IFT 1227 – Architecture des ordinateurs

Devoir 4

- Remise : Le 28.04.2023 avant minuit.

I. Extension du processeur MIPS version un cycle

Il s'agit d'ajouter une implémentation de quelques instructions aux processeurs MIPS version **un cycle** et version **multicycles** vus en cours.

Les schémas donnés représentent les processeurs MIPS version un cycle et multicycles avec les instructions supportées **`lw, sw, j, addi, or, and, add, sub, beq.`** Les instructions **`jr, andi`** seront ajoutées lors de la séance de démonstration.

Les détails des instructions à implémenter :

Code d'opération	Nom	Description	Opération	funct
010001 (17_{10})	**`index2Adr rd, rs,rt`**	Index en Adresse	**`R[rd] = R[rs]*4 + R[rt]`**	**`111111 (3F`$_{16}$`)`**
000011 (3_{10})	**`jal addr`**	Saut et lien	**`R[$ra] = PC + 4, PC = JTA`**	**`-`**

- Étendre les implémentations des chemins de données et des contrôleurs pour traiter les instructions **`index2Adr`** et **`jal`** (solutions papier schématiques).
- Intégrer votre solution schématique MIPS un cycle dans l'implémentation VHDL fournie.
- Ajuster le testbench (**`testbench.vhd`**) et le module mémoire d'instructions (**`imem.vhd`**) du processeur MIPS un cycle pour vérifier les nouvelles instructions. Fichier **`imem.vhd`** modélise la mémoire d'instructions (ROM) avec un programme enregistré contenant les instructions implémentées.

```
mem(0)  := X"20020005"; --     addi $v0, $0, 5        # $v0(2) = 5
mem(1)  := X"2003000c"; --     addi $v1, $0, 12       # $v1(3) = 12
mem(2)  := X"2067fff7"; --     addi $a3, $v1,-9       # $a3(7) = $v1(3)(12) - 9 = 3
mem(3)  := X"00e22025"; --     or   $a0, $a3, $v0     # $a0(4) = $a3(7) or $v0(2) = 3 or 5 = 7
mem(4)  := X"00642824"; --     and  $a1, $v1, $a0     # $a1(5) = $v1(3) and $a0(4)= 12 and 7 = 4
mem(5)  := X"00a42820"; --     add  $a1, $a1, $a0     # $a1(5) = $a1(5) + $a0(4) = 4 + 7 = 11
mem(6)  := X"10a7000a"; --     beq  $a1, $a3, end     # $a1(5)==$a3(7)? end: PC=PC+4; 11==3 ? PC=PC+4
mem(7)  := X"0064202a"; --     slt  $a0, $v1, $a0     # $v1(3)<$a0(4) ? $a0 = 1 : $a0 = 0; 12 < 7 => $a0 = 0
mem(8)  := X"10800001"; --     beq  $a0, $0, ar1      # $a0(4)==0?ar1:PC = PC+4; 0==0 goto ar1
mem(9)  := X"20050000"; --     addi $a1, $0, 0        #
```

```
mem(10) := X"00e2202a"; --ar1: slt  $a0, $a3, $v0     # $a3(7)<$v0(2)?$a0(4)=1:$a0=0; 3<5,$a0=1
mem(11) := X"00853820"; --     add  $a3, $a0, $a1     # $a3(7)=$a0(4)+$a1(5); 1+11=12
mem(12) := X"00e23822"; --     sub  $a3, $a3, $v0     # $a3(7)=$a3(7)-$v0(2); 12-5=7
mem(13) := X"ac670044"; --     sw   $a3, 68($v1)      # $a3(7)->M[68+$v1(3)]; 7->M[68+12=80] #test 1
mem(14) := X"8c020050"; --     lw   $v0, 80($0)       # $v0(2) = M[80+0]; $v0 = 7
mem(15) := X"08000011"; --     j    end               # goto end
mem(16) := X"20020001"; --     addi $v0, $0, 1
mem(17) := X"ac02003C"; --end: sw   $v0, 60($0)       # $v0(2) write M[60]; M[60]=7; #test 2
```

Notations:

