CECS 440 – COMPUTER ARCHITECTURE
INSTRUCTOR: R.W. ALLISON
TTH 12:30 – 2:45 PM
TRIEUVY LE 014466021
MARK AQUIAPAO 014000153
DUE DATE: APRIL 23, 2019

Table of Contents

I. Purpose

This documentation outlines the basic operation of our CECS 440 Computer Architecture MIPS 32-bit RISC machine. Within the next couple weeks, we will be designing a 32-bit Reduced Instruction Set Architecture (RISC) based on the MIPS architecture. The baseline requirements include the memory structure, register set, data types, addressing modes, different types of instruction set (R, I, J), instruction formats, interrupt, and reset. We will be adding more enhancements to the final project including more registers, different data types, more complex addressing modes and operations. As of now the Integer Datapath is the only component built which consist of the ALU which executes all the macro operations of the processor and the dual port 32-bit register which reads and stores data to each register in memory.

II. Instruction Set Architecture

Memory

Harvard Memory Architecture or Organization

This type of architecture stores machine type instructions and data in separate memory addresses which are connected through various busses (signal pathways). Harvard architecture allows the computer to run a program and access data independently where both a 32-bit program memory and data memory can be accessed simultaneously. This avoids the problems of bottlenecks and system performance that are prone to the Von-Neumann architecture. This current implementation of the MIPS 32-bit RISC (Reduced Instruction Set Computer) processor has the capability of having read-only access to instruction memory and read-write access to I/O memory.

Restriction: Harvard architecture has a strict separation between code and data with separate pipelines which makes it more complicated than the Von Neumann architecture. For running code execution simulations, the address space needs a minimum of 0x03FF.

The processor's memory contains a 32-bit address where each memory location is byte-addressable. With the Big Endian format, each 32-bit memory operands are stores in four consecutive memory locations where the eight least significant bits(LSB) are stored at the HIGHEST address while the eight most significant bits(MSB) are stored at the LOWEST address.

32 24 23 16 15 8 7 0

00	AB	CD	EF

0x1209	**00**
0x120A	**CD**
0x120E	**EF**
0x120F	**00**

Machine Type Register Set

Name(s)	Register(s) bit size	Number(s)	Function(s)
$zero	32	0	Contains Constant Value 0
$at	32	1	Assembler Temporary
$v0 - $v1	32	2 - 3	Value for Function Results and Expression Evaluation
$a0 - $a3	32	4 - 7	Argument Values
$t0 - $t7	32	8 - 15	Temporary Values
$s0 - $s7	32	16 - 23	Saved Temporaries
$t8 - $t9	32	24 - 25	More Temporary Values
$k0 - $k1	32	26 - 27	Reserved for OS Kernel
$gp	32	28	Global Pointer
$sp	32	29	Stack Pointer
$fp	32	30	Frame Pointer
$ra	32	31	Return Address
{PC, HI, LO}	32	N/A	{Program Counter, HI, LO}

Flags	5	N/A	Interrupt(I), Carry(C), Overflow(V), Negative(N), Zero(Z)

* Note reference from Green Sheet in *Patterson and Hennessy, Computer Organization and Design 4th edition*

Data Types

32-bit and 16-bit Signed Integers

A signed integer has a value that can be either or positive or negative which shows the ranges below by corresponding bit size and values:

32	16	0
-2147483648		2147483647

Hexadecimal Value	Decimal Value
0x00000001	1
0xFFFFFFF2	-14
0xFFFFFFFC	-4
0x0000000F	15

16	8	0
-32768		32767

Hexadecimal Value	Decimal Value

0x0001	1
0xFFF0	-15
0xFFFE	-1
0x000F	15

32-bit and 16-bit Unsigned Integers

An unsigned integer has a value that can ONLY be positive where values and the ranges are shown below by corresponding bit size:

32	16	0
0		21474836447

Hexadecimal Value	Decimal Value
0x00000001	1
0xFFFFFFF2	4294967282
0xFFFFFFFC	4294967292
0x0000000F	15

16	8	0
0		65535

Hexadecimal Value	Decimal Value
0x0005	5

0xFFF0	65520
0xFFFE	65534
0x000C	12

Addressing Modes

Addressing modes are a specification for generating an operand's address at run-time**.** Since the MIPS 32-bit processor has a RISC machine, the load and store instruction are the ONLY access to memory whereas in CISC(Complex Instruction Set Computer) utilizes push and pop instructions. The addressing modes in this processor are:

Intermediate

The operand is part of the instruction and is useful for initializing registers with address or data constraints.

Instruction Format:	op	$rd, 16-bit immediate value
Examples:	addi	$r1, #0x0A18
	ori	$r3, #0x0123

In these examples 0x0A18 and 0x0123 are the trailing words of this addressing mode. When the a immediate instruction is executed the PC(program counter) will be loaded with the trailing word of instruction. So after the addi instruction the PC would go to memory address 0x0A18 and in the ori 0x0123.

Register Direct

The instruction that contains the address of an operand and can be also known as "absolute" addressing. The address MUST be known at compile time.

Instruction Format:	op	$rd, $rs, $rt
Examples:	sub	$r4, $r7, $r10

```
srl     $r5, $r6, 10
```

The registers are ones that act as the effective address in memory that will either be read or written to. In the subtraction instruction, the data in $r7 is minus the data in $r10 and will stored in $r4. In the shift right logical, the content in $r6 will be shifted to the right 10 times with zero fills and the result is stored in $r5. Note that the $rd is the only address that will have its data overwritten.

Register Indirect

The instruction that specifies the address of an operand and is used to access memory operands by "base pointers" that are stored in each register. This mode can only be executed by load, store, and jump type of instructions

Instruction Formats:	op	$rd, $rs, 16-bit value
	op	$rd, 26-bit value
Examples:	lw	$r4, 2($r7)
	sw	$r4, 2($r7)
	j	loop

The data in the registers are ones that act as the address in memory that will either be read or written to.

Based Index

A variation of register indirect mode where the address of the operand is calculated by the addition of a "base" register and an offset(signed displacement) that is followed after the instruction as a trailing word.

Instruction Formats:	op	$rd, [$rs + offset]
Examples:	j	[$r8 + 0xA231]

The data in the register plus the offset(trailing word) is calculated as the effective address that the instruction will go to in memory. In the jump instruction, the effective address jumped to is calculated by adding the data from $r8 and the offset value when the PC is at the next address after the trailing word of the instruction.

PC Relative

A variation of base index mode where the effective address of the operand is calculated by the addition of a "base" register by the sum of the current PC and an offset(signed displacement) that is followed after the instruction as the trailing word. This mode can ONLY be executed by a jump-type operator.

Instruction Formats:	op	$rd, [PC + offset]
Examples:	j	[PC + 0xA231]

The data in the register plus the offset(trailing word) is calculated as the effective address that the instruction will go to in memory. In the jump instruction, the PC will jump to the effective address is calculated by adding PC and the offset value when the PC is at the next address after the trailing word of the instruction.

Instruction Set

R - Type Instruction Set

Register-Type

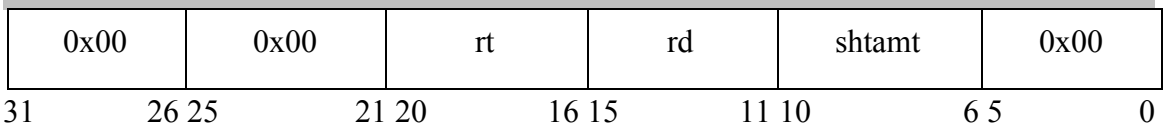
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00

0x0	rs code	rt code	rd code	shtamt	function code

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00

0x1F	rs code	rt code	0x0	0x0	function code

SLL - Shift Word Left Logical



Format: sll $rd, $rt, shtamt

Purpose: Perform a logical left shift by a fixed number of bits.

Description: $rd ← $rt << shtamt

The content of *$rt* is shifted to the left as many as *shtamt* specifies with 0's filled.

Restriction: None

Operation:

Fetch: PC ← PC + 4

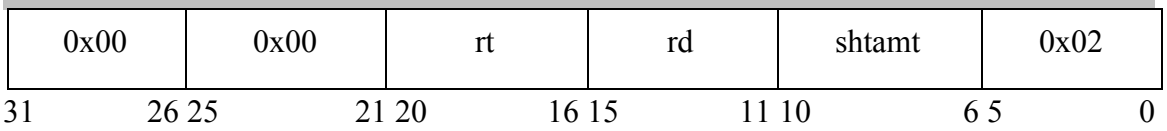
Decode: $rs ← $r[IR[25-21]]; $rt ← $r[IR[20-16]];

SLL: ALU_OUT← $rt << shtamt

Example:

0x3C01_0000	// LUI $r1, 0x0000	
0x3421_FFFF	// ORI $r1, 0xFFFF	
0x0001_0C00	// SLL $r1, $r1, 0x16	R1= 0xFFFF_0000

SRL - Shift Word Right Logical



Format: srl $rd, $rt, shtamt

Purpose: Perform a logical right shift by a fixed number of bits.

Description: $rd ← $rt >> shtamt

The content of *$rt* is shifted to the right as many as *shtamt* specifies. For logical shift, we are operating unsigned operation which means the sign arithmetic bit is replaced with a 0 considering the value of the bus as positive.

Restriction: None

Operation:

Fetch: PC ← PC + 4l;

Decode: $rs ← $r[IR[25-21]]; $rt ← $r[IR[20-16]];

SRL: ALU_OUT ← $rt >> shtamt

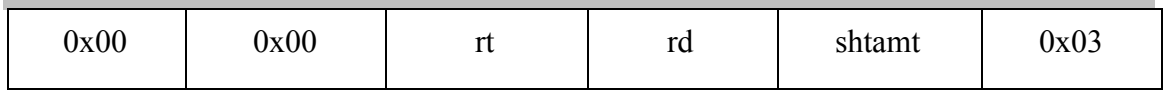
Exceptions: None

Programming Notes:

0x3C01_0000	// LUI $r1, 0x0000	
0x3421_FFFF	// ORI $r1, 0xFFFF	

0x0001_0C00	// SRL $r1, $r1, 0x16	R1= 0x0000_FFFF

SRA – Shift Word Right Arithmetic



Format: sra $rd, $rt, shtamt

Purpose: Perform an arithmetic right shift by a *shtamt* of bits.

Description: $rd ← $rt >> shtamt

The content of $rt is shifted to the right as many as shtamt specifies arithmetically by casting the operand as integer. The program copies the MSB content before the shift and places it in the MSB after the shift operation performed.

