Overview

This report introduce the instruction set architecture of a microprocessor. The instructions include :

- R-Type and I-Type ALU Instructions
- R-Type Multiplication / Division Instructions
- R-Type Shifter Instructions
- R-Type and J-Type Jump Instructions
- Type I and Type II Branch Instructions
- Memory Access Instructions

The dataword width is designed as **4-bit** address inputs as the architecture is 16-bit MIPS processor. It is calculated as $2^4 = 16$ bit where the number 4 is the dataword width.

The number of registers is **16**, as for any R-type instruction 12 bits are spared for *rs*, *rt*, and *rd*. That makes each of the registers is storing 4-bit wide data. $2^4 = 16$ bit hence 16 registers are needed in this architecture.

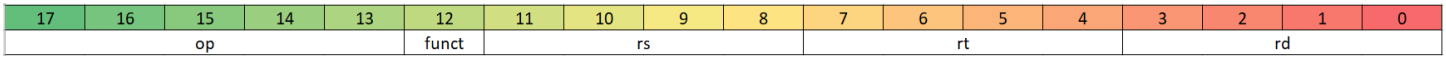

Figure 1: Bit Field Classifciation

Priority Encoding and Decoding

Instruction Type	Opcode
J-Type Jump	00000
Type-1 Branch	00001
Type-2 Branch	0001x
Memory access	001xx
I-Type	01xxx
R-Type	1xxxx

R Type	Opcode
Excep	000
Shift	001
Mult & Div	01x
ALU	1xx

Table 1: Instruction Type Priority Table (Left) & R Type Priority Table (Right)

