Design Sheet

1. Team Members

Team member: Nguyen Tran

2. Design Overview

For this lab, I design and simulate a 16-bit single cycle RISC processor. Since, the processor only have 16-bits which is 2^4. Therefore, the opcode of every instruction is 4 bits. For R-format, I divided evenly into 4 bits per registers. Therefore R-format does not have shift and function. Like R-format, I-format have 4 bits for opcode. Then I-format uses 4 bits for each register and the last 4-bit is for immediate. For j-format, it has 4 bits for opcode and the rest 12 bits is pseudo-address. Since, I will design a pipeline Datapath which each stage will forward to other stage after clock cycle. Therefore, I will use falling edge triggered my clock. For register file, I will have two 4 bits read register, two 4 bits read data for output, and 4 bits write register input. I also have writeReg signal, which input from control files to write data signal to register when clock is falling. I also use 16 registers to store data which each register can store up to 16 bits. In my design, I created three new instructions which are shift left immediate, two complements, and multiply Immediate.

2.1. Instruction format

Example:

R-Format:

Opcode	rs	rt	rd
4 bits	4 bits	4 bits	4 bits

I–Format:

Opcode	Rs	Rt	IMM
4 bits	4 bits	4 bits	4 bits

J- Format:

Opcode	Pseudo-address
4 bits	12 bits

2.2. Instructions

Name	Mnemonic	Operation	Op-	Format
No function	nop		0000	
Add immediate	addi	addi \$s1, \$s2, 100; \$s1 = \$s2 + 100	0001	I
ADD	add	add \$s1, \$s2. \$s3; \$s1 = \$s2 + \$s3	0010	R
Sub	sub	sub \$s1,\$s2,\$s3; \$s1 = \$s2 - \$s3	0011	R
Bitwise XOR	xor	xor \$s1, \$s2, \$s3; \$s1=\$s2 and \$s3	0100	R
Load word	lw	lw \$s1, IMM(\$s2); \$s1=[\$s2] + IMM	0101	I
Store word	sw	sw \$s1, IMM(\$s2); [\$s2] + IMM = \$s1	0110	I

Jump	j	j address; PC = {(PC + 4)[15:11], address, 00}	0111	J
Load immediate	li	li $s1, IMM; $s1 = IMM	1000	I
Bitwise or	or	or $s1, $s2,$s3; $s1 = $s2 | &s3	1001	R
set less than	slt	slt $at,$s2,$s3; if($s1<$s2) $at = 1	1010	R
Branch not equal	bne	bne $at, $zero, addr; $ at != 1 then j addr	1011	I
Branch equal	beq	beq $at, $zero, addr; $ at != 1 then j addr	1100	I
Division immediate	divi	div $s1, $s2, 2; $s1= $s2/2	1101	I
Shift left immediate	slli	slli $s1, $s2, IMM; $s1 = $s2 << IMM	1110	I
Multiply Immediate	muli	muli $s1, $s2, IMM; $s1= $s2 * IMM	1111	I

2.3. Assembly language and machine code for the test program (Pseudocode)

Pseudocode for the test program (refer to the project handout):

```
$v0 = 0040hex; // you can redefine $v0-3, $t0, and $a0-1 with
$v1 = 1010hex; // your register numbers such as $1, $2, etc.
$v2 = 000Fhex;
$v3 = 00F0hex;
$t0 = 0000hex;
$a0 = 0010hex;
$a1 = 0005hex;
while ($a1 > 0) do {
$a1 = $a1 -1;
$t0 = Mem[$a0];
if ($t0 > 0100hex) then {
$v0 = $v0 ÷ 8;
$v1 = $v1 | $v0; //or
Mem[$a0] = FF00hex;
}
else {
$v2 = $v2 × 4;
$v3 = $v3 ⊕ $v2; //xor
Mem[$a0] = 00FFhex;
}
$a0 = $a0 + 2;
}
return;
```

Name	$zero	$at	$v0	$v1	$v2	$v3	$a0	$a1	$t0	$t1	$t2	$t3	$t4	$t5	$t6	$t7
Number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Assembly Language	Machine Code
addi $2, $0, 0010	0000: 0001 0000 0010 0010
slli $2, $2, 4	0004: 1110 0010 0010 0100
Addi $3, $0, 0001	0008: 0001 0000 0011 0001
slli $3, $3, 4	000C: 1110 0011 0011 0100
Slli $3, $3, 4	0010: 1110 0011 0011 0100
Addi $3, $3,0001	0014: 0001 0011 0011 0001
Slli $3, $3, 4	0018: 1110 0011 0011 0100
Addi $4, $0, 0111	001C: 0001 0100 0000 0111
Slli $4,$4,0001	0020: 1110 0100 0100 0001
Addi $4,$4,0001	0024: 0001 0100 0100 0001
slli $5, $4, 4	001C: 1110 0101 0100 0100
addi $8, $0, 0000	0020: 0001 1000 0000 0000
addi $6, $0, 0001	0024: 0001 0110 0000 0001
Slli $6, $6, 4	0028: 1110 0110 0110 0100
Addi $7, $0, 0101	002C: 0001 0111 0000 0101
While: slt $1, $0, $7	0030: 1010 0000 0111 0001
bne $1, $0, 0001	0034: 1011 0001 0000 0001
j return	0038: 0111 0000 0001 0111
Do: addi $9, $0, 0001	003C: 0001 1001 0000 0001
sub $7, $7, $9	0040: 1101 0111 1001 0111
lw $8, 0[$6]	0044: 0101 1000 0110 0000
addi $9, $0, 0001	0048: 0001 1001 0000 0001
slli $9, $9, 8	004C: 1110 1001 1001 1000
slt $1, $8, $9	0050: 1010 1001 1000 0001
bne $1, $0, 0110	0054: 1011 0001 0000 0110
muli $4, $4, 0100	0058: 1111 0100 0100 0100
xor $5, $5, $2	005C: 0100 0101 0010 0101
addi $9, $0, 1111	0060: 0001 1001 0000 1111
slli $9, $9, 4	0064: 1110 1001 1001 0100
addi $9, $9, 1111	0068: 0001 1001 1001 1111
sw $9, 0[$6]	006C: 0110 1001 0110 0000
Endif: addi $9, $9, 1000	0070: 0001 1001 1001 1000
div $2, $2, $9	0074: 1101 0010 0010 1001
or $3, $3, $2	0078: 1001 0011 0010 0011

addi $9, $0, 1111	007C: 0001 1001 0000 1111
slli $9, $9, 4	0080: 1110 1001 1001 0100
addi $9, $9, 1111	0084: 0001 1001 1001 1111
slli $9, $9, 8	0088: 1110 1001 1001 1000
sw $9, 0[$6]	008C: 0110 1001 0110 0000
addi $6, $6, 0010	0090: 0001 0110 0110 0010
j While	0094: 0111 1111 1110 0110
return	0098: 0000 0000 0000 0000

** You can explain your assembly language in brief if needed.