Devoir 4

Processeur MIPS

Etienne Collin | 20237904
Ange Lilian Tchomtchoua Tokam | 20230129
Justin Villeneuve | 20132792

Architecture des ordinateurs - IFT1227

Section A

Professeure Alena Tsikhanovich

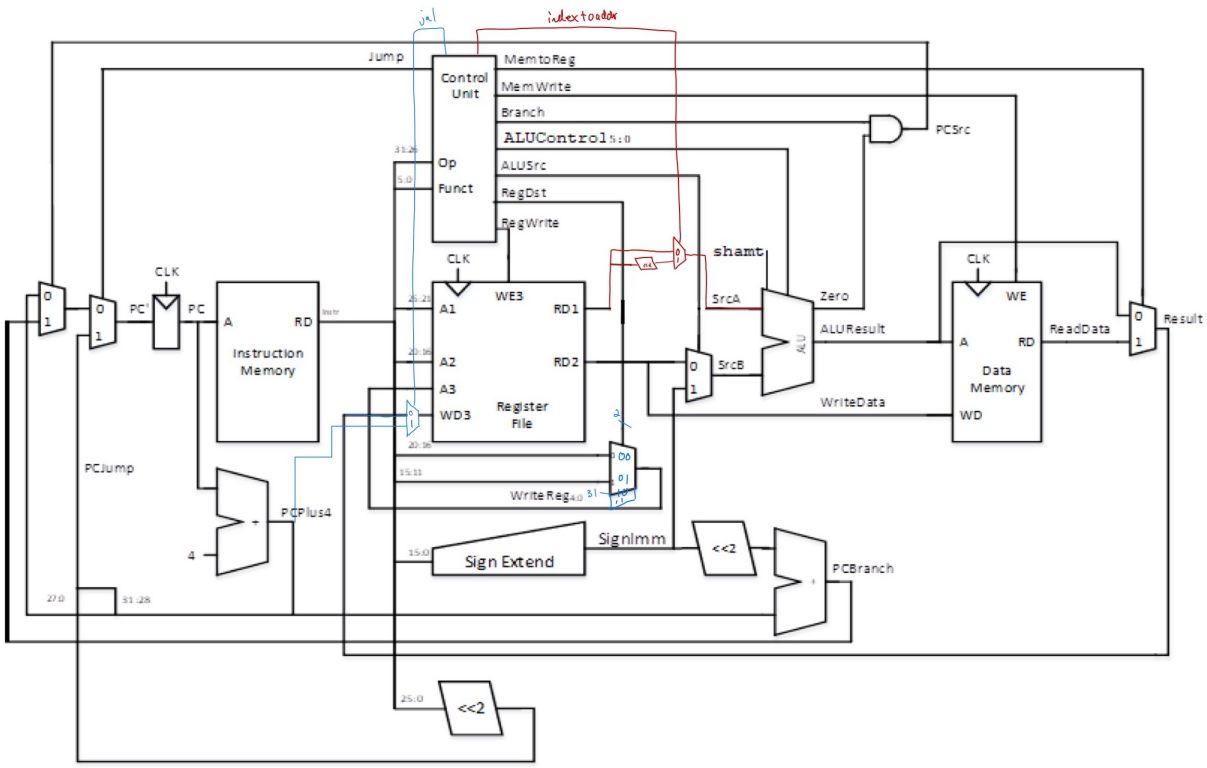
Université de Montréal

À remettre le 30 Avril 2023 à 23:59

Table des matières

1 Solution schématique du processeur MIPS un cycle

1.1 Chemin de données



1.2 Contrôleur

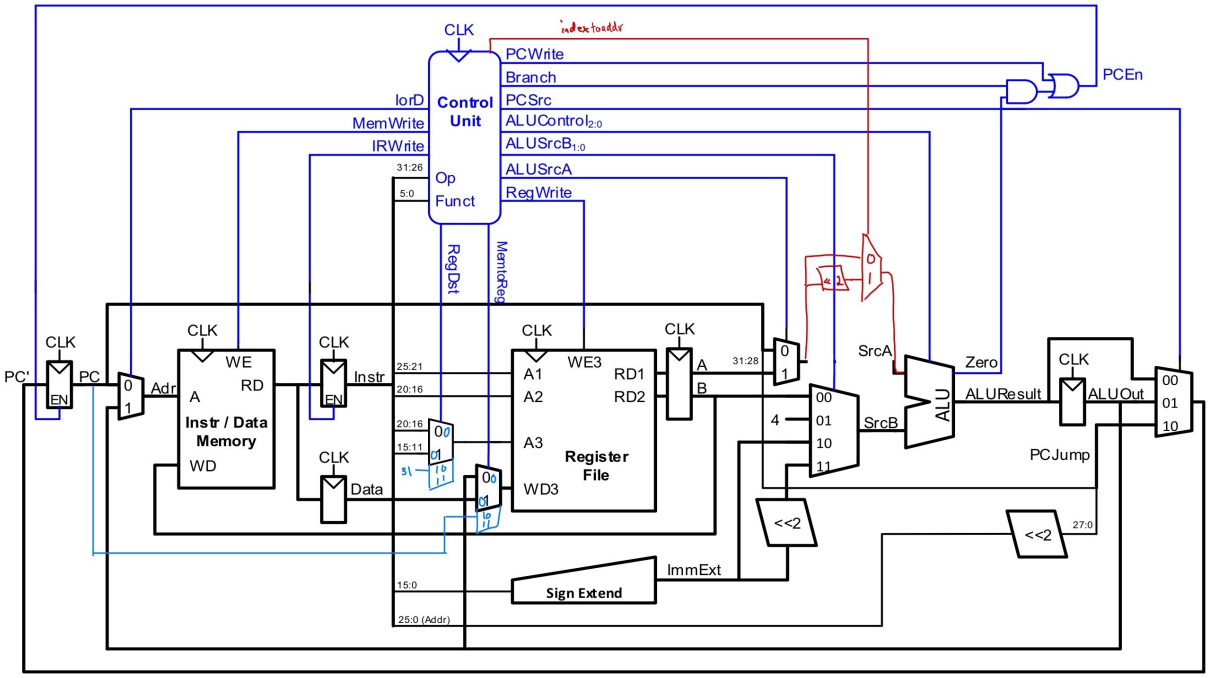
Instr	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUControl_{2:0}$	Jump	indextoaddr	jal	
R-type	000000	1 0	1	0	0	0	0	Diff.	0	0	0	
lw	100011	1 0	0	1	0	0	1	010 (add)	0	0	0	
sw	101011	0 X	X	1	0	1	X	010 (add)	0	0	0	
beq	000100	0 X	X	0	1	0	X	110 (sub)	0	0	X	
j	000010	0 X	X	X	X	0	X	XXX	1	X	X	
addi	001000	1 0	0	1	0	0	0	010 (add)	0	0	0	
index2Adr	000 000	1	01	0	0	0	0	010 (add)	0	1	0	
jal	000011	1	10	X	X	0	X	XXX	1	X	1	