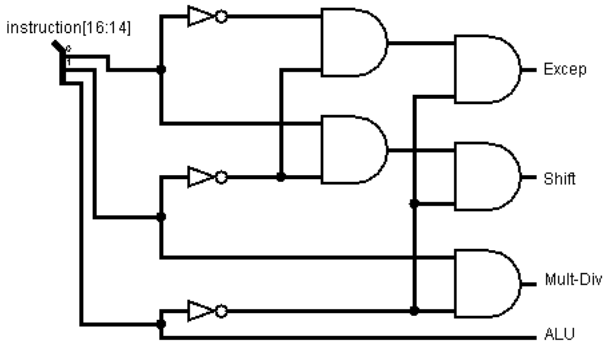

Figure 2: R-Type Instructions Priority

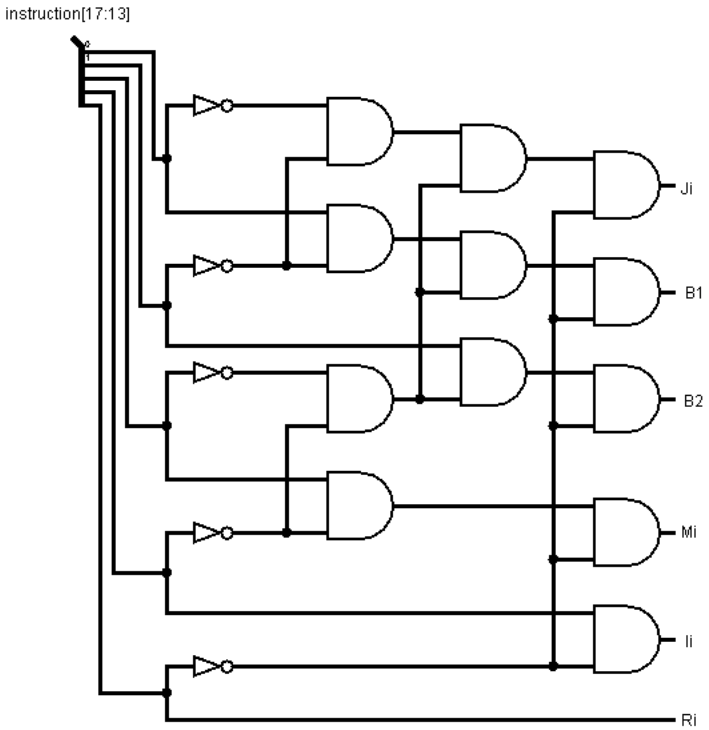

Figure 3: Instruction Types Priority

Instructions

R-TYPE ALU Instructions																	
	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	op	funct[4]	funct[3]	funct[2]	funct[1]	funct[0]	rs[3]	rs[2]	rs[1]	rs[0]	rt[3]	rt[2]	rt[1]	rt[0]	rd[3]	rd[2]	rd[1]	rd[0]
add	1	1	0	0	0	0	Src Reg 2				Src Reg 1				Dest Reg			
addu	1	1	0	0	0	1												
sub	1	1	0	0	1	0												
subu	1	1	0	0	1	1												
and	1	1	0	1	0	0												
or	1	1	0	1	0	1												
xor	1	1	0	1	1	0												
nor	1	1	0	1	1	1												
slt	1	1	1	0	1	0												
sltu	1	1	1	0	1	1												

Figure 4: R-Type ALU Instructions

R-TYPE MUL/DIV Instructions																		
	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	op[1]	op[0]	funct[3]	funct[2]	funct[1]	funct[0]	rs[3]	rs[2]	rs[1]	rs[0]	rt[3]	rt[2]	rt[1]	rt[0]	rd[3]	rd[2]	rd[1]	rd[0]
mfhi	1	0	1	0	0	0	Src Reg 2				Src Reg 1				Dest Reg			
mthi	1	0	1	0	0	1												
mflo	1	0	1	0	1	0												
mtlo	1	0	1	0	1	1												
mult	1	0	1	1	0	0												
div	1	0	1	1	1	0												

Figure 5: R-Type Multiplier and Divider Instructions

R-TYPE Shifter Instructions																		
	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	op[3]	op[2]	op[1]	op[0]	funct[1]	funct[0]	rs[3]	rs[2]	rs[1]	rs[0]	rt[3]	rt[2]	rt[1]	rt[0]	rd[3]	rd[2]	rd[1]	rd[0]
sllv	1	0	0	1	0	0	Src Reg 2				Src Reg 1				Dest Reg			
srlv	1	0	0	1	0	1												
srav	1	0	0	1	1	1												

Figure 6: R-Type Shifter Instructions

R-TYPE Jump Instructions																		
	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	op[4]	op[3]	op[2]	op[1]	op[0]	funct[0]	rs[3]	rs[2]	rs[1]	rs[0]	rt[3]	rt[2]	rt[1]	rt[0]	rd[3]	rd[2]	rd[1]	rd[0]
jr	1	0	0	0	0	0					Src Reg 1							
jalr	1	0	0	0	0	1												

Figure 7: R-Type Jump Instructions

I-TYPE ALU Instructions																		
	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	op[1]	op[0]	funct[3]	funct[2]	funct[1]	funct[0]	imm[3]	imm[2]	imm[1]	imm[0]	rt[3]	rt[2]	rt[1]	rt[0]	rd[3]	rd[2]	rd[1]	rd[0]
addi	0	1	0	0	0	0	Imm				Src Reg 1				Dest Reg			
addiu	0	1	0	0	0	1												
slti	0	1	1	0	1	0												
sltiu	0	1	1	0	1	1												
andi	0	1	0	1	0	0												
ori	0	1	0	1	0	1												
xori	0	1	0	1	1	0												
lui	0	1	0	1	1	1												

Figure 8: I-Type ALU Instructions

Memory Access Instructions																		
	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	op[2]	op[1]	op[0]	funct[2]	funct[1]	funct[0]	imm[3]	imm[2]	imm[1]	imm[0]	rt[3]	rt[2]	rt[1]	rt[0]	rd[3]	rd[2]	rd[1]	rd[0]
lb	0	0	1	0	0	0	Imm				Src Reg 1				Dest Reg			
lw	0	0	1	0	0	1												
lbu	0	0	1	0	1	0												
sb	0	0	1	1	0	0												
sw	0	0	1	1	0	1												

