Politecnico di Milano
Dip. di Elettronica, Informazione e Bioingegneria

prof.ssa	Anna	Antola	prof.	Giuseppe	Pelagatti
prof.	Luca	Breveglieri	prof.ssa	Donatella	Sciuto
prof.	Roberto Negrini		prof.ssa	Cristina Silvano	

AXO – Architettura dei Calcolatori e Sistemi Operativi

CAOS - COMPUTER ARCHITECTURE and OPERATING SYSTEMS

Prova di martedì 6 novembre 2018

Cognome____________________________ **Nome** ____________________________

Matricola_________________________ **Firma**_________________________________

Istruzioni / Instructions

Write/answer only in given tables and use blank pages of the PART_I_booklet for draft.

Books, notes, pocket calculators, mobile phones, PCs and tablets are forbidden during the exam.

The text of the exam should be given back in any case.

Exam duration 1 h : 45 m

Valore indicativo di domande ed esercizi, voti parziali e voto finale:

esercizio 1	**(6 punti)**	________________________
esercizio 2	**(3 punti)**	________________________
esercizio 3	**(7 punti)**	________________________
voto finale:	**(16 punti)**	________________________

esercizio n. 1 – linguaggio macchina – machine language

part one / prima parte – traduzione da C ad assembler – from C to MIPS assembly language

Given the C program above, translate it in MIPS assembly language. The memory model is the MIPS standard one and integer variables adopt a 32-bit representation. Do not optimize independent C statements.

The following hypotheses hold:

– the "frame pointer" register *fp* is not used

– local variables are allocated in registers, if possible

– registers must be saved only if required (by the caller or by the callee, according to the saving rules)

Answer to the following four questions (using the predefined tables):

1. Describe the static data segment giving, for each global variable or element in the global variable: (i) the symbolic content associated with the *starting address*, (ii) the offset of the starting address w.r.t the global pointer *gp*. Translate in MIPS assembly language the overall global variables declaration section.
2. Describe the activation frame of the ***checksum*** function and the allocation of ***checksum*** local variables and arguments into registers.
3. Translate in MIPS assembly language the **C statement *highlighted* in the main program.**
4. Translate in MIPS assembly language the C code of function ***checksum.***

```
/* global variables declaration section                  */
#define N 16

int vec [N]
int idx
int res = 0

int checksum (int * addr) {          /* checksum function*/
   int * ptr
   int val
   ptr = &val
   *ptr = *addr
   if (addr != vec) {
      addr--      /* pointers/addresses arithmetics!!! */
      *ptr = checksum (addr)
   } /* if */
   return *ptr
} /* checksum */

void main ( ) {                          /* main program */
   idx = ...                        /*IDX initialization */
   res = checksum (&vec [idx - 1])
}  /* main */
```

punto 1 – segmento dati statici / static data segment

contenuto simbolico / symbolic content	spiazzamento rispetto a / offset w.r.t. ***gp*** = **0x 1000 8000**	spiazzamento positivo o negativo ? ? / theoffset is pos or neg?	
			High addresses
RES			
IDX			
VEC [N – 1]			
VEC [...]	▨	▨	
VEC [0]			Low addresses

punto 1 – codice MIPS della sezione dichiarativa globale / global declaration section: MIPS code
`.data    0x 1000 0000  // static data segment`

punto 2 – area e registri di ***CHECKSUM*** / ***CHECKSUM*** activation frame and registers

area di attivazione di *CHECKSUM* **/ *CHECKSUM* activation frame**		
contenuto simbolico **/ symbolic content**	spiazz. rispetto a stack pointer **/ offset w.r.t. stack pointer**	
		High addr.
		Low addr.