index2adr
jal

```
R[rd] = R[rs]*4 + R[rt]
R[$ra] = PC + 4, PC = JTA
```

2 Solution schématique du processeur MIPS multicycles

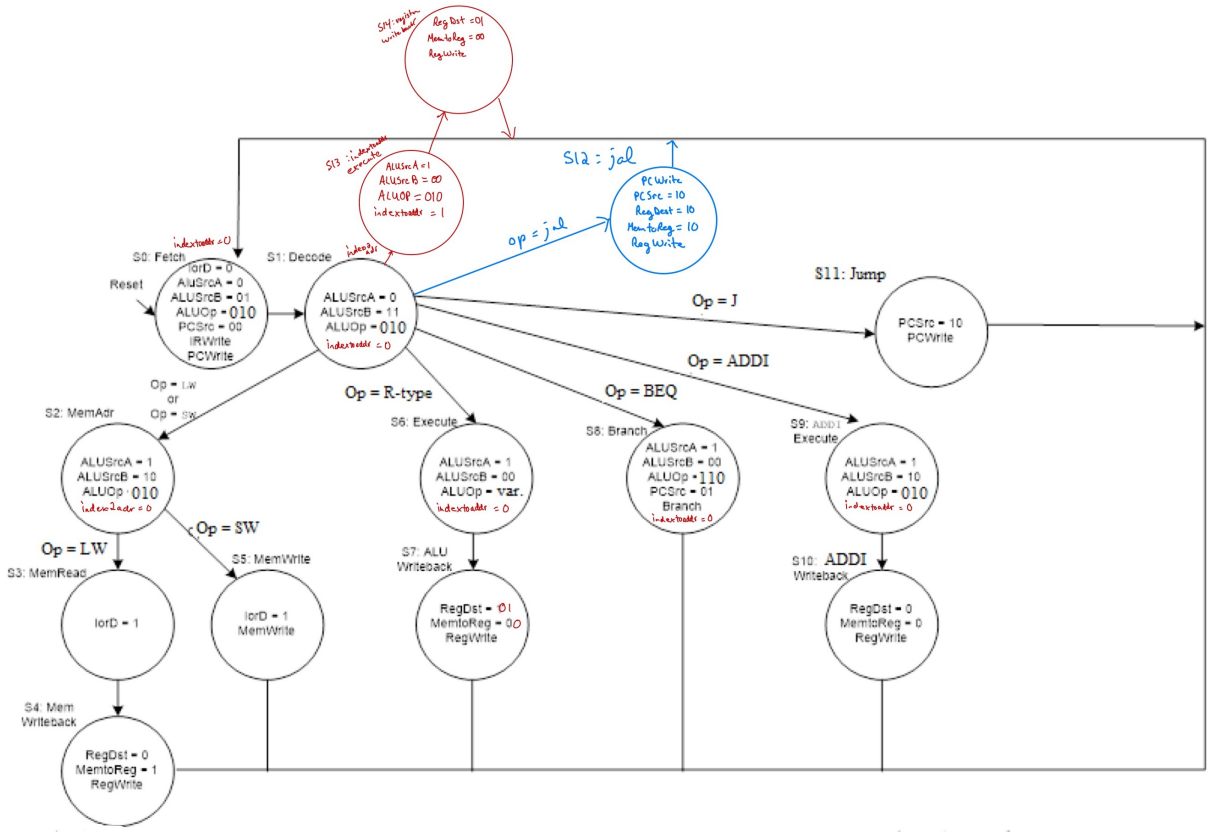
2.1 Chemin de données



indexToAddr
jal

```
R[rd] = R[rs]*4 + R[rt]
R[$ra] = PC + 4, PC = JTA
```

2.2 Contrôleur



3 Code Testbench

3.1 MIPS Assembly

Voici le code qui a été utilisé pour le testbench écrit en MIPS Assembly :

```
        addi        $v0,    $0,         5       # $v0(2)=5
        addi        $v1,    $0,         12      # $v1(3)=12
        addi        $a3,    $v1,        -9      # $a3(7)=$v1(3)(12)-9=3
        or          $a0,    $a3,        $v0     # $a0(4)=$a3(7) or $v0(2)=3 or 5=7
        and         $a1,    $v1,        $a0     # $a1(5)=$v1(3) and $a0(4)= 12 and 7=4
        add         $a1,    $a1,        $a0     # $a1(5)=$a1(5)+$a0(4)=4+7=11
        beq         $a1,    $a3,        end     # $a1(5)==$a3(7) ? end: PC=PC+4; 11==3 ? PC=PC+4
        slt         $a0,    $v1,        $a0     # $v1(3) < $a0(4) ? $a0=1 : $a0=0; 12 < 7 => $a0=0
        beq         $a0,    $0,         ar1     # $a0(4)==0 ? ar1 : PC=PC+4; 0==0 goto ar1
        addi        $a1,    $0,         0       #
ar1:    slt         $a0,    $a3,        $v0     # $a3(7) < $v0(2) ? $a0(4)=1 : $a0=0; 3 < 5, $a0=1
        add         $a3,    $a0,        $a1     # $a3(7)=$a0(4)+$a1(5); 1+11=12
        sub         $a3,    $a3,        $v0     # $a3(7)=$a3(7)-$v0(2); 12-5=7
        sw          $a3,    68($v1)             # $a3(7) -> M[68+$v1(3)]; 7 -> M[68+12=80]              # Test 1
        lw          $v0,    80($0)              # $v0(2)=M[80+0]; $v0=7
        j           end                         # goto end
        addi        $v0,    $0,         1
end:    sw          $v0,    60($0)              # $v0(2) write M[60]; M[60]=7;                          # Test 2
        jal         tag                         # Jump to testJal
        sw          $v0,    40($0)              # This should not be executed if jump is ok
tag:    sw          $ra,    40($0)              # $ra=19 -> M[40]                                       # Test 3
        addi        $t0,    $0,         3       # $t0=3
        addi        $t1,    $0,         5       # $t1=5
        index2Adr   $t0,    $t0,        $t1     # $t0=4*$t0+$t1=4*3+5=17
        sw          $t0,    20($0)              # $t0=17 -> M[20]                                       # Test 4
```

3.2 MIPS Hexadécimal

Voici le code précédent traduit en hexadécimal :

```
0x20020005
0x2003000c
0x2067fff7
0x00e22025
0x00642824
0x00a42820
0x10a7000a
0x0064202a
0x10800001
0x20050000
0x00e2202a
0x00853820
0x00e23822
0xac670044
0x8c020050
0x08000011
0x20020001
0xac02003C
0x0C000014
0xAC020028
0xAC1F0028
0x20080003
0x20090005
0x0109403F
0xAC080014
```