Figure 9: Memory Access Instructions

Type-2 Branch Instruction																		
	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	op[3]	op[2]	op[1]	op[0]	funct[1]	funct[0]	rs[3]	rs[2]	rs[1]	rs[0]	rt[3]	rt[2]	rt[1]	rt[0]	rd[3]	rd[2]	rd[1]	rd[0]
beq	0	0	0	1	0	0	Imm				Src Reg 1				Src Reg 2			
bne	0	0	0	1	0	1												
blez	0	0	0	1	1	0												
bgtz	0	0	0	1	1	1												
Type-1 Branch Instruction																		
	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	op[4]	op[3]	op[2]	op[1]	op[0]	funct[0]					rt[3]	rt[2]	rt[1]	rt[0]	rd[3]	rd[2]	rd[1]	rd[0]
bltz	0	0	0	0	1	0	Imm				Src Reg 1				Src Reg 2			
bgez	0	0	0	0	1	1												

Figure 10: Type 2 and Type 1 Branch Instructions

J-Type Jump Instructions																		
	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	op[4]	op[3]	op[2]	op[1]	op[0]	funct[0]	add[11]	add[10]	add[9]	add[8[	add[7]	add[6]	add[5]	add[4]	add[3]	add[2]	add[1]	add[0]
j	0	0	0	0	0	0	Constant Address Offset											
jal	0	0	0	0	0	1												