Restriction: None

Operation:

Fetch: PC ← PC + 4l;

Decode: $rs ← $r[IR[25-21]]; $rt ← $r[IR[20-16]];

SRA: ALU_OUT ← $rt >> shtamt

Exceptions: None

Programming Notes:

add $r1, $zero, 0xF0F0_3C3C	
add $r2, $zero, 0xBF0F_F5F5	

sra $r0, $r1, $r2	// $r0 = DF87_FAFA

JR–Jump Register

Format: jr $rs

Purpose: Perform jump to the address stored register $*rs*.

Description: Jump to the effective address that $rs contains.

Restriction:

Operation:

Fetch: PC ← PC + 4;

Decode: RS ← Reg [IR[25 - 21]] ;

JR: ALU_OUT ← $rs;

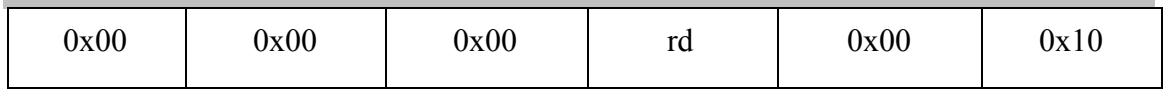
PC ← ALU_OUT($rs);

Exceptions: None

Examples:

0x1234_5678	// LUI $r0, 0x4321	
0xABCD_EF11	// ORI $r0, 0x8765	
0x1111_1111	// JR $r0	// jump to R1= 0x4321_8765

MFHI – Move From Hi Register



Format: mfhi $rd

Purpose: To move the value stored in Hi after a multiply/divide into the destination register *$rd*

Description: $rd ← Hi

The register $rd receives and stores the value in $Hi register which is the upper 2 bytes.

Restriction: MFHI and MFLO

Operation: We will need to have already performed either a multiplication/division.

mult ALU_OUT ← $rs * $rt

div ALU_OUT ← $rs / $rt

Fetch: PC ← PC + 4; IR ← M[PC];

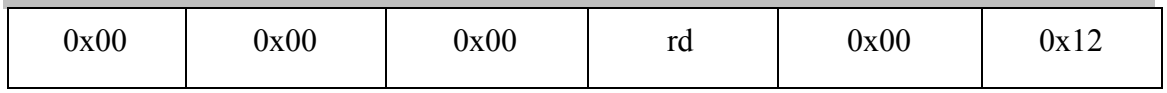
mfhi: ALU_OUT ← Product[63:32]

Exceptions: None

Examples: $r1 = 0x0000 03B2; $r2 = 0x0000 2A71

mult $r1, $r2	$r1 * $r2 = 0x009C D592
mfhi $t0	$t0 = 0x009C

MFLO – Move From Lo Register



Format: mflo $rd

Purpose: To move the value stored in Lo after a multiply/divide into the destination register *$rd*

Description: $rd ← Lo

The register $rd receives and stores the value in $Lo register which is the lower 2 bytes.

Restriction:

Operation: We will need to have already performed either a multiplication/division.

mult ALU_OUT ← $rs * $rt

div ALU_OUT ← $rs / $rt

Fetch: PC ← PC + 4; IR ← M[PC];

Decode: RS ← Reg [IR[25 - 21]] ;

mflo: ALU_OUT ← Product[63:32];

Exceptions: None

Examples: $r1 = 0x0000 03B2; $r2 = 0x0000 2A71

mult $r1, $r2	$r1 * $r2 = 0x009C D592
mflo $t0	$t0 = 0xD592

MUL– Multiplication

0x00	rs	rt	0x00	0x00	0x18

31 26 25 21 20 16 15 11 10 6 5 0

Format: MULT $rs, $rt

Purpose: To multiply two 32-bit signed integers.

Description: The operation yields a 64-bit product after performing a multiplication between $rs and $rt. The lower 32-bit is stored in LO register and the higher 32-bit is stored in the HI register.

Restriction: The operands $rs and $rt must be 32-bit signed value

Operation:

Fetch: PC ← PC + 4; IR ← M[PC];

Decode: RS ← Reg [IR[25 - 21]] ;

MUL ALU_OUT ← $rs * $rt

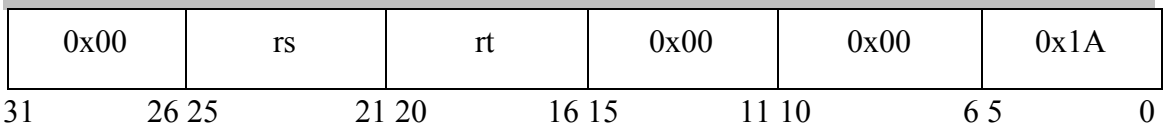
Exceptions: None

Examples: $r1 = 0x0000 03B2; $r2 = 0x0000 2A71

addi $r1, $zero, 0x03B2	$r1 = 0x03B2

addi $r2, $zero, 0x2A71	$r2 = 0x2A71
mult $r1, $r2	$r1 * $r2 = 0x009C D592

DIV - Divide Word



Format: DIV $rs, $rt

Purpose: To divide two 32-bit signed integers

Description: The operands will perform a division $rs / $rt. The 32-bit quotient is stored in the LO register and the remainder is stored in the HI register.

Restriction: If the divisor in $rt is 0, the result of operation is undefined

Operation:

Fetch: PC ← PC + 4; IR ← M[PC];

Decode: RS ← Reg [IR[25 - 21]] ;

DIV ALU_OUT ← $rs / $rt

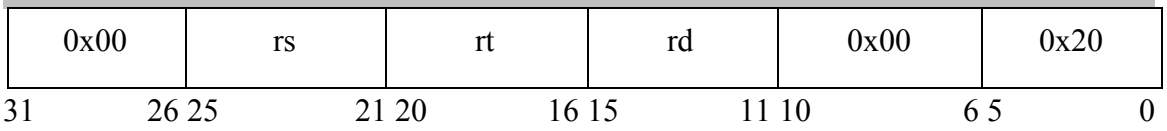
Exceptions: None

Examples:

addi $r1, $zero, 0x0025	$r1 = 0x0025

addi $r2, $zero, 0x001D	$r2 = 0x001D
div $r1, $r2	{Rem, Quot} = {0x0008, 0x0001}

ADD - Add



Format: ADD $rd, $rs, $rt

Purpose: To perform an addition between registers $rs and $rt then store the sum to $rd

Description: The operands will produce a 32-bit sum

The operation will only overflow when adding two positive and result is negative; vice versa, when adding two negative numbers and result is positive.

Restriction: The operands $rs and $rt must be 32-bit signed

Operation:

Fetch: PC ← PC + 4; IR ← M[PC];

Decode: RS ← Reg [IR[25 - 21]] ; RS ← Reg [IR[20 - 16]]

Add RD ← $rs + $rt

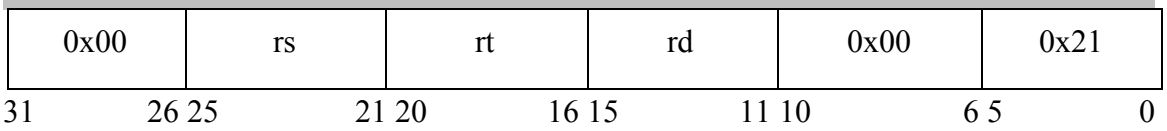
Exceptions: None

Examples:

addi $r1, $zero, 0x0025	$r1 = 0x0025
addi $r2, $zero, 0x001D	$r2 = 0x001D
add $r0,$r1, $r2	$r0 = 0x0042 **No overflow**

addi $r1, $zero, 0x7FFF	$r1 = 0x7FFF
addi $r2, $zero, 0x0002	$r2 = 0x0002
add $r0,$r1, $r2	$r0 = 0x8000 0001 **Yes overflow**

ADDU - Add Unsigned



Format: ADDU $rd, $rs, $rt

Purpose: To add two unsigned 32-bit integers.

Description: $rd ← $rs + $rt

The add unsigned operation adds two unsigned 32-bit operands and puts the result in the destination register $rd. The negative flag is set to 0 and by definition, the unsigned operation produces positive number thus does not require an arithmetic sign.

Restriction: None

Operation:

Fetch: PC ← PC + 4; IR ← M[PC];

Decode: RS ← Reg [IR[25 - 21]] ;

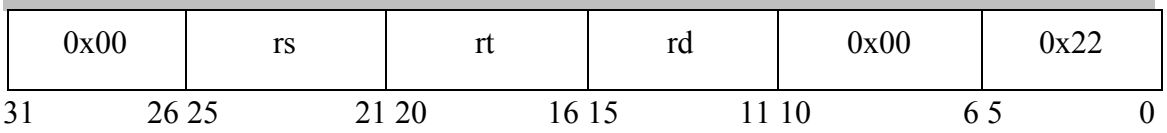
ADDU: ALU_OUT ← $rs + $rt;

Exceptions: None

Examples:

addi $r1, $zero, 0x0025	$r1 = 0xFFFF_FFC9
addi $r2, $zero, 0x001D	$r2 = 0x0000_000D
addu $r0,$r1, $r2	$r0 = 0xFFFF_FFD6

SUB - Subtract



Format: SUB $rd, $rs, $rt

Purpose: To perform a subtraction between registers $rs and $rt then store the difference to $rd.

Description: The operands will produce a 32-bit difference.

The operation will only overflow when subtracting a positive to a negative with a negative result; a negative subtracting a positive and you get a positive result.

Restriction: The operands $rs and $rt must be 32-bit signed

Operation:

Fetch: PC ← PC + 4; IR ← M[PC];

Decode: RS ← Reg [IR[25 - 21]] ; RS ← Reg [IR[20 - 16]]

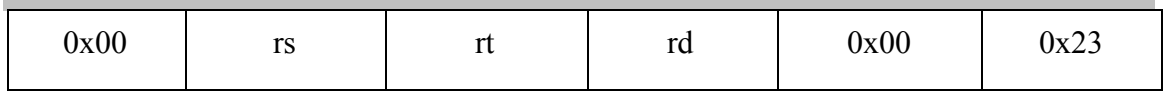
sub RD ← $rs - $rt

Exceptions: None

Examples:

addi $r1, $zero, 0x0025	$r1 = 0x0000_0025
addi $r2, $zero, 0x001D	$r2 = 0x0000_001D
sub $r0,$r1, $r2	$r0 = 0x0000_0008 No overflow

SUBU - Subtract Unsigned



Format: SUBU $rd, $rs, $rt

Purpose: To subtract two unsigned 32-bit integers.

Description: $rd ← $rs + $rt

The subtract unsigned operation subtracts two unsigned 32-bit operands and puts the result in the destination register $rd. The negative flag is set to 0 and by definition, the unsigned operation produces positive number thus does not require an arithmetic sign. The operation will only overflow when subtracting a positive to a negative with a negative result; a negative subtracting a positive and you get a positive result.

