IFT 1227 – Architecture des ordinateurs

Devoir 4

- À faire en groupe de **deux** étudiants.
- Remise : Le **17 décembre** 2018 avant minuit.
- Il y aura une pénalité de **10%** par jour de retard

I. Extension d'un processeur MIPS version un cycle

Il s'agit d'ajouter une implémentation de quelques instructions au processeur MIPS version un cycle vu en cours. Le schéma donné représente le processeur MIPS version un cycle 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.

a) (8 pts) Modifier un module ALU comportemental en VHDL (fichier `alu.vhd`) pour qu'il implémente la table de vérité suivante :

$F_{5:0}$	Function
000000	B sll shamt
100000	A + B
100010	A - B
100100	A and B
100101	A or B
100110	A xor B
100111	A nor B
101011	SLTU (non signé)
101010	SLT (signé)

Avec la méthode de soustraction on implémente la version non signée SLT : SLTU.

b) (7 pts) Écrire un `testbench` auto vérificateur pour tester ALU sur les données suivantes : `shamt = "00001"` et `B = X "00000001"` pour une instruction `sll` et `A = X "80000000" B = X"7FFFFFFF"` pour toutes les autres.

c) (25 pts) Solution papier. Étendre l'implémentation du chemin de données et du contrôleur pour traiter les instructions **`jr, andi, jal, lui, ori, sll, sltu, slt`** et **`lb`** (convention big endian).

- Les modifications nécessaires du chemin de données doivent être faites d'abord sur papier (à remettre). Utilisez le schéma donné ou redessinez votre schéma.

- L'implémentation des instructions **`jr`**, **`andi`** sera montrée lors de séance de démonstration. L'intégration de ces instructions dans votre solution (papier + VHDL) sera notée (total - 10 pts).

- La solution papier des modifications du chemin de données pour les instructions **`jal`**, **`lui`**, **`ori`**, **`lb`** (4*3=12 pts), **`sll`** (1 pts), **`sltu`** (1 pts) et **`slt`** (1 pts).

Pour la partie contrôleur (solution papier):

- (25 pts) Compléter la table de vérité de l'unité de contrôle pour incorporer les nouvelles instructions.

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 une 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 et 2 bits zéros (positions 1 et 0)
- `Label` – le texte qui indique une localisation de l'instruction

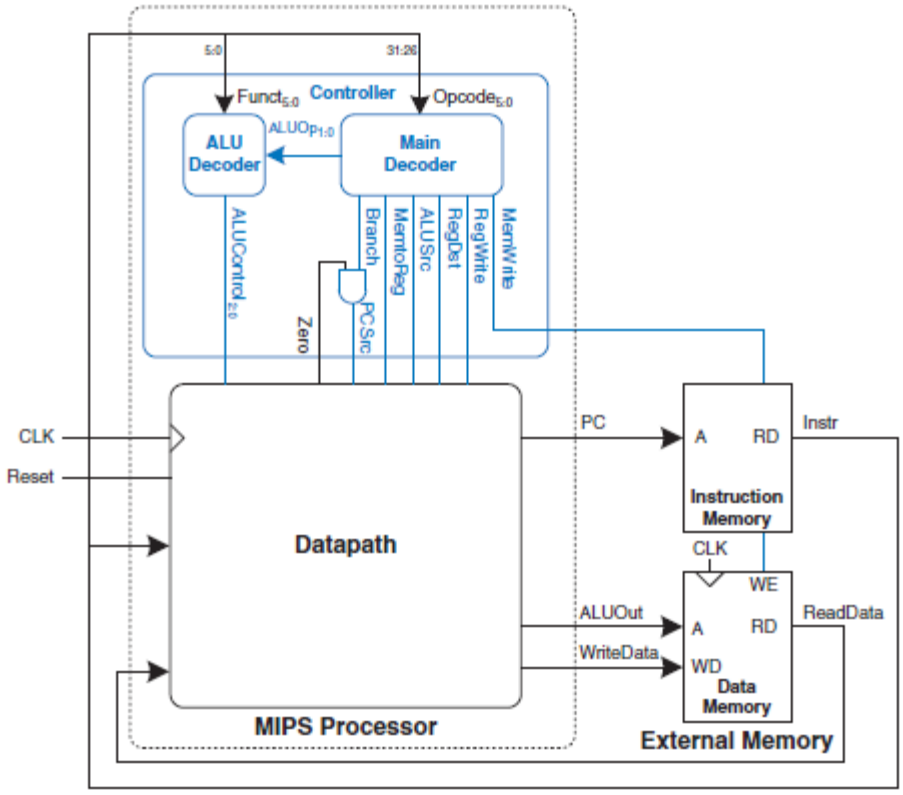
Les détails des instructions à implémenter :