- `R[reg]` – Contenu du registre
- `imm` – Le champ `Immediate` de 16 bits d'instructions de type `I`
- `addr` – Le champ représentant une partie de l'adresse sur 26 bits de l'instruction de type `J`
- `SignImm` – la valeur `imm` étendue par le bit le plus significatif de `imm` représentant un signe sur 32-bits (concaténer 16 fois le bit à la position 15 de `imm` et la valeur immédiate sur 16 bits `imm`) = `{{16{imm[15]}}, imm};`
- `ZeroImm` – la valeur `imm` étendue par zéros sur 32-bits = `{16'bits 0, imm}` (concaténation 16 zéros avec la valeur immédiate `imm`)
- `Address` = R[`rs] + SignImm`
- `Mem[Address]` – contenu de la mémoire à l'adresse `Address`
- `BTA` - branch target address = `PC + 4 + (SignImm << 2)`
- `JTA` - jump target address = `{(PC + 4)[31:28], addr, 2'bits 0}`- concaténation de 4 bits les plus significatifs de 31 à 28 de la valeur `(PC + 4)` + le champs `addr` sur 26 bits + 2 bits zéros (positions 1 et 0)
- `Label` – le texte qui indique une localisation de l'instruction

1. **Solution schématique du processeur MIPS un cycle (30 pts)**

    a. **Composant chemin de données (10 pts)**

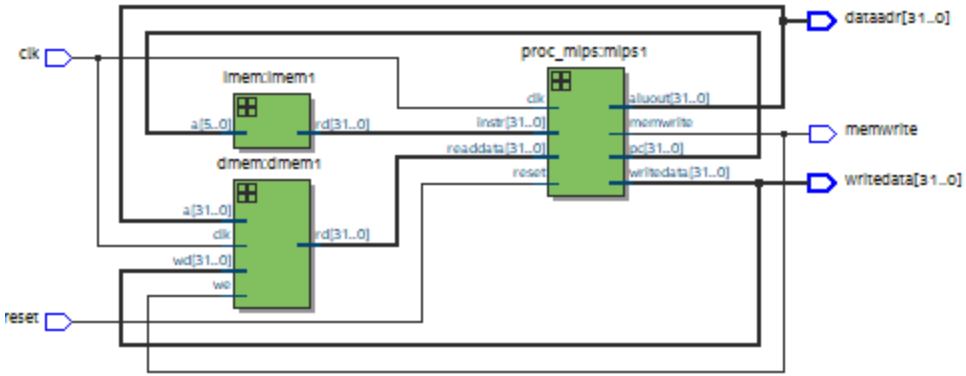
    - La solution schéma des modifications du chemin de données pour les instructions **`index2Adr`** et **`jal`**.
    - Les modifications nécessaires du chemin de données doivent être faites d'abord sur le schéma (inclure dans le rapport à remettre). Utilisez le schéma donné ou redessinez votre schéma.

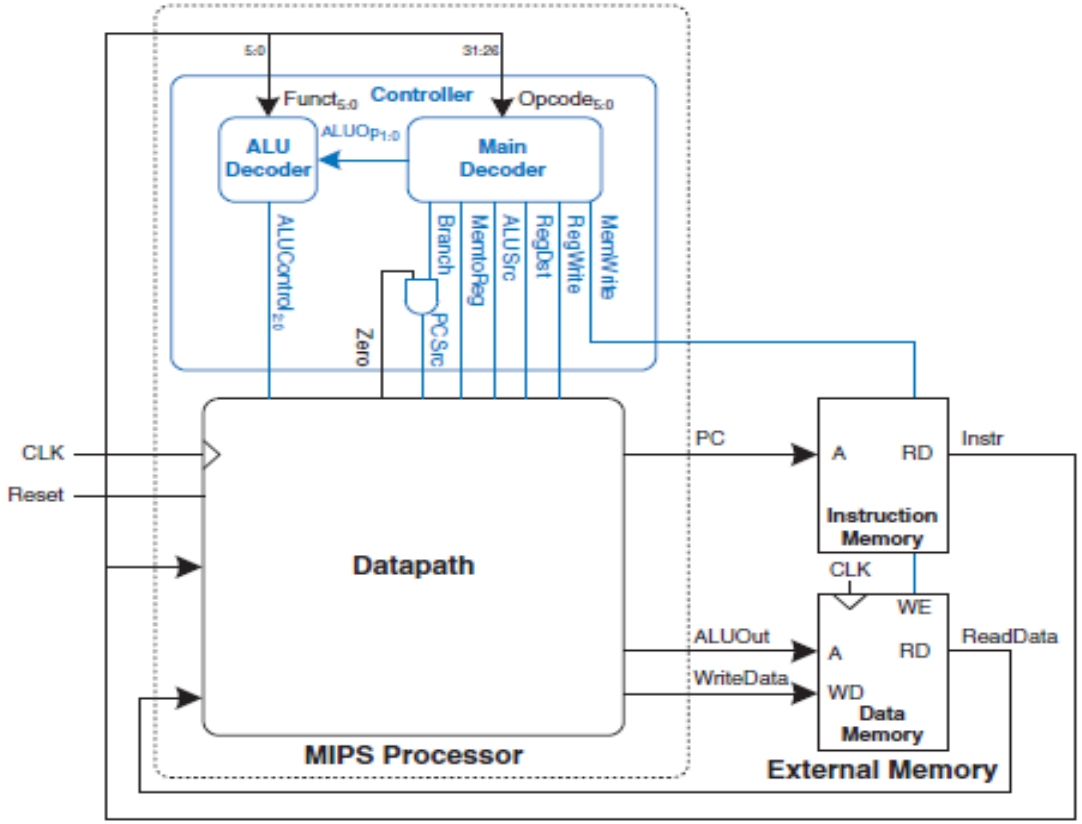
    b. **Pour la partie contrôleur (tableau à compléter) (20 pts) :**