Restriction: None

Operation:

Fetch: PC ← PC + 4; IR ← M[PC];

Decode: RS ← Reg [IR[25 - 21]] ;

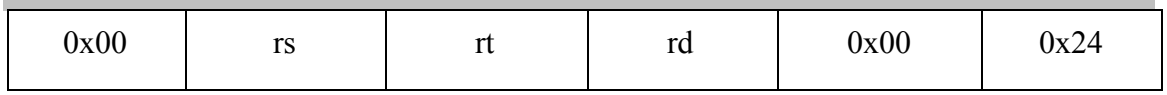
SUBU: ALU_OUT ← $rs – $rt;

Exceptions: None

Examples:

addi $r1, $zero, 0x020D	$r1 = 0x0000_020D
addi $r2, $zero, 0xFFE3	$r2 = 0xFFFF_FFE3
subu $r0,$r1, $r2	$r0 = 0x0000_022A Overflow

AND - Logical AND



Format: AND $rd, $rs, $rt

Purpose: To logically and two 32-bit values.

Description: $rd ← $rs & $rt

The values are initially loaded into $rs and $rt then being logically and then stores the result in $rd. The bit is active if and only both are 1, otherwise stays low. The negative and zero flags will be high if the result is with a leading 1 or 0's respectively.

Restriction: None.

Operation:

Fetch: PC ← PC + 4; IR ← M[PC];

Decode: RS ← Reg [IR[25 - 21]] ; RS ← Reg [IR[20 - 16]]

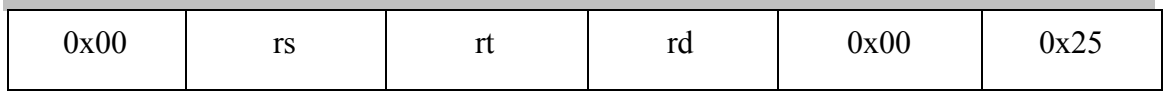
AND: ALU_OUT ← $rs & $rt;

Exceptions: None

Examples:

addi $r1, $zero, 0xABCD_EF01	$r1 = 0xABCD_EF01
addi $r2, $zero, 0xA210_2811	$r2 = 0xA210_2811
and $r0,$r1, $r2	$r0 = A200_2801 Negative

OR - Logical OR



Format: OR $rd, $rs, $rt

Purpose: To logically or two 32-bit values.

Description: $rd ← $rs | $rt

The values are initially loaded into $rs and $rt then being logically or then stores the result in $rd. The bit is active if either one of the values being compared is a 1. The negative and zero flags will be high if the result is with a leading 1 or 0's respectively.

Restriction: None.

Operation:

Fetch: PC ← PC + 4; IR ← M[PC];

Decode: RS ← Reg [IR[25 - 21]] ; RS ← Reg [IR[20 - 16]]

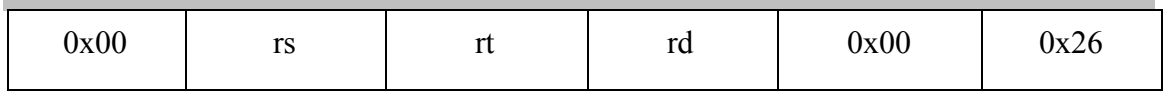
OR : ALU_OUT ← $rs | $rt;

Exceptions: None

Examples:

addi $r1, $zero, 0xABCD_EF01	$r1 = 0xABCD_EF01
addi $r2, $zero, 0xA210_2811	$r2 = 0xA210_2811
or $r0,$r1, $r2	$r0 = ABDD_EF11Negative

XOR - Logical Exclusive OR



Format: XOR $rd, $rs, $rt

Purpose: To logically exclusive or two 32-bit values.

Description: $rd ← $rs ^ $rt

The values are initially loaded into $rs and $rt then being logically exclusive or then stores the result in $rd. The bit is active if the operand bits are opposite, otherwise the signal stays low.

Restriction: None.

Operation:

Fetch: PC ← PC + 4; IR ← M[PC];

Decode: RS ← Reg [IR[25 - 21]] ; RS ← Reg [IR[20 - 16]]

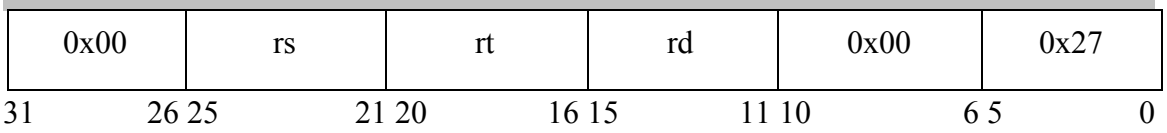
XOR : ALU_OUT ← $rs ^ $rt;

Exceptions: None

Examples:

addi $r1, $zero, 0xABCD_EF01	$r1 = 0xABCD_EF01
addi $r2, $zero, 0xA210_2811	$r2 = 0xA210_2811
xor $r0,$r1, $r2	$r0 = 0x09DD_C710

NOR – Logical Not OR



Format: NOR $rd, $rs, $rt

Purpose: To negate the result after OR two 32-bit values.

Description: $rd ← ~ ($rs | $rt)

The values are initially loaded into $rs and $rt then being logically exclusive or then stores the result in $rd. The negative and zero flags will be high if the result is with a leading 1 or all 0's respectively.

Restriction: None.

Operation:

Fetch: PC ← PC + 4; IR ← M[PC];

Decode: RS ← Reg [IR[25 - 21]] ; RS ← Reg [IR[20 - 16]]

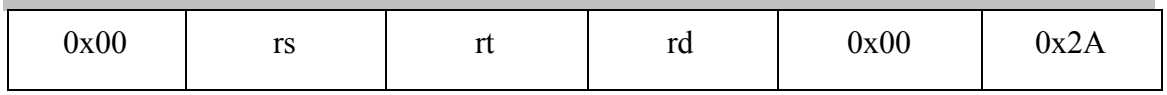
NOR : ALU_OUT ← ~ ($rs | $rt)

Exceptions: None

Examples:

addi $r1, $zero, 0xABCD_EF01	$r1 = 0xABCD_EF01
addi $r2, $zero, 0xA210_2811	$r2 = 0xA210_2811
nor $r0,$r1, $r2	$r0 = 0x5422_10EE

SLT - Set Less Than



Format: SLT $rd, $rs, $rt

Purpose: The operation will set the destination register at high level if condition is met.

Description: $rd ← ($rs < $rt)

The destination register is set to a 1 if this condition is met $rs < $rt, else stays low.

Restriction: None

Operation:

Fetch: PC ← PC + 4; IR ← M[PC];

Decode: RS ← Reg [IR[25 - 21]] ;

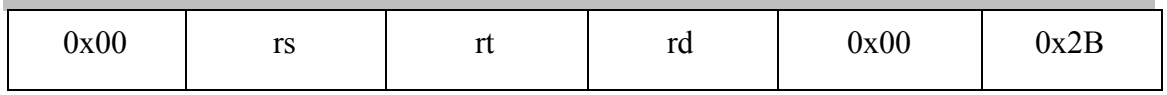
slt: ALU_OUT ← ($rs < $rt) ? 1 : 0 ;

Exceptions: None

Examples:

addi $r1, $zero, 0x0000_0025	$r1 = 0x0000_0025
addi $r2, $zero, 0x0000_001D	$r2 = 0x0000_001D
sltu $r0,$r1, $r2	$r0 = 0

SLTU - Set Less Than Unsigned



Format: SLTU $rd, $rs, $rt

Purpose: The unsigned operation will set the destination register at high level if condition is met.

Description: $rd ← ($rs < $rt)

The operation will compare the content in $rs and $rt as unsigned integers then if the condition is met, destination register is set to a 1. Otherwise the boolean result is 0.

Restriction: None

Operation:

Fetch: PC ← PC + 4; IR ← M[PC];

Decode: RS ← Reg [IR[25 - 21]] ;

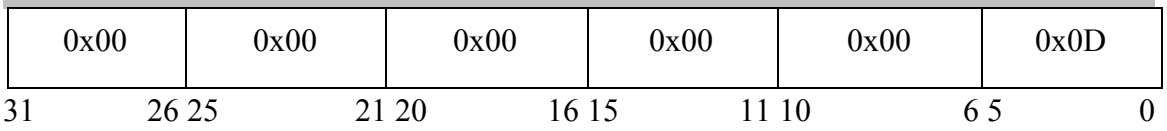
sltu: ALU_OUT ← ($rs < $rt) ? 1 : 0 ;

Exceptions: None

Examples:

addi $r1, $zero, 0x0000_001D	$r1 = 0x0000_001D
addi $r2, $zero, 0x0000_0025	$r2 = 0x0000_0025
sltu $r0,$r1, $r2	$r0 = 1

BREAK



Format: Break

Purpose: To stop the instruction

Description: After executing this instruction, the program will display the current runtime letting us know as a safe break. Then it will do 2 tasks of Register Dump and Memory Dump which to display the 32-bit CPU registers and Memory Registers.

Restriction: None

Operation:

Fetch: PC ← PC + 4; IR ← M[PC];

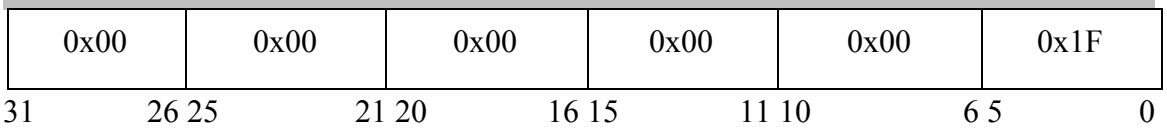
Next State: Break

Exceptions: None

Examples:

add $t1, $t1, $s0	//$t1 = 2*k
add $t0, $t0, $t0	//$t1 = 4*k
add $t1, $t1, $t4	//load with base addr $t4
lw $t0, 0($t1)	//load $t0 with the content
jr $t0	//jump to the addr in $t0
break	//safety break

SETIE - Set Interrupt Enable



Format: SETIE

Purpose: To set interrupt enable flag

Description: We will pass the current interrupt enable flag of a 1 to the next interrupt enable flag state.

Restriction: None

Operation:

Fetch: PC ← PC + 4; IR ← M[PC];

Decode: RS ← Reg [IR[25 - 21]] ;

Next State: SETIE

Exceptions: None

Examples:

main: setie		
ISR	lui $r1, 0xABCD	//load upper word
	ori $r1, $r1, 0xEF01	//or with lower word
	add $r2, $r1, $zero	//copy
	beq $r1, $r2, 0xFFFC	//branch to different location
	addi $r1, $r2, $r2	//else add immediate value

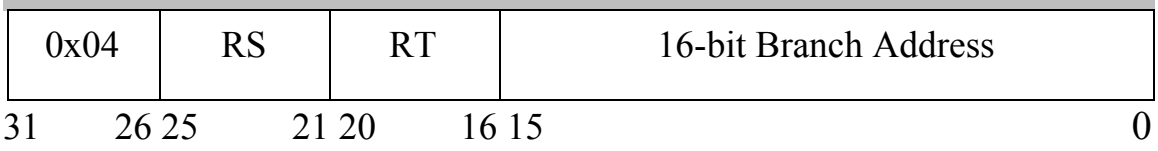
I - Type Instruction Set

Immediate-Type

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00

opcode	rs code	rt code	16-bit immediate

BEQ - Branch if Equal



Format: beq $rs, $rt, Label

Purpose: The PC will branch off into another memory location if both operands are equal to each other.