Code d'opération	Nom	Description	Opération	funct
000000 (0)	`sll rd, rt, shamt`	Shift left logical	`R[rd] = R[rt] << shamt`	000000 (0)
000000 (0)	`slt rd,rs,rt`	Set Less Then	`R[rs] <R[rt]?R[rd]=1:R[rd]=0`	101010 (2A)
000000 (0)	`sltu rd,rs,rt`	Set Less Then Unsigned	`R[rs]<R[rt]?R[rd]=1:R[rd]=0`	101011 (2B)
000000 (0)	`jr rs`	Jump register	`PC = R[rs]`	001000 (08)
001100 (C)	`andi rt,rs,imm`	And Immediate	`R[rt] = R[rs] & ZeroImm`	-
000011 (3)	`jal label`	jump and link	`R[$ra] = PC + 4, PC = JTA`	-
001101(D)	`ori rt,rs,imm`	Or Immediate	`R[rt] = R[rs] | ZeroImm`	-
001111 (F)	`lui rt,rs,imm`	Load upper immediate	`R[rt] = {imm, 16'b0}`	-
100100 (36)	`lbu rt,imm(rs)`	Load byte unsigned (convention «little indian »)	`R[rt]={24'b0,M[Address](7:0)}`	-

Pour implémenter l'instruction `lb`, il faut ajouter deux modules MUX 4-1 et `ZeroExtM`. Le module de mémoire `dmem`, selon l'adresse d'un octet/mot fourni, retourne le mot mémoire qui contienne l'octet/mot demandé. Pour chercher un octet, il faudra utiliser un MUX 4-1 qui en utilisant les deux bits les moins significatifs (l'adresse d'un octet dans un mot : 0, 1, 2 ou 3) propagera en sortie cet octet. L'octet extrait devra être utilisé comme entrée du module `ZeroExtM` pour effectuer son extension non signée.

d) (25 pts) Modifier le code VHDL du processeur MIPS initial (disponible sur le site du cours) pour incorporer les neuf instructions spécifiées plus haut et 2 instructions présentées en classe (andi et jr). Remplacer le module **`alu`** initial par votre version de l'**`alu`** et adapter le code du module **`aludec.`**

Le schéma structurel du code VHDL est présenté plus bas.



e) (5 pts) Enlever les commentaires dans le code test en langage machine MIPS (module `imem`) et dans le code de testbench pour pouvoir tester les nouvelles instructions.

Simuler votre processeur en observant le résultat d'exécution des instructions spécifiées plus haut.