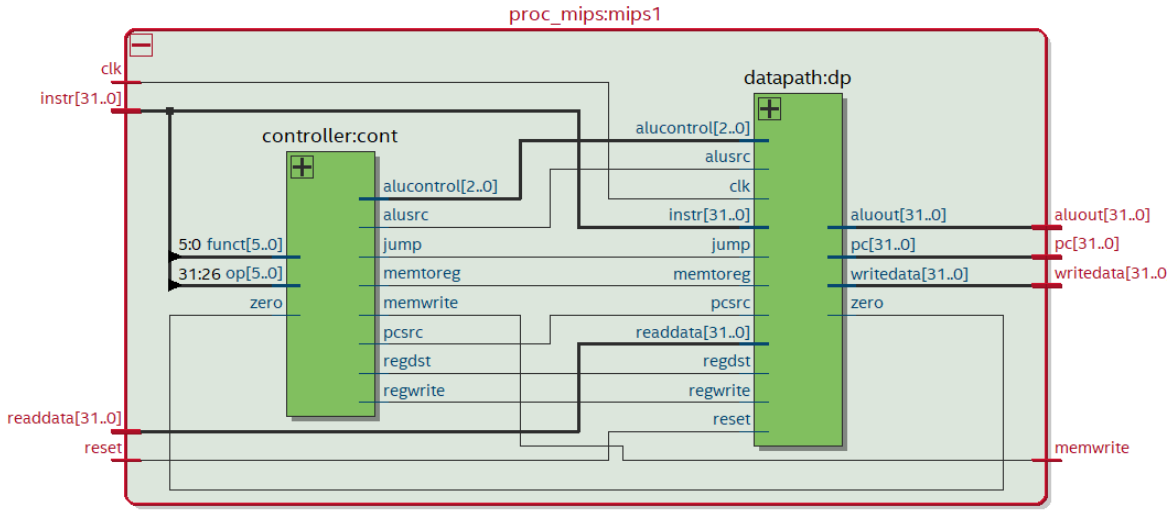
    Compléter la table de vérité de l'unité de contrôle pour incorporer les nouvelles instructions.

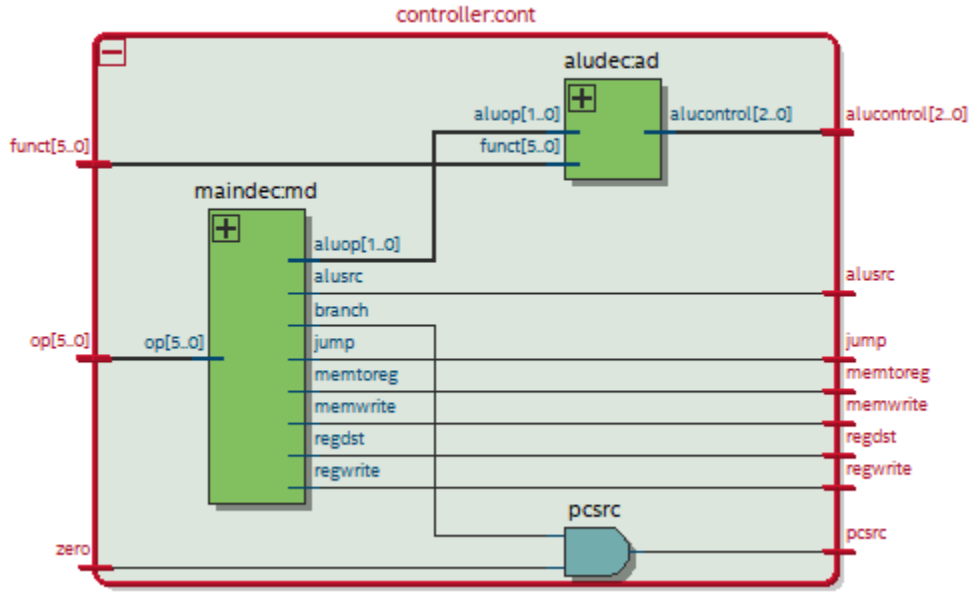
2. **Solution VHDL du processeur MIPS un cycle (30 pts)**