allocazione dei parametri e delle variabili locali di *CHECKSUM* nei registri **/ allocation of *CHECKSUM* arguments and local variables into registers**	
parametro / variabile locale **argument / local variable**	Registro **/ register**

punto 3 – codice MIPS dello statement di *MAIN* / MIPS code of **MAIN statement to be translated**

punto 4 – codice MIPS dell’intera funzione ***CHECKSUM*** **/** MIPS code of ***CHECKSUM*** function
`CHECKSUM:  addiu  $sp, $sp,`
`           // ptr = &val`
`           // *ptr = *addr`
`IF:        // if (addr != vec)`
`THEN:      // addr--`
`           // *ptr = checksum (addr)`
`ENDIF:     // return *ptr`

seconda parte – assemblaggio e collegamento / assembler and linking

Given the two assembly language modules, fill the following tables:

1. Table 1 – object modules produced by the assembler
2. Table 2 – text (code) and data relocation bases of each object module
3. Table 3 – global symbols table and executable code table

Note that Table 1 for the GRAPHICS module is given, while Table 2 and Table 3 have to be filled also for GRAPHICS.

modulo MAIN	**modulo GRAPHIX**
`          .data`	`          .data`
`PIXEL     .word    64`	`COORD     .word    130`
`SPRITE    .space   80`	`          .text`
`          .text`	`          .globl   GRAPHICS`
`          .globl   MAIN`	`GRAPHICS: lw   $a0, COORD`
`MAIN:     lw   $t0, PIXEL`	`          jal  0`
`          beq  $0, $t0, MAINEND`	`          beq  $v0, $0, MAINEND`
`          sw   $t0, COORD`	`GRAPHEND: jr  $ra`
`          jal  GRAPHICS`	
`MAINEND:  sw   $v0, SPRITE`	

General rules when generating tables containing code:

- Represent operation codes and registers names in symbolic format
- Represent numeric constants in hex format, without the prefix 0x, and with the correct length

  example: instruction ***addi $t0, $t0, 15*** should be represented as ***addi $t0, $t0, 000F***

- In object modules numerical values to be relocated by the linker must be represented as "zero" values. Correct values will be defined in the executable code

  example: ***lw $t0, RIL*** should be represented as ***lw $t0, 0000($gp)*** in the object module, and as ***lw $t0, nnnn($gp)*** in the executable code, where the value *nnnn* is computed for the relocatable symbol *RIL* after the linking process

(1) – moduli oggetto **/ object modules**

modulo *MAIN*		**modulo *GRAPHICS***	
text dimension:		dimensione testo:	10 hex (16 dec)
Data dimension:		dimensione dati:	4

text		**testo**	
address	istruction	indirizzo	istruzione
		0	**lw** $a0, **0000**($gp)
		4	**jal** 000 0000
		8	**beq** $v0, $0, **0000**
		C	**jr** $ra
		10	

data		**dati**	
address	content (in hex)	indirizzo	contenuto (in esadecimale)
		0	0000 0082
		4	

Symbol table tipo può essere *T* (text) oppure *D* (data)			**tabella dei simboli** tipo può essere *T* (testo) oppure *D* (dato)		
symbol	Type T/D	value	simbolo	tipo	valore
MAIN			GRAPHICS	T	0000 0000
MAINEND			GRAPHEND	T	0000 000C
PIXEL			COORD	D	0000 0000
SPRITE					

Relocation table			**tabella di rilocazione**		
address	Op. code	symbol	indirizzo	cod. operativo	simbolo
			0	**lw**	COORD
			8	**beq**	MAINEND

(2) – memory starting addresses of modules	
MAIN	*GRAPHICS*
text:	text:
data:	data:

(3) – global symbol table

symbol	Final value		symbol	Final value
MAIN			GRAPHICS	
MAINEND			GRAPHEND	
PIXEL			COORD	
SPRITE				

(3) – executable code

text	
address	**code** (op. codes and regs in in symbolic form)
0040 0000	
0040 0004	
0040 0008	
0040 000C	
0040 0010	
0040 0014	
0040 0018	
0040 001C	
0040 0020	

