

DA RESTITUIRE INSIEME AGLI ELABORATI e A TUTTI I FOGLI

→ NON USARE FOGLI NON TIMBRATI

→ ANDARE IN BAGNO PRIMA DELL'INIZIO DELLA PROVA

→ NO APPUNTI O FOGLI PERSONALI, NO TELEFONI, SMARTPHONE, ETC

COGNOME \_\_\_\_\_

NOME \_\_\_\_\_

SVOLGIMENTO DELLA PROVA (selezionare una delle seguenti 4 opzioni):

 PER GLI STUDENTI DI "ARCHITETTURA DEI CALCOLATORI – A.A. 2015/16 e 16/17": es. N.1+2+3+7. PER GLI STUDENTI DEGLI ANNI PRECEDENTI che devono svolgere sia il modulo CALCOLATORI che il modulo RETI: es. N.1+2+3+4+6. PER GLI STUDENTI DEGLI ANNI PRECEDENTI che devono svolgere SOLO il modulo CALCOLATORI es. N.1+2+3+4+5. PER GLI STUDENTI DEGLI ANNI PRECEDENTI che devono svolgere SOLO il modulo RETI: es. N.6+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) [14] Trovare il codice assembly MIPS corrispondente del seguente programma (**utilizzando solo e unicamente istruzioni dalla tabella sottostante e rispettando le convenzioni di utilizzazione dei registri dell'assembly MIPS** riportate qua sotto, per riferimento).

```

int count = 0;
void solve(int n, int col, int *hist) {
    int i, j;
    char s[2];
    s[1] = '\0';
    if (col == n) {
        print_str("\nNo. "); print_int(++count); print_str("\n----\n");
        for (i = 0; i < n; i++, print_str("\n"))
            for (j = 0; j < n; j++, print_str(s))
                s[0] = j == hist[i] ? 'Q' : ((i + j) & 1) ? ' ' : '.';
        return;
    } // fineif
    for (i = 0; i < n; i++) { //inizio for
        for (j = 0; j < col && !(hist[j] == i || abs(hist[j] - i) == col - j); j++);
        if (j < col) continue;
        hist[col] = i;
        solve(n, col + 1, hist);
    } //fine for
} //fine solve

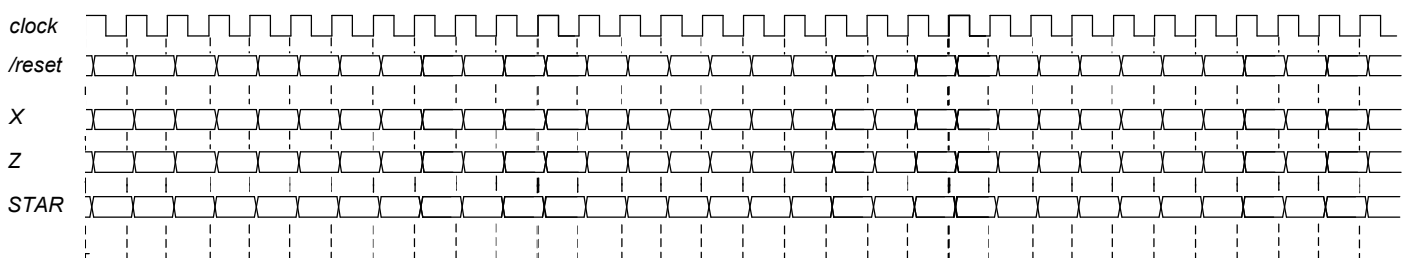
```

```

int main() {
    int hist[8];
    solve(8, 0, hist);
}