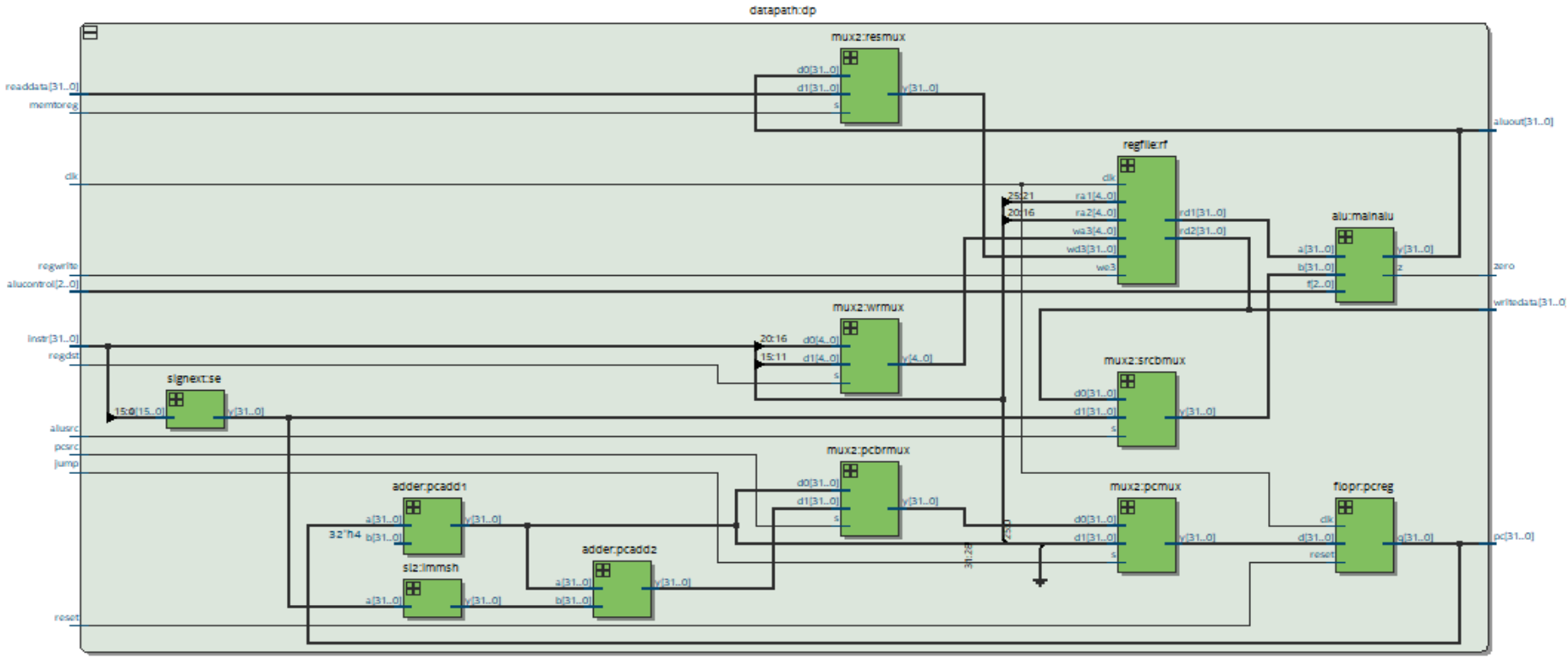
    a) (20 pts) Modifier le code VHDL du processeur MIPS initial (disponible sur le site du cours) pour incorporer les 2 instructions spécifiées plus haut. Le schéma structurel du code VHDL est présenté plus bas.


clk
reset
imem:imem1
a[5..0]
rd[31..0]
dmem:dmem1
a[31..0]
clk
wd[31..0]
we
rd[31..0]
proc_mips:mips1
clk
instr[31..0]
readdata[31..0]
reset
aluout[31..0]
memwrite
pc[31..0]
writedata[31..0]
dataadr[31..0]
memwrite
writedata[31..0]


5:0
31:26
Funct5:0
Controller
Opcode5:0
ALU Decoder
ALUOp1:0
Main Decoder
ALUControl2:0
Zero
Branch
PCSrc
MemtoReg
ALUSrc
RegDst
RegWrite
MemWrite
CLK
Reset
Datapath
MIPS Processor
PC
A
RD
Instr
Instruction Memory
CLK
WE
ALUOut
WriteData
A
RD
ReadData
WD
Data Memory
External Memory