Présentation : 5 pts

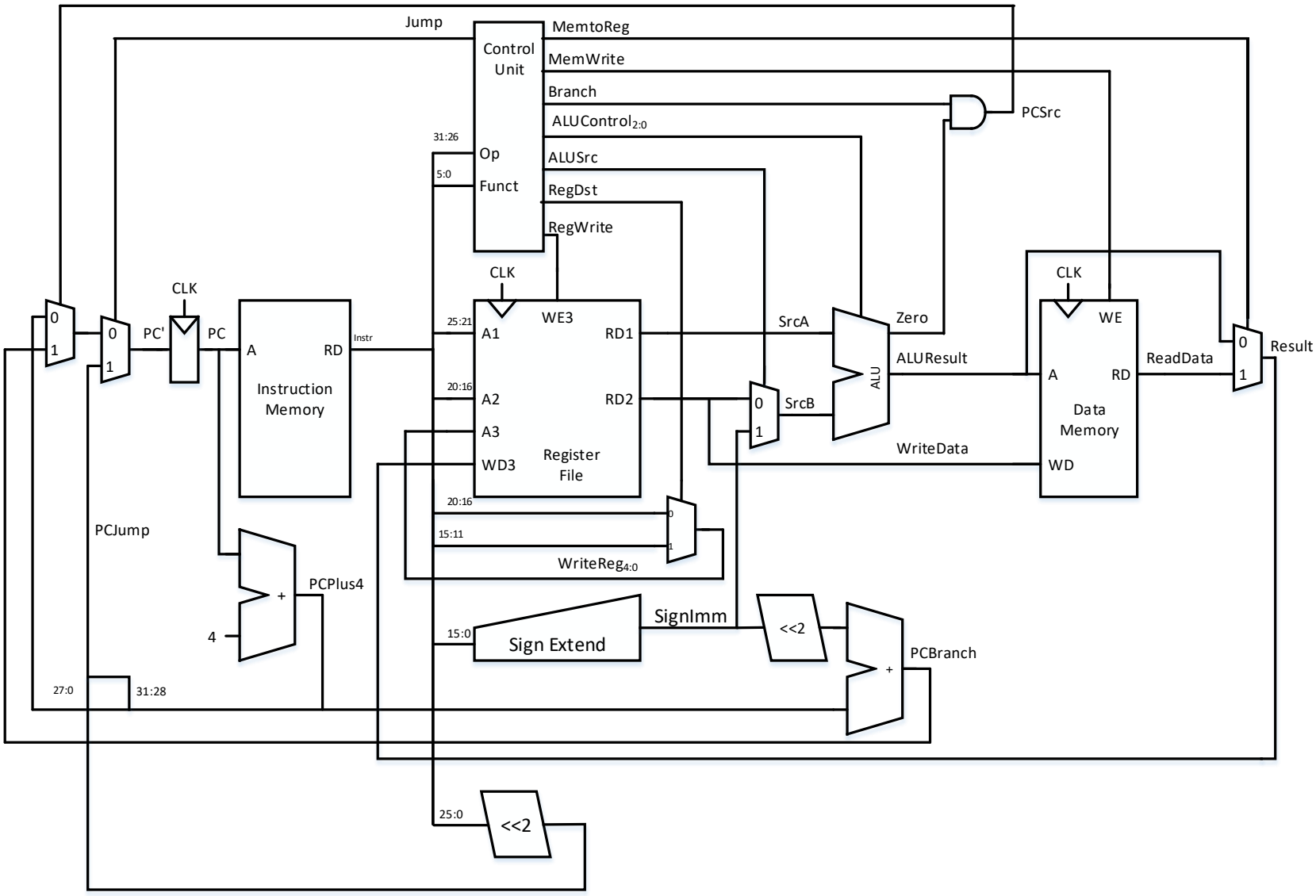
<u>À remettre.</u>

- Le rapport : le schéma, la table de vérité de l'unité de contrôle, le programme test contenant le code assembleur et le code machine (voir page 437 (428) de votre livre pour un exemple).

  Pour la remise papier du rapport: scanner les copies papier et faire la remise électronique des documents scannés par StudiUM ou faites la remise papier au démonstrateur/professeur.

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

Instr	**$Op_{5:0}$**	**RegWrite**	**RegDst**	**AluSrc**	**Branch**	**MemWrite**	**MemtoReg**	**$F_{5:0}$**	**Jump**			
R-type	000000	1	1	0	0	0	0	Voir tab	0			
lw	100011	1	0	1	0	0	1	100000	0			
sw	101011	0	X	1	0	1	X	100000	0			
beq	000100	0	X	0	1	0	X	100010	0			
j	000010	0	X	X	X	0	X	XX	1			
addi	001000	1	0	1	0	0	0	100000	0			


Jump
Control Unit
MemtoReg
MemWrite
Branch
ALUControl2:0
ALUSrc
RegDst
RegWrite
PCSrc
31:26
5:0
Op
Funct
CLK
PC'
PC
A
RD
Instr
Instruction Memory
25:21
20:16
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
SrcA
SrcB
ALU
Zero
ALUResult
WE
ReadData
Data Memory
WriteData
WD
Result
PCJump
PCPlus4
4
20:16
15:11
WriteReg4:0
15:0
Sign Extend
SignImm
<<2
PCBranch
27:0
31:28
25:0
<<2