Description: PC ← PC + 4 + 16-bit immediate

At run time, the 16-bit immediate label is calculated to move a specific number of instruction forward or backwards.

Restriction: Since the 16-bit immediate value is signed, the branch has a limit of jumping a value between the range of-32768 to 32767

Operation: Functions as conditional if/else statement.

if($rs == $rt) PC ← PC + 4 + Branch Address

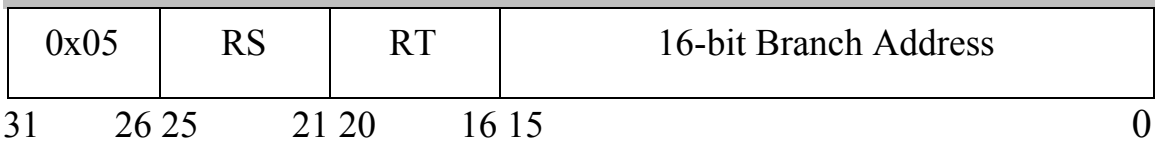
else PC ← PC + 4

Exceptions: No Exceptions

Examples:

beq $r1, $zero, 0xABCD	$r1 = 0x0025, will not branch to address 0xABCD(0x0025 != 0x0)
beq $r2, $zero, 0x001D	$r2 = 0x0, will branch to address 0x001D(0x0 = 0x0)
beq $r2, $r3, 0x018F	$r2 = 0x0, $r3 = 0x1, will not branch to address 0x018f(0x != 0x1)

BNE - Branch Not Equal



Format: bne $rs, $rt, Label

Purpose: The PC will branch off into another memory location if both operands are NOT equal to each other.

Description: PC ← PC + 4 + 16-bit immediate

At run time, the 16-bit immediate label is calculated to move a specific number of instructions forward or backwards.

Restriction: Since the 16-bit immediate value is signed, the branch has a limit of jumping a value between the range of -32768 to 32767

Operation: Functions as conditional if/else statement.

if($rs != $rt) PC ← PC + 4 + Branch Address

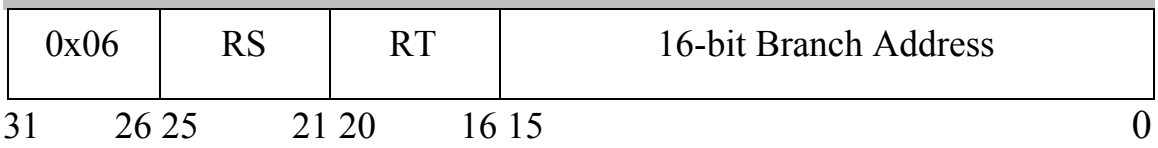
else PC ← PC + 4

Exceptions: No Exceptions

Examples:

bne $r8, $zero, 0x1DEF	$r8 = 0x0018, will branch to address 0x1DEF(0x0018 != 0x0)
bne $r3, $zero, 0x001D	$r3 = 0x0, will not branch to address 0x001D(0x0 = 0x0)
bne $r5, $r4, 0x1567	$r5 = 0x1, $r4 = 0x1, will branch to address 0x1567(0x1 != 0x1)

BLEZ - Branch if Less Than or Equal to Zero



Format: blez $rs, Label # $rt = 0

Purpose: The PC will branch off into another memory location if $rs is less than or equal to zero.

Description: PC ← PC + 4 + 16-bit immediate

At run time, the 16-bit immediate label is calculated to move a specific number of instruction forward or backwards.

Restriction: Since the 16-bit immediate value is signed, the branch has a limit of jumping a value between the range of -32768 to 32767

Operation: Functions as conditional if/else statement.

if($rs != $rt) PC ← PC + 4 + Branch Address

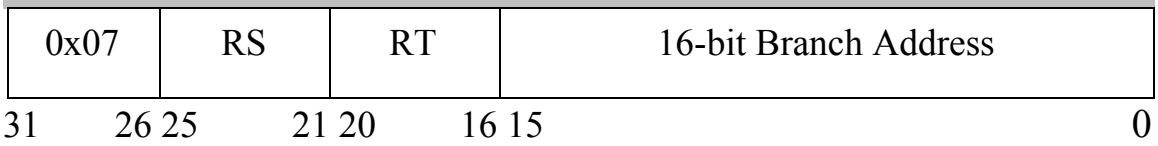
else PC ← PC + 4

Exceptions: No Exceptions

Examples:

blez $r7, 0x1DE8	$r7 = 0x0018, will branch to address 0x1DE8(0x0018 != 0x0)
blez $r3, 0x001C	$r3 = 0x0, will branch to address 0x001C(0x0 = 0x0)
blez $zero, 0x156C	$zero = 0x0, will branch to to address 0x156C

BGTZ - Branch if Greater Than or Equal to Zero



Format: bgtz $rs, Label # $rt = 0

Purpose: The PC will branch off into another memory location if $rs is greater than zero.

Description: PC ← PC + 4 + 16-bit immediate

At run time, the 16-bit immediate label is calculated to move a specific number of instruction forward or backwards.

Restriction: Since the 16-bit immediate value is signed, the branch has a limit of jumping a value between the range of -32768 to 32767

Operation: Functions as conditional if/else statement.

if($rs != $rt) PC ← PC + 4 + Branch Address

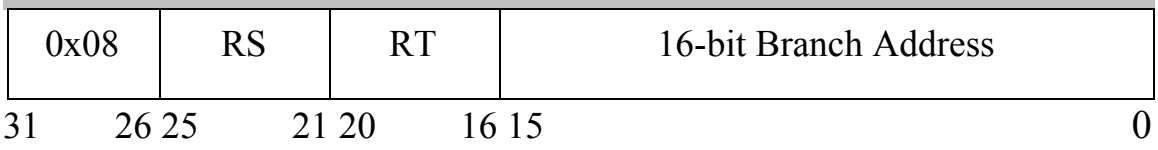
else PC ← PC + 4

Exceptions: No Exceptions

Examples:

bgtz $r7, 0x0008	$r7 = 0x0008, will branch to address 0x1DEF(0x0008 > 0x0)
bgtz $r3, 0x0018	$r3 = 0x0, will not branch to address 0x0018(0x0 !<= 0x0)
bgtz $zero, 0x1564	$r5 = 0x1, will branch to address 0x1564(0x1564 > 0x0)

ADDI – Add an Immediate Value



Format: addi $rt, $rs, Number

Purpose: To add a signed extended 16-bit value to a register and save the result in another register.

Description: $rt ← $rs + 16-bit immediate

The contents of $rs and the 16-bit immediate value will be added and stored as the content of $rt.

Restriction: Since the 16-bit immediate value is signed, the the instruction can add a register's contents to a value between the range of -32768 to 32767

Operation: y = a + (value)

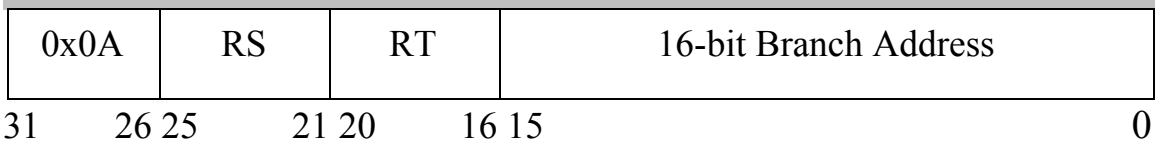
Exceptions: No Exceptions

Examples:

addi $r5, $r8, 0x1098	$r8 = 0x0987, therefore $r5 = 0x117F

addi $r4, $r8, 0x0002	$r8 = 0x0125, therefore $r4 = 109A
addi $r4, $r0, 0x00FF	Therefore $r4 = 0x00FF

SLTI- Set Less Than Immediate



Format: slti $rt, $rs, value

Purpose: To compare the contents of a register with a signed 16-bit immediate value.

Description: $rt ← 0x1 if $rs is less than the 16-bit value

$rt ← 0x0 if $rs is not less than the 16-bit value

Restriction: Since the 16-bit immediate value is signed, the instruction has a limit of a register's content being compared to a value between the range of **-32768 to 32767**

Operation: if($rs < 16-bit value) $rt ← 0x1

else $rt ← 0x0

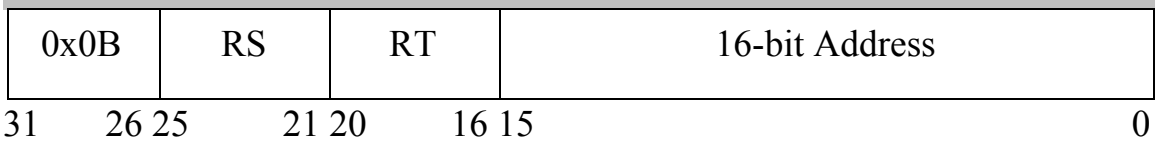
Exceptions: No Exceptions

Examples:

slti $r7, $zero, 0x0006	$r7=32'b1 since 0 < 0x0006

slti $r3, $r1 0x0010	$r1 = 0x000A, $r3=32'b1 since 0x000A < 0x00010
slti $r10, $r5, 0x0001	$r5 = 0x1A23, $r10=32'b0 since 0x1A23 is not < 0x0001

SLTIU - Set Less Than Immediate Unsigned



Format: sltiu $rt, $rs, value

Purpose: To compare the contents of a register with a unsigned 16-bit immediate value.