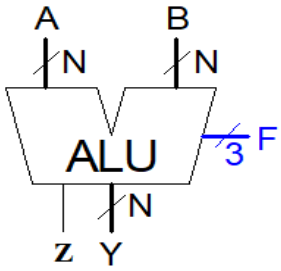

proc_mips:mips1
clk
instr[31..0]
controller:cont
alucontrol[2..0]
alusrc
jump
memtoreg
memwrite
pcsrc
regdst
regwrite
5:0 funct[5..0]
31:26 op[5..0]
zero
datapath:dp
alucontrol[2..0]
alusrc
clk
instr[31..0]
jump
memtoreg
pcsrc
readdata[31..0]
regdst
regwrite
reset
aluout[31..0]
pc[31..0]
writedata[31..0]
zero
readdata[31..0]
reset
aluout[31..0]
pc[31..0]
writedata[31..0
memwrite


controller:cont
aludec:ad
aluop[1..0]
funct[5..0]
alucontrol[2..0]
alucontrol[2..0]
funct[5..0]
maindec:md
aluop[1..0]
alusrc
branch
jump
memtoreg
memwrite
regdst
regwrite
alusrc
op[5..0]
op[5..0]
jump
memtoreg
memwrite
regdst
regwrite
pcsrc
zero
pcsrc



b) (10 pts) Modifier le code du `testbench` et le programme dans le module `imem` pour tester toutes les instructions ajoutées . Simuler votre processeur en observant le résultat d'exécution des instructions spécifiées plus haut.

ALU utilisée dans le devoir 4 :



$F_{2:0}$	Fonction
000	A & B
001	A | B
010	A + B
011	Non utilisé
100	A & ~B
101	A | ~B
110	A - B
111	SLT

```
entity alu is
   port (a, b: in STD_LOGIC_VECTOR(31 downto 0);
            f: in STD_LOGIC_VECTOR (2 downto 0);
            z: out STD_LOGIC;
            y: out STD_LOGIC_VECTOR(31 downto 0):= X"00000000");
end;
architecture behave of alu is
signal diff, tmp: STD_LOGIC_VECTOR(31 downto 0);
begin
      diff <= a - b;
      process (a, b, f, diff, tmp) begin
            case f is
                  when "000" => tmp <= a and b; --
                  when "001" => tmp <= a or b; --
                  when "010" => tmp <= a + b; --
                  when "100" => tmp <= a and (not b); --
                  when "101" => tmp <= a or (not b); --
                  when "110" => tmp <= diff; --
                  when "111" => --SLT
                        tmp <= X"0000000" & "000" & diff(31);
                  when others => tmp <= X"FFFFFFFF";
            end case;
      end process;
      z <= '1' when tmp = X"00000000" else '0';
      y <= tmp;
end;
```

3. **Solution schématique du processeur MIPS version multicycles (30 pts)**

   a. **Composant chemin de données (15 pts)**

   - La solution schéma des modifications du chemin de données pour les instructions **`index2Adr`** et **`jal.`**

   b. **Contrôleur (FSM) (15 pts) :**

   Incorporer les nouvelles instructions.