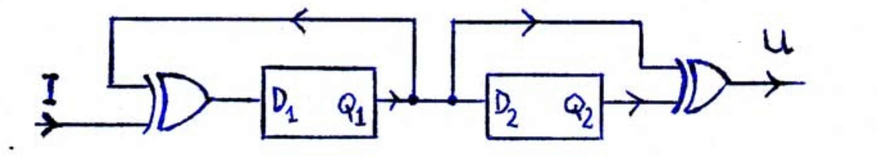
Data global table is not required.

esercizio n. 2 – logica digitale / digital logic

prima parte – logica sequenziale / sequential logic

Consider the circuit in figure. The circuit has:

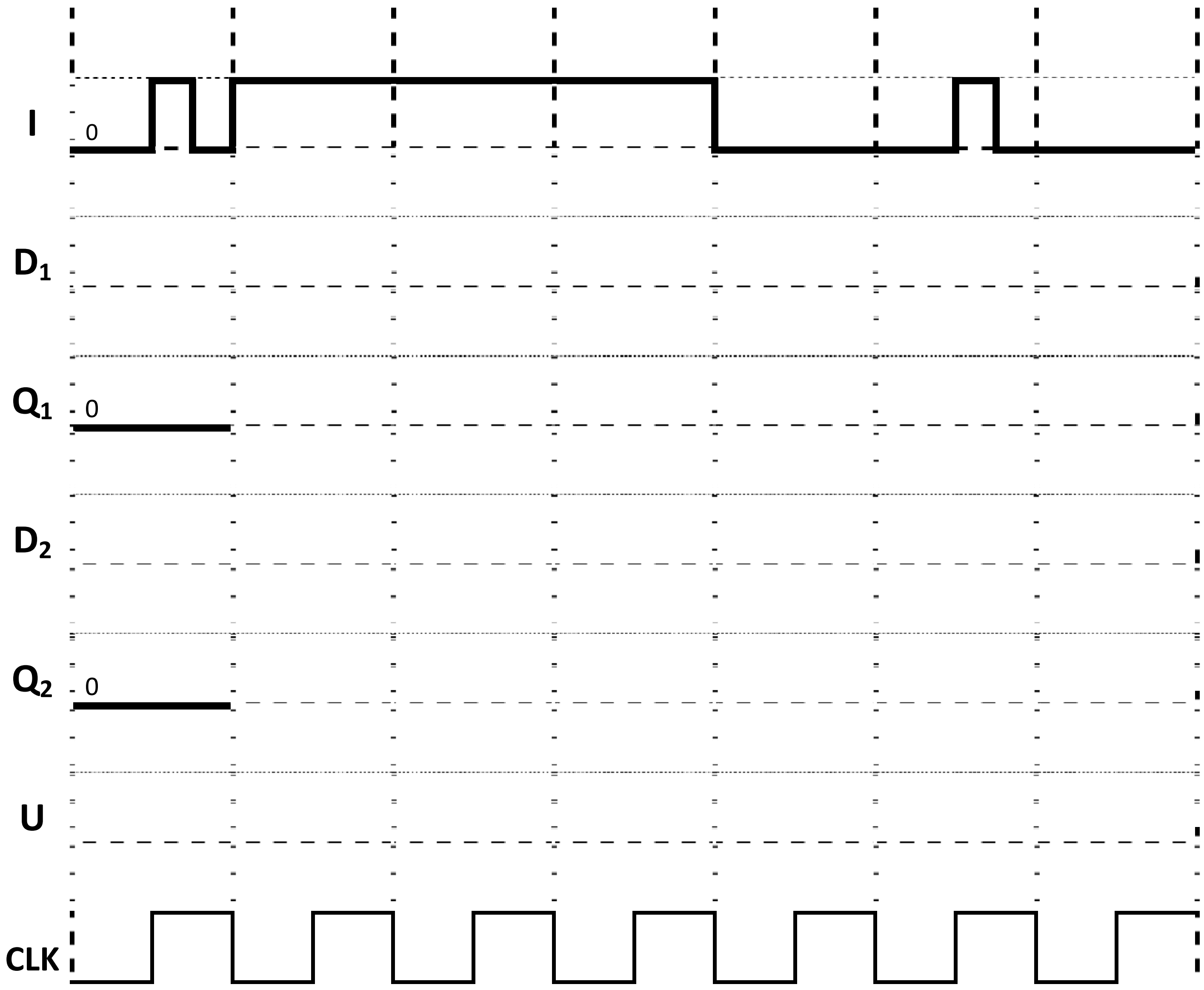
- two **D-type master-slave flip-flops** (D1, Q1 and D2, Q2), where Di is the input of FF i and Qi is the corresponding state/output
- one **input I** and one **output U**