Figure 11: J-Type Jump Instructions

Data Path

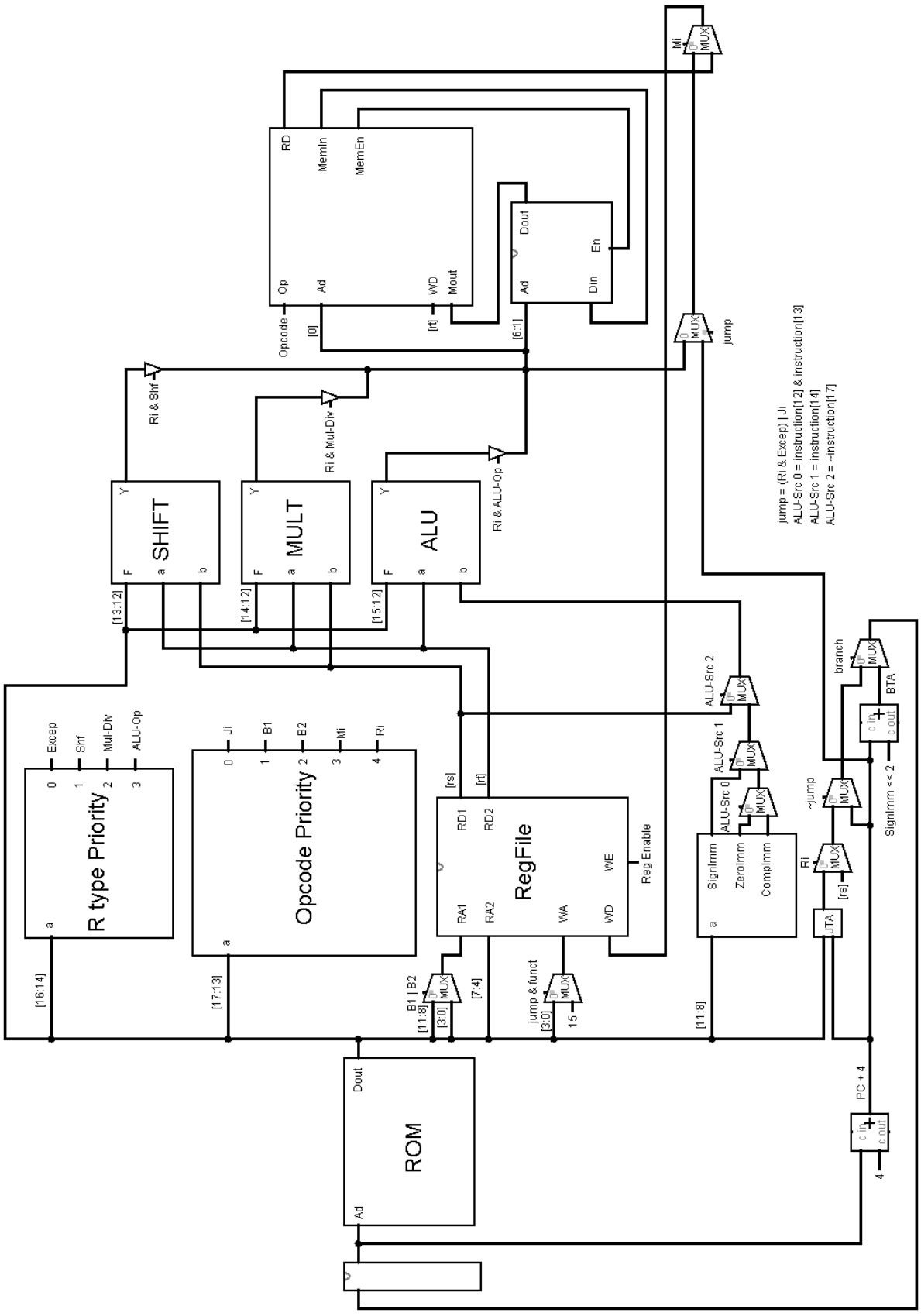

Figure 12: Complete Data path

Control Signals

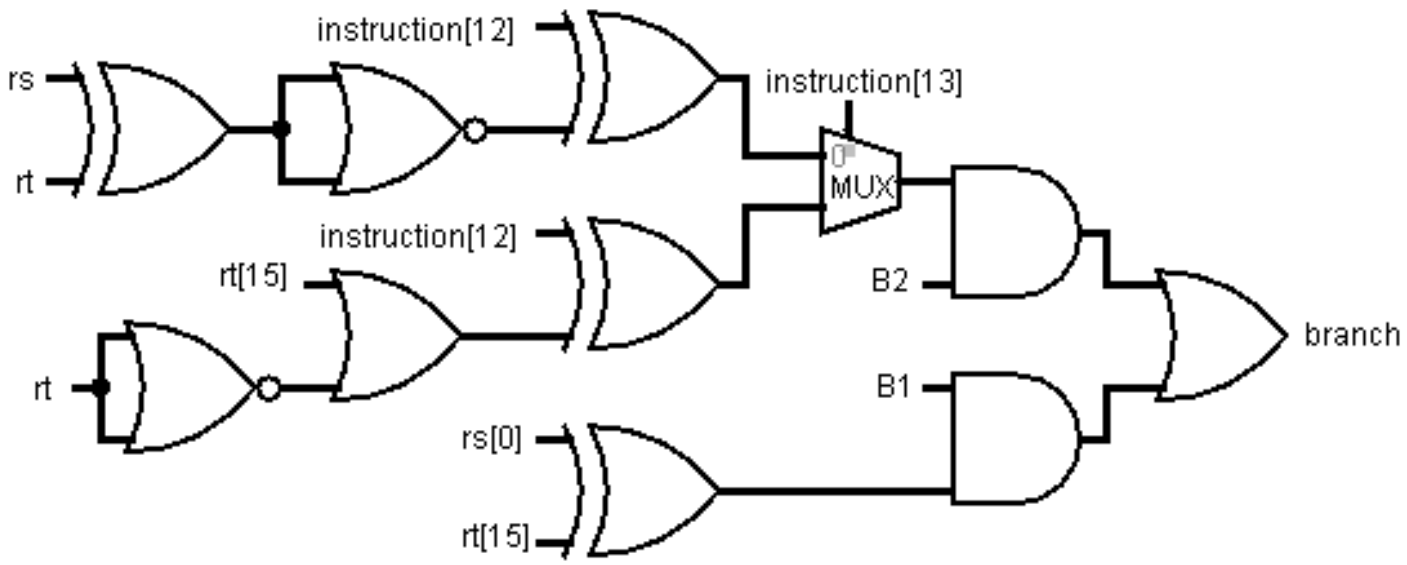

Figure 13: Branch Control Signals

For both Type-1 and Type-2 Branch instructions, the offset depends on the *imm* value which is 4-bit wide (bit 8 to bit 11 in an instruction). The offset is calculated based on the expression :

$$Offset = SignImm << 2$$

That makes the offset ranging from **0 to 16**.