Barème

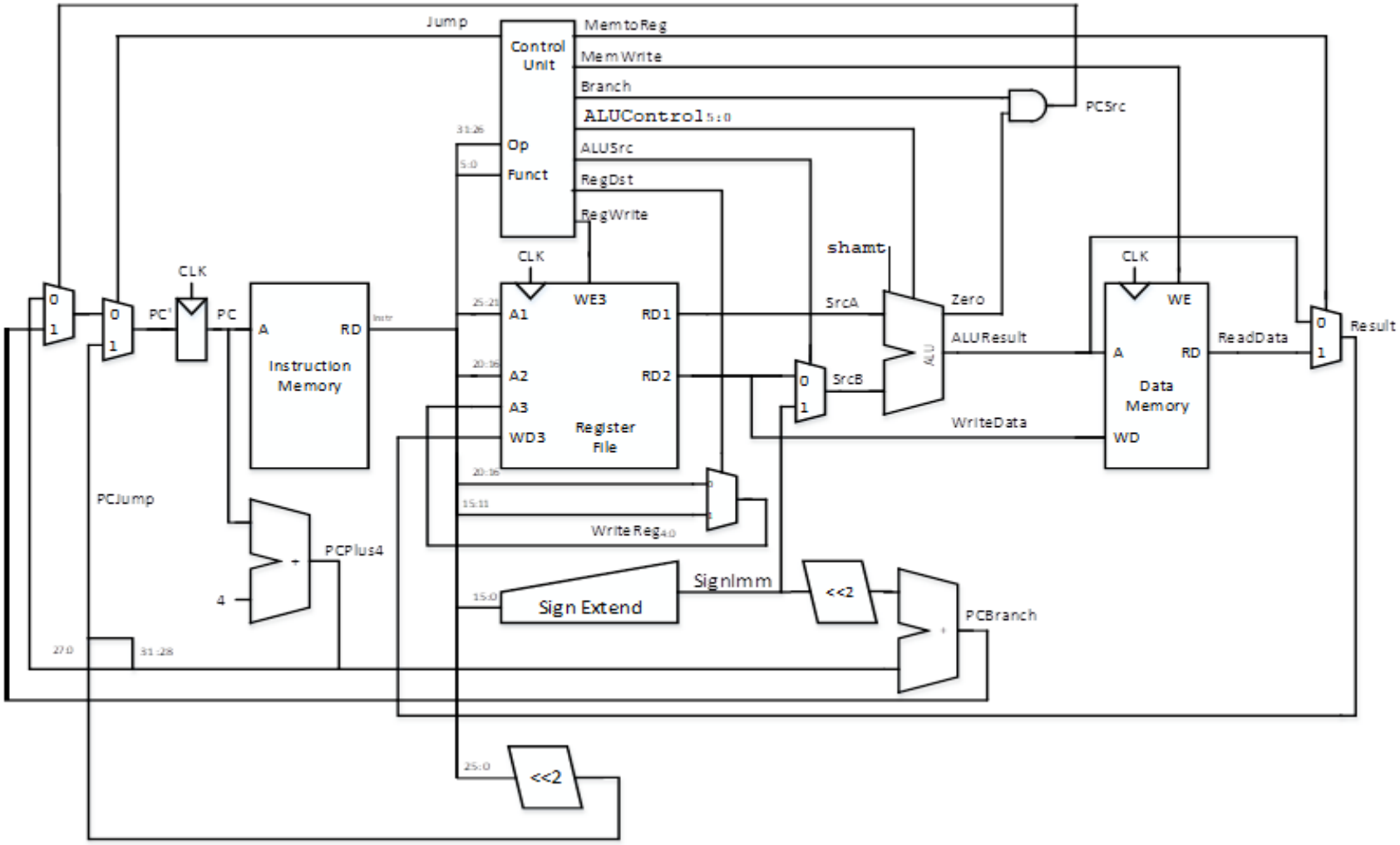
- Q1(30 pts), Q2(30 pts), Q3(30 pts), Présentation (10 pts)

