print_double(b[i][j]);
         if (j==2) print_string("\n"); else print_string("
");
      }
   return 0;
}{
```

4 [10] Nella seguente figura e' rappresentata una pipeline di un processore. Supponendo che su di essa si esegua il seguente codice:

```
lw   $5, 40($2)
add  $6, $3, $2
or   $7, $2, $1
and  $8, $4, $3
sub  $9, $2, $1
```

Al ciclo 1, subito prima che si inizi ad eseguire queste istruzioni, lo stato del processore e': a) il PC vale 100 che e'l'indirizzo dell'istruzione lw ; b) ogni registro ha il valore iniziale di 10 piu' il numero del corrispondente registro (es. il registro $8 contiene 18); c) ogni parola in memoria contiene il valore iniziale 1000 piu' il valore dell'indirizzo di tale parola (es. la locazione Memory[8] ha il valore inizale 1008). Determinare il valore sui fili in uscita da ogni stadio al ciclo 5.



Instructions

Instruction	Example	Meaning	Comments
add	`add $1,$2,$3`	$1 = $2 + $3	3 operands; exception possible
subtract	`sub $1,$2,$3`	$1 = $2 - $3	3 operands; exception possible
add immediate	`addi $1,$2,100`	$1 = $2 + 100	+ constant; exception possible
subtract immediate	`subi $1,$2,100`	$1 = $2 - 100	- constant; exception possible
Multiplication	`mult $1, $2`	Hi,Lo= $1 x $2	64-bit Signed Product ; result in Hi,Lo
Division	`div $1, $2`	Hi= $1 % $2, Lo = $1 / $2	Signed division
move from Hi	`mfhi $1`	$1 = Hi	Create copy of Hi
move from Lo	`mflo $1`	$1 = Lo	Create 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 logical	`srl $1,$2,10`	$1 = $2 >> 10	Shift right by constant
load word	`lw $1,100($2)`	$1 = Memory[$2+100]	Data from memory to register
load byte	`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	`sw $1,100($2)`	Memory[$2+100] = $1	Data from register to memory
store byte	`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
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 absolute 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 $1,$f2`	$f2=$1	Data from gen.reg. to C1 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]	
store floating point (32bit)	`swc1 $f0,0($1)`	Memory[$1]←$f0	
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	Register 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	Register 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

Name	Usage
$f0, $f1, …, $f31	Single precision floating point registers
$f0, $f2, …, $f30	Double precision floating point 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	---	---