Description: $rt ← 0x1 if $rs is less than the 16-bit value

$rt ← 0x0 if $rs is not less than the 16-bit value

Restriction: Since the 16-bit immediate value is unsigned, the instruction has a limit of a register's content being compared to a value between the range of **0 to 65535**

Operation: if($rs < 16-bit value) $rt ← 0x1

else $rt ← 0x0

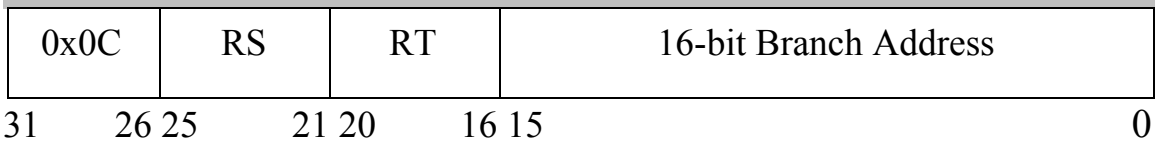
Exceptions: No Exceptions

Examples:

sltiu $r8, $zero, 0x0007	$r8=32'b1 since 0x0 < 0x0007

sltiu $r2, $r1, 0x0011	$r1 = 0x000B, $r2=32'b1 since 0x000B < 0x00011
sltiu $r10, $r5, 0x0001	$r5 = 0x1A23, $r10=32'b0 since 0x1A23 is not < 0x0001

ANDI - And Immediate



Format: andi $rt, $rs, value

Purpose: To perform a *bitwise and* on the contents of a register with a 16-bit immediate value and store the result in another register.

Description: $rt ← $rs & value

Restriction : No restriction.

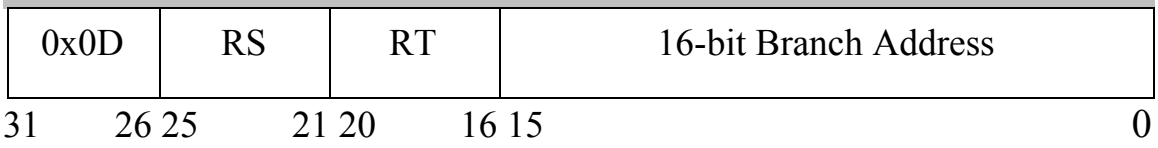
Operation: result = (register content) & (16-bit value)

Exceptions: No Exceptions

Examples:

andi $r7, $r4, 0x000C	$r7 = 0x00A1, $r7= 0x00A1 & 0x000C = 0x0
andi $r7, $r3, 0x0018	$r3 = 0x0045, $r7= 0x0045 & 0x0018 = 0x0004
andi $r10, $r5, 0x0011	$r5 = 0x0011, $r7= 0x0011 & 0x0011= 0x0

ORI - Or Immediate



Format: ori $rt, $rs, value

Purpose: To perform a *bitwise or* on the contents of a register with a 16-bit immediate value and store the result in another register.

Used after a LUI instruction to finish loading a register with a 32-bit immediate value by loading the lower 16 bits.

Description: $rt ← $rs | value

Restriction: No Restrictions

Operation: result = (register content) | (16-bit value)

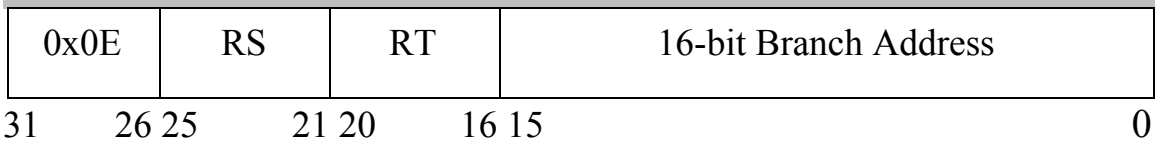
Exceptions: No Exceptions

Examples:

ori $r4, $r3, 0x000C	$r3 = 0x00B2, $r4= 0x00B2 | 0x000C = 0x00BC
ori $r3, $r7, r0x0018	$r3 = 0x0010, $r7= 0x0010 | 0x0018 = 0x0018

ori $r11, $r5, 0x0011	$r5 = 0x0011, $r5= 0x0011| 0x0011= 0x0011

XORI - Exclusive Or Immediate



Format: xori $rt, $rs, value

Purpose: To perform a *bitwise exclusive or* on the contents of a register with a 16-bit immediate value and store the result in another register.

Description: $rt ← $rs ^ value

Restriction: No Restrictions

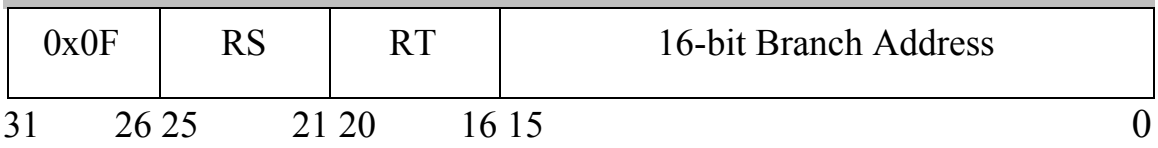
Operation: result = (register content) ^ (16-bit value)

Exceptions: No Exceptions

Examples:

xori $r4, $r3, 0x000D	$r3 = 0x00B2, $r4= 0x00B2 ^ 0x000A = 0x00B8
xori $r3, $r7, r0x001A	$r3 = 0x001A, $r7= 0x001A ^ 0x001A = 0x0
xori $r11, $r5, 0x001A	$r5 = 0x0011, $r5= 0x0011^ 0x001A= 0x000B

LUI - Load Upper Immediate



Format: lui $rt, value

Purpose: To load a 16-bit value(half a word) into the upper 16 bits of a register.

Used before a ORI instruction to start loading a register with a 32-bit immediate value.

Description: $rt ← 16-bit value, 16'h0

Restriction: No Restrictions

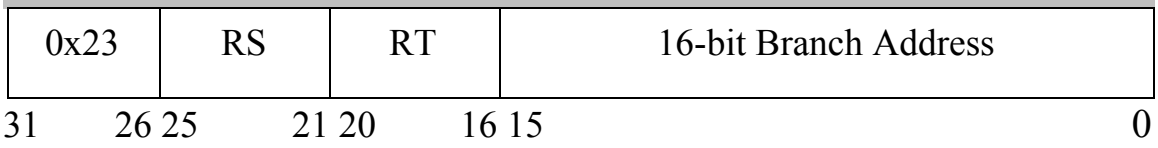
Operation: $rt ← { 16-bit value, 16'h0 }

Exceptions: No Exceptions

Examples:

lui $r7, 0x0EF1	$r7 = 0x0EF1_XXXX
lui $r3, 0x0018	$r3 = 0x0008_XXXX
lui $r10, 0x1564	$r10 = 0x1564_XXXX

LW - Load Word



Format: lw $rt, 16-bit immediate($rs)

Purpose: To load a 32-bit immediate value to a register by adding the contents of another register and a sign extended 16-bit immediate.

Description: $rt ← M[$rs + 16-bit immediate]

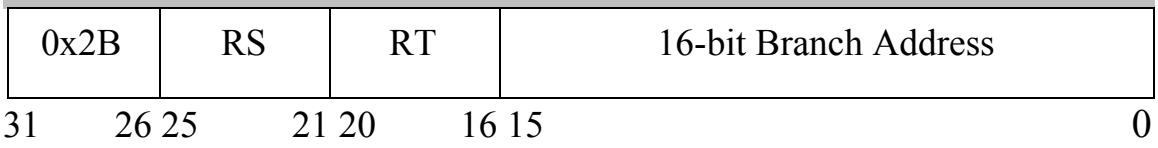
Restriction: The 16-bit immediate value can only have a value from -32768 to 32767.

Exceptions: No Exceptions

Examples:

lw $r8, $r7, 0x0008	$r7 = 0x4000056A, $r8 = 0x40000572
lw $r5, $r7, 0x0001	$r7 = 0x4000056A, $r5 = 0x4000056B
lw $r10, $r1, 0x0003	$r1 = 0x40000ABA, $r10 = 0x40000ABD

SW - Store Word



Format: sw $rt, 16-bit immediate($rs)

Purpose: To store a 32-bit immediate value to a memory location from adding the contents of another register and a sign extended 16-bit immediate.

Description: M[$rs + 16-bit immediate] ← $rt

Restriction: The 16-bit immediate value can only have a value from -32768 to 32767.

Exceptions: No Exceptions

Examples:

sw $r8, $r7, 0x000A	$r7 = 0x40000560, $r8 = 0x4000056A
sw $r6, $r5, 0x0001	$r5 = 0x4000000A, $r6 = 0x4000000B
sw $r3, $r1, 0x0003	$r1 = 0x4000089A, $r3 = 0x4000089D

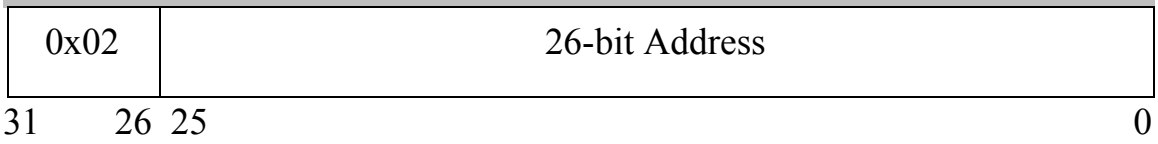
J - Type Instruction Set

Jump-Type

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00

opcode	26-bit jump address

J - Jump



Format: j Label

Purpose: To jump to another memory address

Description: PC ← { PC[31:28], Label[25:0], 2'b0 }

The updated 32-bit address that PC will jump to starts with the 4 upper bits of the current PC, then followed by the 26-bit Label and ended with 2 bit 0's.

Restriction: Since the 26-bit value is signed, the instruction has a limit how many times the PC can jump towards an instruction address between a value of **-33554432 to -33554431**

Operation:

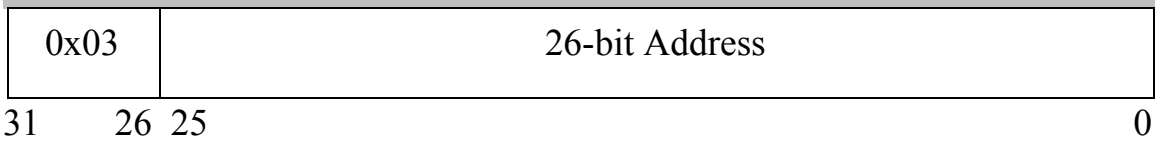
```
Label:    sub $r5, $r6, $r10
          j   Label
```

In this operation, the PC will jump back one instruction before which is the subtraction instruction.

Exceptions: No Exceptions

Examples: See **Operation**

JAL - Jump and Link



Format: jal Label

Purpose: A variation of the jump instruction where instead it will jump and save the current PC to the $ra(return address) that will run a function at a different memory address and jump back to the $ra after it is finished executing.

Description: $ra ← PC, then PC ← { PC[31:28], Label[25:0], 2'b0 }

The return address will get the current PC. The updated 32-bit address that PC will jump to starts with the 4 upper bits of the current PC, then followed by the 26-bit Label and ended with 2 bit 0's.