```

- 2) [8] Si consideri una cache di dimensione 32B e a 4 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: 355, 373, 315, 319, 322, 347, 318, 349, 334, 348, 377, 319, 283, 243, 391, 344, 370, 345, 61, 394. 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) [8] Calcolare e confrontare i tempi di esecuzione in spazio Kernel (comprensivi dei tempi di setup dei controller e di gestione della stampa stessa) della stampa di un testo di lunghezza 1024 byte nei tre casi in cui si gestisca l'operazione con la tecnica di: i) **polling**; ii) **interrupt**; iii) **DMA**. Si utilizzino i seguenti tempi: A) per il setup del DMA controller 20 cicli; B) per acknowledge di interrupt 4 cicli; C) per abilitazione di interrupt, per ritorno a User space e per ritorno da interrupt 2 cicli; D) per sbloccaggio utente 15 cicli; E) per passaggio di controllo allo scheduler e per riconoscimento dell'interrupt e lancio della routine di gestione dell'interrupt 3 cicli; F) nell'accesso ai registri di I/O della periferica (status, control, data): 2 cicli per ogni scrittura e 2 cicli per ogni lettura; G) ogni variabile temporanea e allocata in un registro del processore e ogni operazione del processore impiega sempre un ciclo (ALUJ, BRANCH, LOAD/STORE); G) si supponga che letture successive al registro di stato abbiano successo una volta ogni 10 accessi.
- 4) [4] Spiegare tramite un diagramma architetturale il funzionamento della paginazione inversa per la gestione della memoria virtuale assumendo di avere come ingresso un indirizzo di pagina virtuale VPN e come uscita un indirizzo di pagina fisica PPN.
- 5) [4] Spiegare tramite un diagramma architetturale il funzionamento della paginazione a tre livelli per la gestione della memoria virtuale assumendo di avere come ingresso un indirizzo di pagina virtuale VPN e come uscita un indirizzo di pagina fisica PPN.
- 6) [8] Sintetizzare una rete sequenziale utilizzando il modello di Moore con un ingresso X su tre bit e una uscita Z su tre bit che funziona nel seguente modo: l'uscita rappresenta un numero binario naturale tale che  $Z = (cx_2 + cx_1 + cx_0) \bmod 5$  essendo  $cx_2, cx_1, cx_0$  il numero degli 1 logici che sono stati presentati fino all'istante considerato agli ingressi  $X[2], X[1], X[0]$  rispettivamente. Rappresentare la macchina a stati finiti per tale rete di Moore, la tabella delle transizioni, le equazioni booleane delle reti CN1 e CN2 e il circuito sequenziale sincronizzato basato su flip-flop D.
- 7) [10] Descrivere e sintetizzare in Verilog la rete sequenziale descritta nell'esercizio 6 e il modulo TopLevel. **Tracciare il diagramma di temporizzazione** come verifica della correttezza dell'unità. 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.



**Instructions**

Instruction	Example	Meaning	Comments
add	add/addu \$1,\$2,\$3	\$1 = \$2 + \$3	(signed/unsigned) 3 operands; exception possible
subtract	sub/subu \$1,\$2,\$3	\$1 = \$2 - \$3	(signed/unsigned) 3 operands; exception possible
add immediate	addi/addiu \$1,\$2,100	\$1 = \$2 + 100	(signed/unsigned) + constant ; exception possible
multiplication	mult/multu \$1, \$2	Hi,Lo= \$1 x \$2	(signed/unsigned) 64-bit Product ; result in Hi,Lo
division	div/divu \$1, \$2	Hi= \$1 % \$2, Lo = \$1 / \$2	(signed/unsigned) division
move from Hi / move from Lo	mflhi/mflo \$1	\$1 = Hi (\$1 = Lo)	Create copy of Hi (Create a 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=arithmetic)	srl/sra \$1,\$2,10	\$1 = \$2 >> 10	Shift right by constant (in the arithmetic case, the sign is always preserved)
load word / load byte	lw/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 / store byte	sw/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
no operation	nop	Do nothing	Do nothing
load-linked	ll \$1,100(\$2)	\$1=Memory[\$2+100]	Read and start to monitor the given memory location
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)
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 opposite 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 <=, !=, <=, >=
mtcl/mfcl	mtcl/mfcl \$1,\$f2	\$f2=\$1 / \$1=\$f2	Data from gen.reg. \$1 to C1 reg. \$f2 (no conversion) / and viceversa
ctcl/cfcl	ctcl/cfcl \$1,\$cf2	\$cf2=\$1 / \$1=\$cf2	Data from gen.reg. to C1 CONTROL reg. (no conversion) / and viceversa
branch on false	bclf label	If (Temp == false) go to label	Temp is 'Condition-Code'
branch on true	bclt label	If (Temp == true) go to label	Temp is 'Condition-Code'
load floating point (32bit)	lwc1 \$f0,0(\$1)	\$f0←Memory[\$1]	Data from FP (C1) register to memory
store floating point (32bit)	swc1 \$f0,0(\$1)	Memory[\$1]←\$f0	Data from memory to FP (C1) register
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	Reg. Num.	Usage
\$zero	0	The constant value 0
\$s0-\$s7	16-23	Saved
\$t0-\$t9	8-15,24-25	Temporaries
\$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	---	---