Complete the temporal diagram below. It should be noticed that:

- Propagation delays of the logical gates and switching delays of FFs have to be ignored
- The output of the **master-slave** FF changes during the clock falling edge

diagramma temporale da completare



seconda parte – logica combinatoria / combinatorial logic

Synthetize in first canonical form (SOP), without optimization, the combinatorial circuit:

- receiving as input an unsigned integer binary number on four digits (**B3, B2, B1, B0**), thus with values ranging from 0 to 15
- producing an output **F** , that is set to 1 iff the input number has a value which is multiple of 2 and of 3 (including 0). In all other cases, F is 0.

Define the input/output table of the circuit:

B3	B2	B1	B0	F
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

Give the logical expression for F in the SOP form:

F = ____

esercizio n. 3 – microarchitettura del processore pipeline

prima parte – pipeline e segnali di controllo / pipeline and control signals

Refer to the architectural scheme "processore pipeline e campi registri interstadio".

Consider the following symbolic MIPS machine code, the starting execution address of the code, and the initial content of some processor registers and memory locations.

indirizzo	codice MIPS
0x 0040 0800	`sw   $t1, 0x 001A($t3)`
	`add  $t3, $t2, $t3`
	`beq  $t2, $t0, 0x FFFE`
	`addi $t2, $t1, 0x 1007`
	`lw   $t3, 0x 000F($t1)`

registro	contenuto iniziale
$t0	0x 0110 A010
$t1	0x 1001 0000
$t2	0x 0001 0015
$t3	0x 1000 FFF5
memoria	**contenuto iniziale**
0x 1000 4004	0x 0044 0FFF
0x 1000 4008	0x 11FF 0040
0x 1001 000F	0x 1001 1112
0x 1001 1BB5	0x 0600 C427

The pipeline is NOT optimized for control hazards. Consider the clock cycle where the above instructions are in the following pipeline stages:

WB	`sw   $t1, 0x 001A($t3)`
MEM	`add  $t3, $t2, $t3`
EX	`beq  $t2, $t0, 0x FFFE`
ID	`addi $t2, $t1, 0x 1007`
IF	`lw   $t3, 0x 000F($t1)`

1) Give the number of the considered clock cycle: ________________________

2) Compute the value of $t2 + $t3 in the *add* instruction:

____

3) Compute the address of the branch destination (remember that the offset in *beq* is a word-offset):

____

Is offset in *beq* positive or negative? ________________________

4) Compute the value of $t1 + 0x 1007 *addi* instruction (addition with immediate):

____

Fill the following tables.