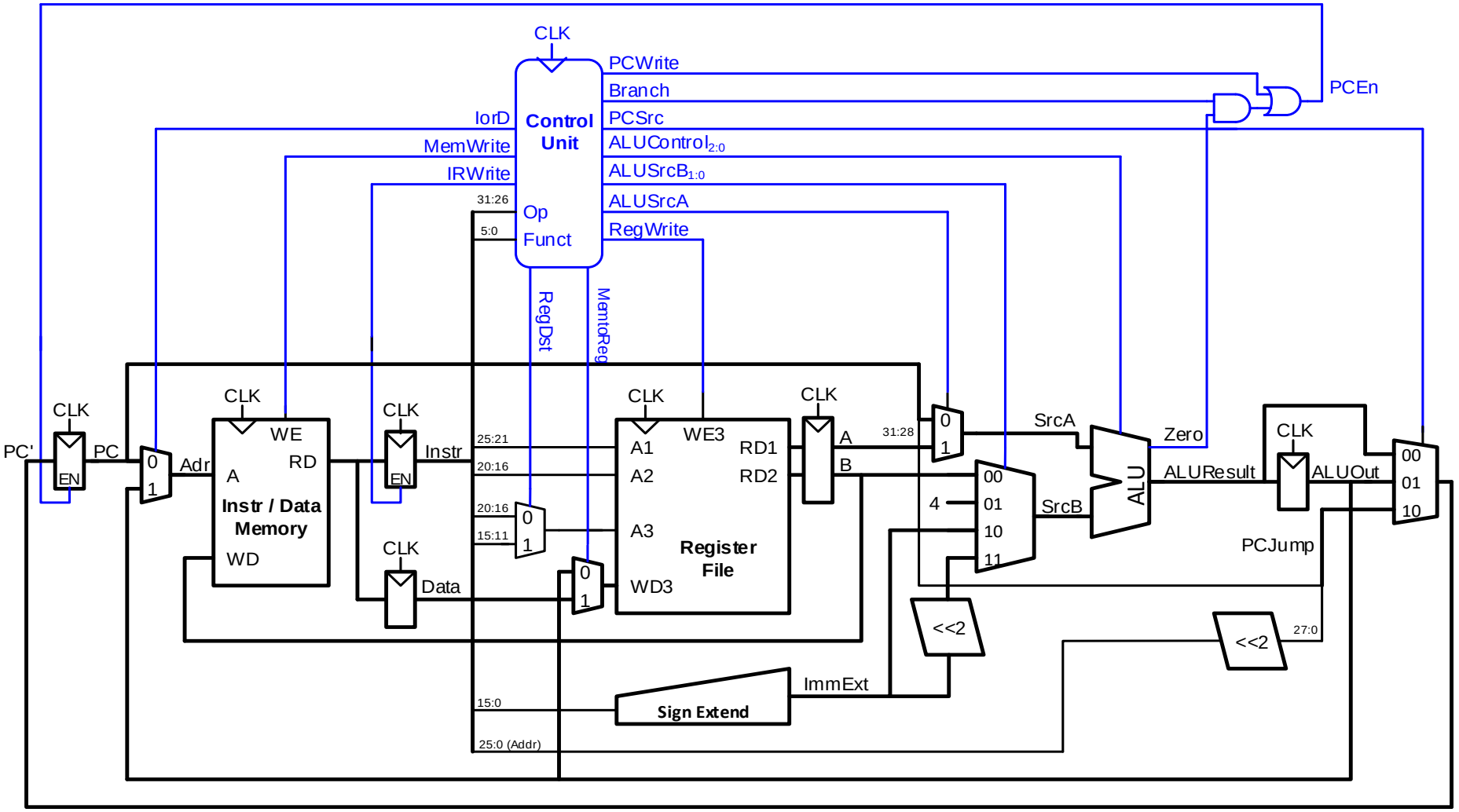
À remettre

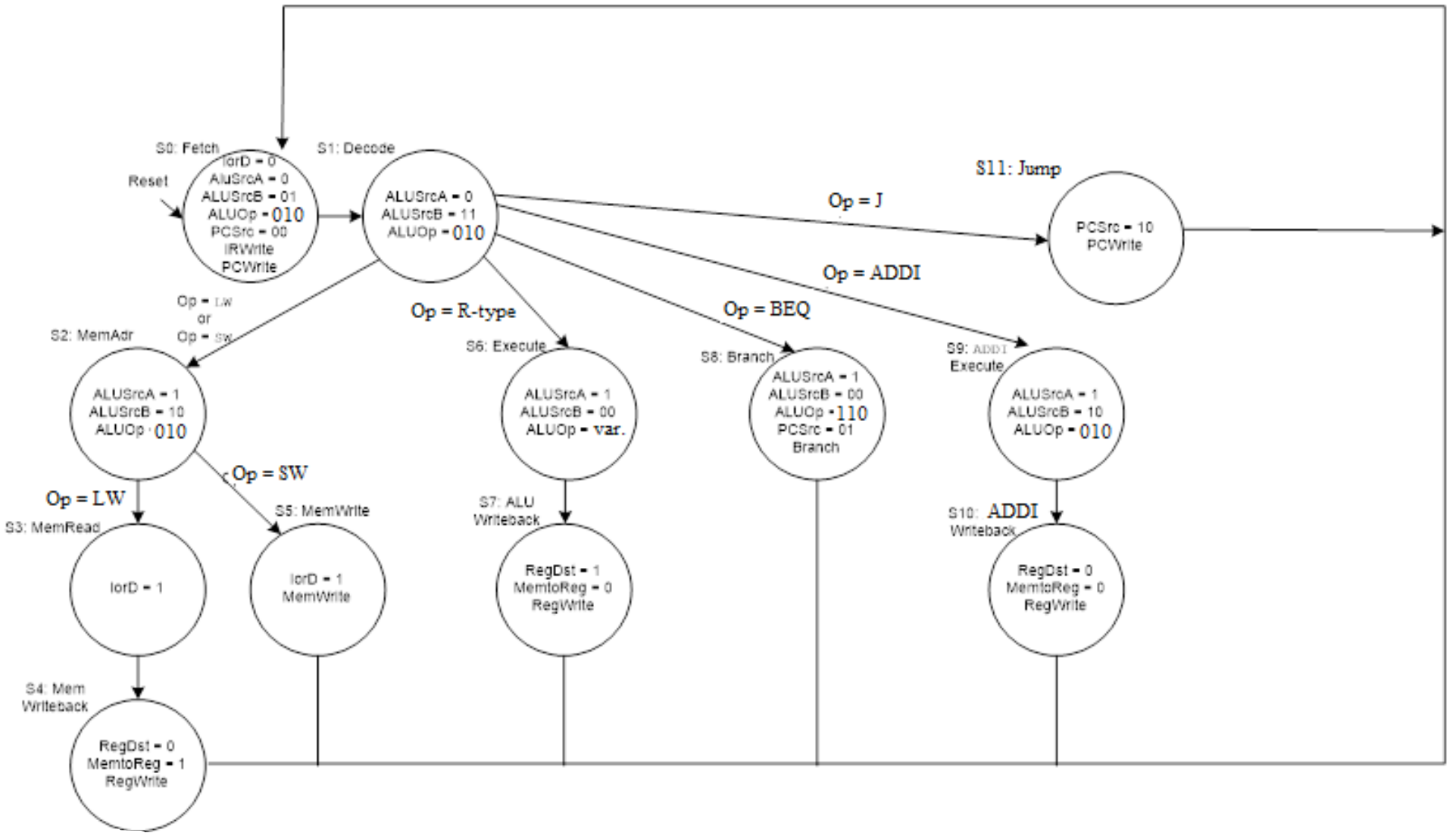
- Le rapport, format pdf : les 2 schémas, la table de vérité de l'unité de contrôle MIPS un cycle, FSM (MIPS multicycles), le programme test contenant le code assembleur et le code machine;

