

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

**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) [18] 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 i = 0, j = 0, k = 0;
void multiply (int m1, int n1, int a[2][2], int m2, int n2,
int b[2][2], int c[2][2]) {
    if (i >= m1) {
        return;
    } else if (i < m1) {
        if (j < n2) {
            if (k < n1) {
                c[i][j] += a[i][k] * b[k][j];
                k++;
                multiply(m1, n1, a, m2, n2, b, c);
            }
            k = 0; j++;
            multiply(m1, n1, a, m2, n2, b, c);
        }
        j = 0; i++;
        multiply(m1, n1, a, m2, n2, b, c);
    }
}

void display(int m1, int n2, int c[2][2]) {
    int i, j;
    for (i = 0; i < m1; i++) {
        for (j = 0; j < n2; j++) {
            print_int(c[i][j]);
            print_string(" ");
        }
        print_string("\n");
    }
}

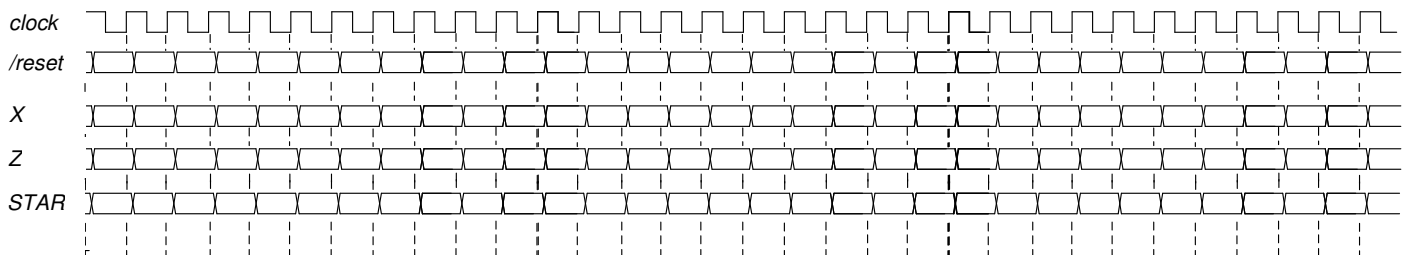
int main() {
    int x[2][2] = { 12, 56, 45, 78 };
    int y[2][2] = { 2, 6, 5, 8 };
    int z[2][2] = {0, 0, 0, 0 };
    multiply(2, 2, x, 2, 2, y, z);
    display(2, 2, z);
}
    
```

- 2) [8] Si consideri una cache di dimensione 128B 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: 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) [6] In un processore MIPS con pipeline determinare i cicli necessari per eseguire due iterazioni per il seguente frammento di codice sia nel caso di propagazione abilitata che di propagazione disabilitata. Nota: e' presente 1 delay-slot e l'accesso ai registri nella fase di decodifica e write-back possono essere sovrapposte.

```

add $1, $2, $3
L1: lw $4, 0($1)
    lw $5, 0($1)
    bne $4, $5, L1
nop
    
```

- 4) [4] Spiegare tramite un diagramma il funzionamento della tecnica di gestione dell'I/O con interrupt vettorizzato.
- 5) [4] Spiegare tramite un diagramma e un esempio il funzionamento della gestione dell'I/O a polling
- 6) [8] Sintetizzare una rete sequenziale utilizzando il modello di Mealy ritardato con un ingresso X su un bit e una uscita Z su un bit che riconosca le sequenze interallacciate 1,0,0,1. Rappresentare la macchina a stati finiti per tale rete logica, la tabella delle transizioni, le equazioni booleane delle reti CN1 e CN2 e il circuito sequenziale sincronizzato basato su flip-flop D.
- 7) [8] Descrivere e sintetizzare in Verilog la rete sequenziale descritta nell'esercizio 6 e il modulo TopLevel con sequenza di ingresso 0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,1,0,0,0,0. **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	mfmhi/mfelo \$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 (l=logical,a=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/mfc1	mtcl/mfc1 \$1,\$f2	\$P2=\$1 / \$1=\$f2	Data from gen.reg. \$1 to C1 reg. \$f2 (no conversion) / and viceversa
ctcl/cfc1	ctcl/cfc1 \$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	Name	Reg. Num.	Usage	Reg. Num.	Usage
\$zero	0	The constant value 0	\$v0-\$v1	2-3	Results	\$f0, \$f2	Return values
\$s0-\$s7	16-23	Saved	\$fp, \$sp	30,29	frame pointer, stack pointer	\$f12,\$f14	Function arguments
\$t0-\$t9	8-15,24-25	Temporaires	\$ra, \$gp	31,28	return address, global pointer	\$f20,\$f22,\$f24,\$f26,\$f28,\$f30	Saved registers
\$a0-\$a3	4-7	Arguments	\$k0-\$k1	26,27	Kernel usage	\$f4,\$f6,\$f8,\$f10,\$f16,\$f18	Temporaries registers

**System calls**

Service Name	Service Num. (Sv0)	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 ASCII string to print	---
read_int	5	---	Sv0=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	Sv0=pointer to the allocated memory
exit	10	---	---