Use the symbolic form for fields “Istruzione/instruction” and for fields representing “NumeroRegistro/RegisterNumber”. For all other fields use the hex notation. If the value cannot be computed, use **n.a./n.d.**

signals’ values at the input of interstage registers (immediately before the clock RISING edge)			
IF	**ID**	**EX**	**MEM**
registro IF/ID	***registro ID/EX***	***registro EX/MEM***	***registro MEM/WB***
▨	.WB.MemtoReg	.WB.MemtoReg	.WB.MemtoReg
▨	.WB.RegWrite	.WB.RegWrite	.WB.RegWrite
▨	.M.MemWrite	.M.MemWrite	▨
▨	.M.MemRead	.M.MemRead	▨
▨	.M.Branch	.M.Branch	▨
.PC	.PC	.PC	▨
.istruzione	.(Rs)	▨	▨
▨	.(Rt)	.(Rt)	▨
▨	.Rt	.R	.R
▨	.Rd	▨	▨
▨	.imm/offset esteso	.ALU_out ****************	.ALU_out
▨	.EX.ALUSrc	.Zero	.DatoLetto ****************
▨	.EX.RegDest	▨	▨

RegisterFile (RF) signals (immediately before clock FALLING edge)		
RF.RegLettura1	RF.RegScrittura	RF.DatoLetto1
RF.RegLettura2	RF.DatoScritto	RF.DatoLetto2

signals related to other functional units (immediately before the clock RISING edge)	
Mem.indirizzo	RegWrite
MemWrite	RegDest
MemRead	MemtoReg

seconda parte – gestione di conflitti e stalli – hazards and stalls

Consider the following machine instruction sequence (the multicycle diagram is given for your convenience)

```
1.  add   $3, $1, $5
2.  and   $1, $3, $2
3.  sw    $1, 16($3)
4.  beq   $5, $1, 8
```

ciclo di clock (clock cycle)

istruzione	1	2	3	4	5	6	7	8	9	10	11	12	13
1	IF	ID	EX	MEM	WB	▨	▨	▨	▨	▨	▨	▨	▨
2	▨	IF	ID	EX	MEM	WB	▨	▨	▨	▨	▨	▨	▨
3	▨	▨	IF	ID	EX	MEM	WB	▨	▨	▨	▨	▨	▨
4	▨	▨	▨	IF	ID	EX	MEM	WB	▨	▨	▨	▨	▨

Assume the pipeline:

(i) **optimized** for managing control hazards (evaluation of the branch condition in the ID stage)

(ii) provided with **all** the forwarding data-paths (**EX/EX**, **MEM/EX**, **MEM/MEM**, **EX/ID** and **MEM/ID**).

Answer to the following questions:

a) **draw** in the temporal diagram above **all** data dependencies

b) fill the first three column of Table 1 with data dependencies resulting in **hazards**

c) **draw** in **Table 2** the pipeline temporal diagram, giving evidence of the forwarding paths that **may** be activated to solve hazards, and inserting stalls when necessary for effectiveness of the forwarding

d) **complete** last column of **Table 1** with the forwarding paths actually activated and the number of resulting stalls

Tabella 1 (numero di righe non significativo)

istruzione	istruzione da cui dipende	registro coinvolto	percorso di propagazione e stalli	ciclo di clock in cui è attivo il percorso di propagazione

Tabella 2

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1														
2														
3														
4														

e) Refer to the architectural scheme PROCESSORE PIPELINE CON STALLO E PROPAGAZIONE (pipeline architecture with forwarding paths and hazard detection unit) and consider the forwarding paths in Table 1 (of type EX/EX and MEM/EX, only). Fill – for each clock cycle in which at least one propagation is active – one column of Table 3. Use the notation S.V. (stall value) and n.a. (not applicable) if necessary

Use the same column for paths active in the same clock cycle.

		Tabella 3				
	ciclo di clock dove è attiva almeno una propagazione					
	istruzione che ne usufruisce					
	tipo(i) di propagazione					
segnali di ingresso all'unità di propagazione	ID / EX.Rs					
	ID / EX.Rt					
	EX / MEM.R					
	EX / MEM.RegWrite					
	MEM / WB.R					
	MEM / WB.RegWrite					
	MUX di propagazione interessato					
segnali di uscita dell'unità di propagazione	ingresso di selezione MUX PA					
	ingresso di selezione MUX PB					