Restriction: Since the 26-bit value is signed, the instruction has a limit how many times the PC can jump towards an instruction address between a value of **-33554432 to -33554431**

Operation:

```
Label:  sub  $r5,  $r6, $r10
        shl  $r12, $r1, $r5
```

```
addi $r9,   $r8, 0x0001

add $r5, $zero, $r2
jal   Label
srl  $r7, $r7, 8
```

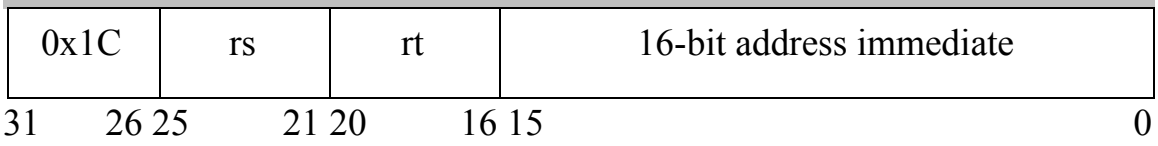
In this operation, the once the Label function is finished the PC will jump towards the address containing the shift right logical instruction.

Exceptions: No Exceptions

Examples: See **Operation**

4. Enhanced Instructions

INPUT



Format: INPUT $rt offset($rs)

Purpose: To receive an input from input and loaded to $rt

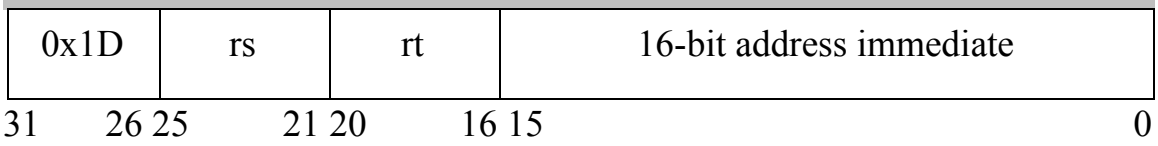
Description: The 32-bit value stored in $rt is summed along with the 16-bit sign-extended value produces the effective address and copies to $rt. The 32-bit value is now the memory location to be stored in.

Restriction: None

Examples:

addi $r1, $zero, 0x0000_0004	$r1 = 0x0000_0004
input $r4,0($r1)	$r4 = M[0x0000_0004]

OUTPUT



Format: OUTPUT $rt immediate($rs)

Purpose: To store a word from pointer to the memory.

Description: M[$rs + immediate] ← $rt

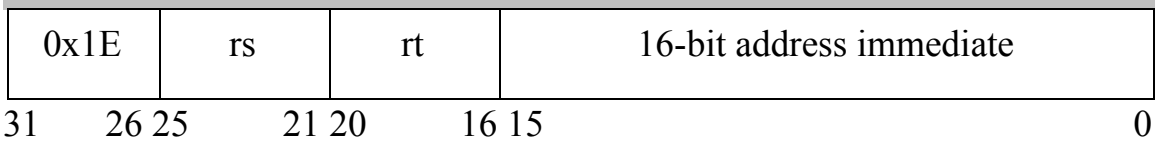
The 32-bit value stored in $rt is summed along with the 16-bit sign-extended value produces the effective address and copies to $rt. The 32-bit value is now the memory location to be stored in.

Restriction: None

Examples:

addi $r1, $zero, 0x0000_0004	$r1 = 0x0000_0004
output $r4,0($r1)	M[$r1] = 0x0000_0004

RETI - Return From Interrupt



Format: RETI

Purpose: To return from an interrupt service routine to the previous location .

Description: PC ← M[$sp]

The program copies the current stack pointer, then pop the top of stack twice for status flags. After all, return PC.

Restriction: None

Operation:

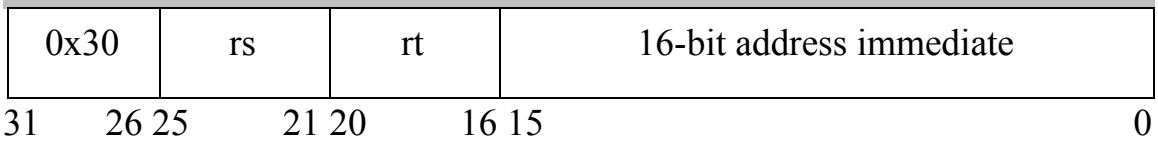
Fetch: PC ← PC +4;

RETI: ALU_OUT← $rs($sp)

Examples:

0x0000_0000	//interrupt
0x0000_0100	//return
RETI	

BLT - Branch Less Than



Format: BLT $rs, $rt, 16-bit offset

Purpose: To branch to a specific location if $rs < $rt

Description: if($rs < $rt) PC ← PC + 4 + 16-bit immediate

The content of $rs and $rt are subtracted. If the negative flag is high, the address is calculated and branched to.

Restriction: None

Operation: Functions as conditional if/else statement.

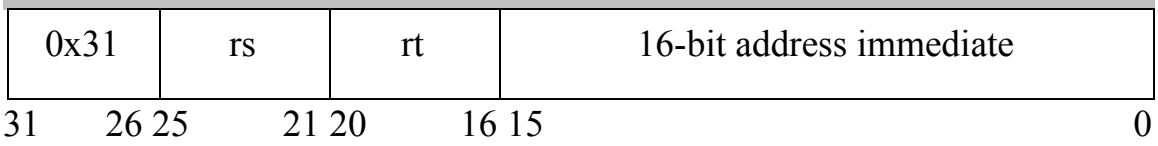
if($rs < $rt) PC ← PC + 4 + Branch Address

else PC ← PC + 4

Example:

addi $r6, $r0, 08	// $r6 = 0x000000008
addi $r1, $r0, 01	// $r2 = 0x000000001
blt $r6, $r1, 5	// $r1< $r6 therefore it branches to branch address

BGE - Branch Greater than or Equal



Format: BLT $rs, $rt, 16-bit offset

Purpose: To branch to a specific location if $rs > $rt or $rs = $rt

Description: if($rs > $rt || $rs == $rt) PC ← PC + 4 + 16-bit immediate

The content of $rs and $rt are subtracted. If the negative flag is low or the zero flags is high, the address is calculated and branched to.

Restriction: None

Operation: Functions as conditional if/else statement.

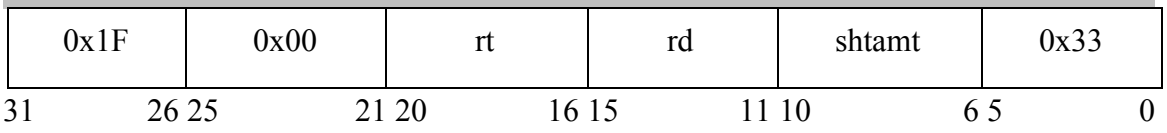
if($rs > $rt || $rs ==$ rt) PC ← PC + 4 + Branch Address

else PC ← PC + 4

Examples:

addi $r6, $r0, 05	// $r6 = 0x000000005
addi $r2, $r0, 07	// $r2 = 0x000000007
blt $r6, $r2, 2	// $r2> $r6 therefore it branches to branch address

CLR - Clear Register



Format: CLR $rt

Purpose: To clear the ALU_Out Register.

Description: ALU_Out ← 0x0

Load all zeros into ALU_Out

Restriction: None

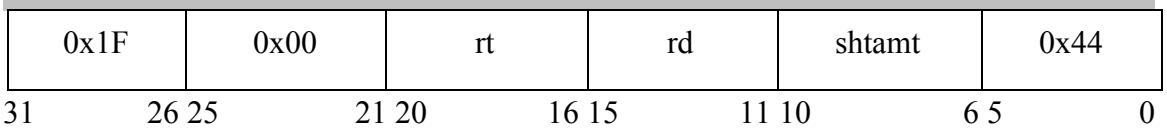
Operation: CLR: ALU_OUT ← 0x0;

CLR2: $rt ← ALU_OUT(0x0)

Examples:

clr $r5	$r5 = 0x0
clr $r9	$r9 = 0x0

MOV - Move



Format: CLR

Purpose: To move the content of one register to another.

Description: $rt ← $rs

Moves $rs into $rt

Restriction: None

Operation: MOV: ALU_OUT ← RS($rs)

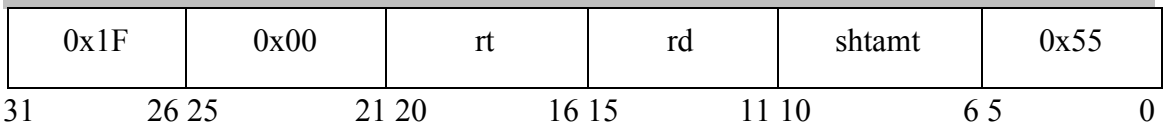
MOV2: $rt ← ALU_OUT($rs)

Examples

lui $r8, 0x0001	$r8 ← 0x0001
ori $r8, 0x0001	$r8 ← 0x00010001

mov $r3, $r8	$r3 ← 0x00010001

NOP - No Operation



Format: NOP

Purpose: To provide one clock tick with no instruction preformed.

Description: The value of the ALU_OUT register is passed back into same register for

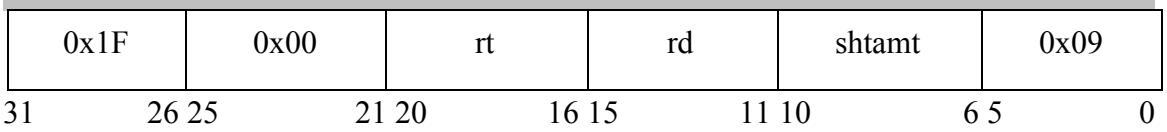
Restriction: None

Operation: ALU_OUT ← ALU_OUT(ALU_OUT)

Examples

NOP	
NOP	
NOP	// three clock ticks have passed

PUSH



Format: PUSH $rt

Purpose: To push a 32-bit value from $rt into the stack.

Description: dM[$sp] ← $rt

Pushes $rt into the stack and pre-decrements before updating the the $sp into the stack

Restriction: None

Operation: PUSH: RS ← RF[$sp], RT ← $rt

PUSH2: ALU_OUT ← RS($sp) - 4

PUSH3: dM[ALU_OUT($sp-4)] ← RT($rt)

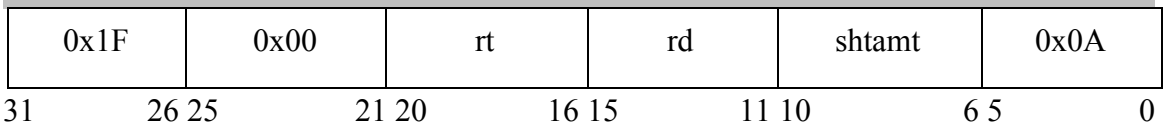
PUSH4: $sp ← ALU_OUT($sp)

Example:

lui $r5, 0xABCD	

ori $r5, 0xABCD	// $r5 ← 0xABCD_ABCD
push $r5	// dM[3FC] ← 0xABCD_ABCD

POP



Format: POP $rt

Purpose: To push a 32-bit value from the stack into the $rt.