- Le projet doit contenir tous les fichiers sources VHDL dans le répertoire **`files`** pour qu'on puisse simuler et vérifier votre processeur.
- Compressez le tout dans un seul fichier .zip.

Instr	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUControl_{2:0}$	Jump			
R-type	000000	1	1	0	0	0	0	Diff.	0			
lw	100011	1	0	1	0	0	1	010(add)	0			
sw	101011	0	X	1	0	1	X	010(add)	0			
beq	000100	0	X	0	1	0	X	110(sub)	0			
j	000010	0	X	X	X	0	X	XXX	1			
addi	001000	1	0	1	0	0	0	010(add)	0			


Jump
Control Unit
MemtoReg
MemWrite
Branch
ALUControl5:0
ALUSrc
RegDst
RegWrite
Op
Funct
31:26
5:0
PCSrc
shamt
CLK
PC'
PC
A
RD
Instruction Memory
Instr
WE3
A1
A2
A3
WD3
RD1
RD2
Register File
SrcA
SrcB
ALU
Zero
ALUResult
WE
Data Memory
ReadData
WriteData
Result
PCJump
PCPlus4
4
WriteReg4:0
15:11
20:16
25:21
SignImm
Sign Extend
15:0
<<2
PCBranch
27:0
31:28
25:0


CLK
PCWrite
Branch
PCSrc
ALUControl2:0
ALUSrcB1:0
ALUSrcA
RegWrite
IorD
MemWrite
IRWrite
31:26
5:0
Op
Funct
Control Unit
RegDst
MemtoReg
PCEn
PC'
PC
EN
Adr
WE
RD
A
Instr / Data Memory
WD
Instr
Data
25:21
20:16
15:11
A1
A2
A3
WE3
WD3
RD1
RD2
Register File
A
B
31:28
SrcA
SrcB
ALU
Zero
ALUResult
ALUOut
4
00
01
10
11
<<2
27:0
PCJump
ImmExt
Sign Extend
15:0
25:0 (Addr)


S0: Fetch
IorD = 0
AluSrcA = 0
ALUSrcB = 01
ALUOp = 010
PCSrc = 00
IRWrite
PCWrite
Reset
S1: Decode
ALUSrcA = 0
ALUSrcB = 11
ALUOp = 010
S11: Jump
Op = J
PCSrc = 10
PCWrite
Op = ADDI
Op = BEQ
Op = R-type
Op = LW
or
Op = SW
S2: MemAdr
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 010
S6: Execute
ALUSrcA = 1
ALUSrcB = 00
ALUOp = var.
S8: Branch
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 110
PCSrc = 01
Branch
S9: ADDI Execute
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 010
Op = SW
Op = LW
S3: MemRead
IorD = 1
S5: MemWrite
IorD = 1
MemWrite
S7: ALU Writeback
RegDst = 1
MemtoReg = 0
RegWrite
S10: ADDI Writeback
RegDst = 0
MemtoReg = 0
RegWrite
S4: Mem Writeback
RegDst = 0
MemtoReg = 1
RegWrite