As for the J-Type Jump instruction, the offset datawidth is 12-bit wide, ranging from **0 to 4095**.

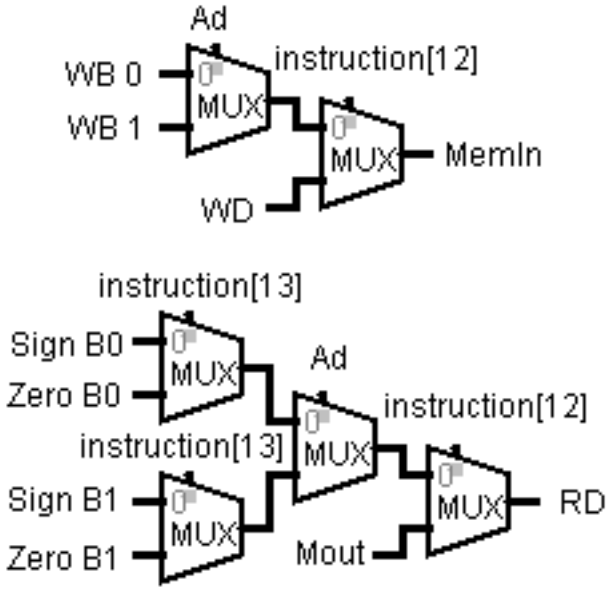

Figure 14: Memory Access Control Signals

The memory address scheme is **base + offset** based addressing, where the **base** is the base address stored in *rt*, and the offset is stored in *imm*. The offset is 4-bit wide ranging between **0 to 15**.

Testing Methodology

Each module was tested individually through the development board with switch pin inputs and LED/Seven Segment Display outputs. The test programs that were used were done by hand as this was done by selecting the specific Opcodes as well as the relative registers (Source, Target, Destination). Below are the test programs mentioned used for testing :

Instruction set to test I-Type ALU instructions

```
0100000010000010001 // addi 4 1 1
0100001110000010010 // addi -4 1 2
0100010001001100011 // addiu 1 3 3
0110100010001101000 // slti 2 3 4
0101000010001101010 // andi 2 3 5
0101010010001101100 // ori 2 3 6
0101100010001101110 // xori 2 3 7
0000000000000000000 // j 0 0 0
```

Instruction set to test R-Type multiplication and division module

```
0100000010000010001 // addi 2 1 1
0100000011001000100 // addi 3 2 2
1011000010000100000 // mult 2 1 x
1010100000000000011 // mflo x x 3
1010000000000000000 // mfhi x x 0
1011100001001100000 // div 1 3 x
1010100000000000100 // mflo x x 4
1010000000000000000 // mfhi x x 0
1010110000010000000 // mtlo x 4 0
1010100000000000101 // mflo x x 5
1010010000001000000 // mthi x 2 x
1010000000000000110 // mfhi x x 6
0000000000000000000 // j 0 0 0
```

Instruction set to test R-Type shifter instructions.

```
0100000010000010001 // addi 2 1 1
0100000001001000100 // addi 1 2 2
1001000001000100110 // sllv 1 1 3
1001010010001101000 // srlv 2 3 4
1001110010010001010 // srav 2 4 5
0000000000000000000 // j 0 0 0
```

The instruction set for demonstration :

```
0100000001000100011 // addi 1 1 1
0100000010000100102 // addi 2 1 2
1001000001000100110 // sllv 1 1 3
1011000011001000000 // mult 3 2 0
1010100000000001000 // mflo 0 0 4
1011100011010000000 // div 3 4 0
1010100000000001010 // mflo 0 0 5
1100100001010001100 // sub 1 4 6
0001000001010100100 // beq 1 1 2
0001010001010100100 // bne 1 1 2
0100000001011101110 // addi 1 7 7
0100000001011101110 // addi 1 7 7
0100000001011101110 // addi 1 7 7
0000000000000000000 // j 0 0 0 0
```