Description: $rt ← dM[$sp]

Pops the top of stack into $rt and post-decrements updating the the $sp after popping.

Restriction: None

Operation: POP: RS <- RF[$sp]

POP2: ALU_OUT <- RS($sp)

POP3: D_in <- dM[ALU_OUT($sp)]

POP4: $rt <- D_in, ALU_OUT <- ALU_OUT($sp) + 4

POP5: $sp <- ALU_OUT

Example:

lui $r5, 0xABCD	
ori $r5, 0xABCD	// $r5 ← 0xABCD_ABCD
push $r5	// dM[3FC] ← 0xABCD_ABCD
pop $r11	// $r11 ← 0xABCD_ABCD

Instruction Formats

---

1. Register-Type

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00

0x1F	rs code	rt code	0x0	0x0	0x77

2. Immediate-Type

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00

opcode	rs code	rt code	16-bit immediate

3. Jump-Type

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00

opcode	26-bit jump address

Interrupts

The CPU that executes instructions in the MIPS ISA will have one external interrupt(**INTR**) input and one interrupt acknowledge(**INTA**) output. When a external interrupt is executed, the processor must retain the state of the machine by **pushing/storing** the **PC(Program Counter)** and **Flags** registers on top of the stack. For the 440 MIPS ISA simulation/verification, the PC will be loaded with an "interrupt vector" stored in the last location of Data Memory(i.e. Address 0x3FC)

Example: PC ← dM[0x3FC]
*retrieve interrupt vector

The "return from interrupt" instruction must **pop/loaded** in reverse order, the **Flags** register and **PC** from off the **stack pointer(SP).**

Reset

The purpose of reset is to initialize hex values within the Program Counter(PC), Stack Pointer(SP), and Flags(Carry, Overflow, Negative, Zero) registers within the processor. Therefore in response to an external reset input, the processor will execute the following minimum:

PC ← 0x00000000

$sp ← 0x000003FC

Flags(IE, N, Z, O, C) ← 5'b00000

When reset goes active, the PC register will be at 0x0, stack pointer($sp) will contain 0x000003FC and all the flag registers will contain 0x0 as well. Usually when a reset is active you want the registers to go the the starting point or be low active/off which is

typically a zero value. Having external reset can also help clear any registers within the processor at any time during its fetch, decode, and execute cycles.

References

2019 R.W Allison Website, CD-Rom Material and Lecture Slides

Patterson and Hennessy, Computer Organization and Design 4th edition

Intel® 64 and IA-32 Architectures Software Developer's Manual

MIPS IV Instruction Set

III. Verilog Implementation/ Design/ Verification

A. Top Level source code

```
`timescale ins / ips
/***************************** C E C S  4 4 0 ****************************
 * File Name:  MIPS_CPU_TB.v
 * Project:    Final Project
 * Designers:  TrieuVy Le and Mark Aquiapao
 * Rev. No.:   Version 1.0
 * Rev. Date:  April 22, 2019
   Rev. No.:   Version 1.1(4/13): Debugged testing out the first module
   Rev. No.:   Version 1.1(4/20): Rewired due to the addition of I/O Memory
 *
 * Purpose: This modules serves as the testbench that verifies the performance of
 *              MIPS CPU that includes the Control Unit, Instruction Unit,
 *              Integer Datapath, IO Memory, and Data Memory.
 *          We start off with the reset to get the MCU module going.
 *
 * The control unit will execute the machine instructions load from .dat files,
 * eventually will end with the BREAK instruction.
 *
 *  - Interrupt: we will introduce the interrupt once after a certain amount of
 * instructions are executed. Once received, wait for posedge clk of inta.
 * We can turn off and print all registers, PC, and IR.
 *
 *  WHEN RUNNING SIMULATION, PLEASE TURN YOUR TIME SCALE TO 10.00 us
 *
 * ©R.W Allison 2019
 ********************************************************************************/
module MIPS_CPU_TB;

    // Inputs
    reg clk;
    reg reset;

    // Outputs
    wire intr, inta;            // Interrupt request and acknowledge

    wire [31:0] Address ;       // From CPU to access both DM and IM
    wire [31:0] D_OUT;          // CPU output to access both memories
    wire [31:0] DM_out;         // data out from both memories

    // Data Memory output
    wire dm_cs;
    wire dm_rd;
    wire dm_wr;

    // I/O Memory output
    wire io_cs;
    wire io_rd;
    wire io_wr;

    // Instantiate the Unit Under Test (UUT)
    CPU cpu  (      .clk(clk),
                    .reset(reset),
                    .intr(intr),
                    .inta(inta),
                    .dm_cs(dm_cs), .dm_rd(dm_rd), .dm_wr(dm_wr),
                    .io_cs(io_cs), .io_rd(io_rd), .io_wr(io_wr),

                    .DM_out(DM_out),                //DM bus signal
                    .Address(Address),              //Bus to access both memories
                    .D_OUT(D_OUT));                 // ALU result to access both memories


    Data_Memory dm(
                    .clk(clk),
                    .Address(Address[11:0]),        // Extract bottom 12 bits
                    .DM_In(D_OUT),                  // Address result from CPU
                    .dm_cs(dm_cs),
                    .dm_wr(dm_wr),
                    .dm_rd(dm_rd),
```

```
70                      .DM_Out(DM_out));               // Results from both memories
71
72      IOMemory io(
73                      .clk(clk),
74                      .Address({20'h0, Address[11:0]}),   // only bottom 12 bits
75                      .IO_In(D_OUT),
76                      .io_cs(io_cs),
77                      .io_wr(io_wr),
78                      .io_rd(io_rd),
79                      .intr(intr),
80                      .inta(inta),
81
82                      .IO_Out(DM_out));
83
84
85
86      // Generate 10ns clock period
87      always #5 clk = ~clk;
88
89      // Set time format and input signals to start
90      initial begin
91          $timeformat(-9, 1, " ns", 9);
92          clk    = 0;
93          reset  = 1;
94
95          // Read both Instruction and Data Memory .dat files
96
97          // Instruction Memory
98          @(negedge clk)
99          $readmemh("iMem14_Sp19.dat", cpu.iu.im.IMem);
100
101         //Data Memory
102         @(negedge clk)
103         $readmemh("dMem14_Sp19.dat", dm.DataMem);
104
105         /*****************************************************************
106         * Testing Interrupt
107         * We will turn on the request after certain amount of time has propagated
108         * executed a few instructions from IM and DM.
109         * We then instroduced the interrupt request, once received
110         * the program will automatically acknowledge and we will wait for the posedge
111         * of inta from the MCU then turn off the request from here.
112         *****************************************************************/
113
114
115         @(negedge clk)
116         reset  = 0;
117         $display(" ");
118         $display("*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*");
119         $display("******CECS 440 FINAL MIPS PROJECT RESULTS**********");
120         $display("*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*");
121         $display(" ");
122     end
123
124 endmodule
125
126
```

B. Verilog Modules source code

```
`timescale 1ns / 1ps
/***************************** C E C S  4 4 0 ****************************
 * File Name:  CPU.v
 * Project:    Final Project
 * Designers:  Mark Aquiapao and TrieuVy Le
 * Rev. No.:   Version 1.0
 * Rev. No.:   Version 1.1: port mapped the barrel shifter and signals
 * Rev. No.:   Version 1.2: added flags stored in stack pointer register
 * and select signals for those regsiters
 * Rev. No.:   Version 1.3: added the IO memory module contains the interrupt

 * Rev. Date:  April 22, 2019
 *
 * Purpose: This modules serves as the Top Level of the Integer Datapath,
 *             Data Memory, IO Memory, MCU, and Instruction Unit.
 *
 * Notes: The Address going into the Data Mmeory will only take 12 bits of
 *       32-bit ALU_Out coming out of the Integer Datapath.

 * ©R.W Allison 2019
 *********************************************************************************/
module CPU(     clk, reset,
                DM_out, Address, D_OUT,
                intr, inta,
                dm_cs, dm_rd, dm_wr,
                io_cs, io_rd, io_wr     );

    input           clk, reset;
    input  [31:0]   DM_out;
    input           intr;


    output          inta;
    output [31:0]   Address, D_OUT;
    output          dm_cs, dm_rd, dm_wr, io_cs, io_rd, io_wr;

    wire            pc_ld, pc_inc, ir_ld;
    wire            im_cs, im_rd, im_wr;
    wire            intr, inta;

    wire            D_En,   HILO_ld;
    wire            SP_Sel, S_Sel;
    wire [1:0]      T_Sel,  pc_sel, DA_sel;
    wire [2:0]      Y_Sel;
    wire [4:0]      FS, in_flagsToMCU, out_flagsFromMCU;


    wire            C, N, V, Z;
    wire [31:0]     IR_out, PC_out, SE_16;


    // Instantiation of MCU, IU, IDP
    MCU mcu (   .clk(clk),
                .reset(reset),
                .intr(intr),
                .C(C), .N(N), .Z(Z), .V(V),
                .IR(IR_out),
                .inta(inta),

                //Program Counter signals
                .pc_sel(pc_sel),
                .pc_ld(pc_ld),
                .pc_inc(pc_inc),
                .ir_ld(ir_ld),

                //Instruction Memory signals
                .im_cs(im_cs),
                .im_rd(im_rd),
                .im_wr(im_wr),
```

```
70
71
72
73                  // 5-bit status flags and selects
74                  .stack_InFlags(in_flagsToMCU),
75                  .stack_OutFlags(out_flagsFromMCU),
76                  .sp_sel(sp_sel),
77                  .s_sel(s_sel),
78
79                  // IDP signals
80                  .D_En(D_En),
81                  .DA_sel(DA_sel),
82                  .T_Sel(T_Sel),
83                  .HILO_ld(HILO_ld),
84                  .Y_Sel(Y_Sel),
85                  .FS(FS),
86
87                  // Data Memorysignals
88                  .dm_cs(dm_cs),
89                  .dm_rd(dm_rd),
90                  .dm_wr(dm_wr),
91
92                  // I/O Memory signals
93                  .io_cs(io_cs),
94                  .io_rd(io_rd),
95                  .io_wr(io_wr));
96
97
98      Instruction_Unit iu(
99                  .clk(clk),
100                 .reset(reset),
101
102                 .PC_sel(pc_sel),
103                 .PC_ld(pc_ld),
104                 .PC_inc(pc_inc),
105                 .ir_ld(ir_ld),
106
107                 // Instruction Memory signals
108                 .im_cs(im_cs),
109                 .im_wr(im_wr),
110                 .im_rd(im_rd),
111
112                 .PC_In(Address),
113                 .PC_out(PC_out),
114                 .IR_out(IR_out),
115                 .SE_16(SE_16));
116
117     IntegerDatapath idp(
118                 .clk(clk),
119                 .reset(reset),
120                 .D_En(D_En),
121                 .DA_sel(DA_sel),
122                 .shtamt(IR_out[10:6]),
123                 .D_Addr(IR_out[15:11]),
124                 .T_Addr(IR_out[20:16]),
125                 .S_Addr(IR_out[25:21]),
126
127                 .DT(SE_16),
128                 .stack_InFlags(out_flagsFromMCU),
129                 .stack_OutFlags(in_flagsToMCU),
130                 .sp_sel(sp_sel),
131                 .s_sel(s_sel),
132                 .T_Sel(T_Sel),
133                 .FS(FS),
134                 .HILO_ld(HILO_ld),
135                 .DY(DM_out),
136                 .PC_In(PC_out),
137                 .Y_Sel(Y_Sel),
138                 .C(C), .V(V), .N(N), .Z(Z),


139                 .ALU_OUT(Address),
140                 .D_OUT(D_OUT));
141
142 endmodule
143
```

```
`timescale 1ns / 1ps
/****************************** C E C S 4 4 0 ******************************
 *
 * File Name:  MCU.v
 * Project:    Final Project
 * Designers:  Mark Aquiapao and TrieuVy Le
 * Rev. No.:   1.13
 * Version 1.1  (3/29): TrieuVy added barrel and flags
 * Version 1.2  (4/3) : Mark added SLL, SRL, SRA, ADDU, SUB, SUBU, SLT, SLTU, MUL,
 *                      DIV, SETIE
 * Version 1.3  (4/4) : TrieuVy added JR, LW, BEQ, BNE, MUL, DIV, J, JAL, MFHI, MFLO,

 * Version 1.4  (4/11): TrieuVy added SETIE, BLEZ, BGEZ
 * Version 1.5  (4/13): Mark edited and organized parameter values of states
 * Version 1.6  (4/15): Updated the flags with the barrel shifter and added the signals
 * Version 1.7  (4/16): Tested out the 2, 3, 4 modules for the shifting instructions.
   Wiring/port mapping issue really affected output
 * Version 1.8  (4/17): Debugged and tested for memory modules 4 to 9 because they
   all use the same DataMem
 * Debugged control signals for some new instructions
 * Version 1.9  (4/18): Commented most of the instructions.
   Also added a few more instructions for enhancements.
 * Version 1.10 (4/19): Rearranged the formatting of the instructions due to the fact
 * it is too clustered, needs to be organized better.
 * Version 1.11 (4/21): Debugged SETIE due to the introduction of the instruction
   in the memory.
 * Version 1.12 (4/22): Added the RETI instruction which takes in total of 8 cycles
   We saved the updated PC and flags to stack pointers and return at appropriate cycle



 * Rev. Date:  April 22, 2019
 * Purpose:    A state machine implementing the MIPS Control Unit (MCU)
 *             for the major cycles of fetch, execute and some MIPS instructions
 *             from memory, including checking for interrupts.
 *             Each state represents one clock cyle where EVERY control signal
 *             is updated at the negative edge of the clock.
 *
 * Notes:       This control unit is a modified MOORE FSM that will change states
 *              on the rising edge of the clock.
 *
 * If using INTERRUPT then we will print out all 32 registers otherwise
 * if not, we will print out the first 16 registers.
 *
 ********************************************************************************

 *-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
 * MCU CONTROL WORDS TEMPLATE FOR EVERY SINGLE CYCLE
 *-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
 *
 *   inta = 0;
 *   FS   = sltu;
 *   {pc_sel, pc_ld, pc_inc, ir_ld}        = 5'b00_0_0_0;
 *   {im_cs, im_rd, im_wr}                 = 3'b0_0_0;
 *   {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel} = 9'b0_00_00_0_000;
 *   {dm_cs, dm_rd, dm_wr}                 = 3'b0_0_0;
 *   {io_cs, io_rd, io_wr}                 = 3'b0_0_0;
 *   {sp_sel, s_sel}                       = 2'b0_0;
 *
 *-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*/


module MCU (clk, reset, intr,                   // system inputs
            C, N, Z, V,                         // ALU status inputs
            IR,                                 // Instruction Register input
            inta,                               // output to I/O subsystem
            pc_sel,  pc_ld,  pc_inc, ir_ld,     // rest of control word fields
            im_cs,   im_rd,  im_wr,             // instruction mem control signals
```

```
70              D_En,                                   // enable to write data to reg signal
71              DA_sel,                                 // destination register to write to 2 bit
72              T_Sel,                                  // select either T, DT, PC_IN, or flags
73              HILO_ld,                                // enables to write data to HI or LO regs
74
75              // selects 1 of 5 preset 32-bit values between ALU_OUT, HI, LO, D_IN, PC_IN
76              Y_Sel,
77              // selects value for S register, ALU_OUT or RS
78              s_sel,
79              // write to S_Addr with either $r29/$ra or S_Addr
80              sp_sel,
81              // carries flags from IDP
82              // after calculations, then transfer to MCU for control words
83              stack_InFlags, stack_OutFlags,
84
85              dm_cs,   dm_rd,  dm_wr,                 // data memory control signals
86              io_cs,   io_rd,  io_wr,                 // instruction memory control signals
87              FS);                                    // function select depends on operations
88  //*****************************************************************************
89
90      // Initialize input, output, integer for port mapping purposes
91      input               clk, reset, intr;           // system clock, reset, intr
92      input               C, N, Z, V;                 // Integer ALU status inputs
93      input    [4:0]      stack_InFlags;              // out from IDP to MCu
94      input    [31:0]     IR;                         // Instruction register
95
96      output   reg        inta;                       // Interrupt acknowledge
97
98      // Program Counter Controls
99      output   reg        pc_ld, pc_inc, ir_ld;       // Program Counter Register
100
101     //Instruction Memory Controls
102     output   reg        im_cs, im_rd,  im_wr;       // Instruction Memory
103
104     // Data Memory Controls
105     output   reg        dm_cs, dm_rd,  dm_wr;       // Data Memory
106
107     // IO Module Controls
108     output   reg        io_cs,   io_rd,  io_wr;     // Input/Output Module
109
110     // Integer Datapath (IDP) Controls
111     output   reg        D_En, HILO_ld;              // Register File
112     output   reg [1:0]  pc_sel, DA_sel, T_Sel;      // T-MUX and DA_MUX for T or D_Addr
113     output   reg [2:0]  Y_Sel;                      // ALU_Out select
114     output   reg [4:0]  FS;                         // Function select
115     output   reg        s_sel, sp_sel;              // Stack pointer and S_Addr selects
116     output   reg [4:0]  stack_OutFlags;             // flags received from IDP operations
117
118     // Loop counter, Data and Instruction Memory variables
119     integer i, DM_Dump, IOM_Dump;
120
121
122
123     // Function select assignments
124     parameter [4:0]          pass_s  = 5'h00,
125                              pass_t  = 5'h01,
126                              add     = 5'h02,
127                              addu    = 5'h03,
128                              sub     = 5'h04,
129                              subu    = 5'h05,
130                              slt     = 5'h06,
131                              sltu    = 5'h07,
132                              mul     = 5'h1E,
133                              div     = 5'h1F,
134
135                              and_op  = 5'h08,
136                              or_op   = 5'h09,
137                              xor_op  = 5'h0A,
138                              nor_op  = 5'h0B,
```

```
139
140                             srl     = 5'h0C,
141                             sra     = 5'h0D,
142                             sll     = 5'h0E,
143
144                             andi    = 5'h16,
145                             ori     = 5'h17,
146                             lui     = 5'h18,
147                             xori    = 5'h19,
148
149                             inc     = 5'h0F,
150                             inc4    = 5'h10,
151                             dec     = 5'h11,
152                             dec4    = 5'h12,
153                             zeros   = 5'h13,
154                             ones    = 5'h14,
155                             sp_init = 5'h15;
156
157
158
159
160       // state assignments
161        parameter
162           RESET   = 00,   FETCH  = 01,  DECODE = 02,
163
164           // R-Types
165           SLL     = 03,   SRL    = 04,  SRA    = 05,
166           ADD     = 06,   ADDU   = 07,  SUB    = 08,
167           SUBU    = 09,   AND    = 10,  OR     = 11,
168           NOR     = 12,   XOR    = 13,   SLT   = 14,
169           SLTU    = 15,   MUL    = 16,  DIV    = 17,
170           MFHI    = 29,   MFLO   = 30,  SETIE  = 18,
171
172           // I-Types
173           ORI     = 19,   LUI    = 20,  LW     = 21,
174           LW2     = 22,   SW     = 23,  XORI   = 24,
175           ADDI    = 25,   SLTI   = 26,  SLTIU  = 27,
176           ANDI    = 28,
177
178           // Jumps
179           JR = 31, JR2 = 32,
180           J  = 35, JAL = 36,
181
182           // Branches
183           BEQ    = 37, BEQ2  = 38,
184           BNE    = 39, BNE2  = 40,
185           BLEZ   = 41, BLEZ2 = 42,
186           BGTZ   = 43, BGTZ2 = 44,
187           BLT    = 62, BLT2  = 63,
188           BGE    = 64, BGE2  = 65,
189
190           // Write Backs
191
192           WB_alu  = 45,  WB_imm = 46,  WB_Din = 47,
193           WB_hi   = 48,  WB_lo  = 49,  WB_mem = 50,
194
195           // Interrupts and Others
196           INTR1   = 501, INTR2     = 502, INTR3 = 503,
197           INTR4   = 504, INTR5     = 505, INTR6 = 506,
198           BREAK   = 510, ILLEGAL_OP = 511,
199
200
201           //Enhancement type instructions
202           INPUT  = 51, INPUT2  = 52,
203           OUTPUT = 53, OUTPUT2 = 54,
204           RETI   = 56, RETI2   = 57, RETI3 = 58,
205           RETI4  = 59, RETI5   = 60, RETI6 = 61,
206           CLR    = 66, CLR2    = 67,
207           MOV    = 68, MOV2    = 69,
```

```
208           NOP    = 70,
209           PUSH   = 71, PUSH2   = 72, PUSH3 = 73, PUSH4 = 74,
210           POP    = 75, POP2    = 76, POP3  = 77, POP4  = 78,   POP5 = 79;
211
212
213       //state register (up to 512 states)
214       reg [8:0] state;
215
216      /*****************
217       * Flags register *
218       *****************/
219      reg   psi, psc, psv, psn, psz;   // flags present state registers
220      reg   nsi, nsc, nsv, nsn, nsz;   // flags next state registers
221
222      // Updating flags register
223      always @(posedge clk, posedge reset)
224         if(reset)
225            {psi, psc, psv, psn, psz} = 5'b0;
226         else
227            {psi, psc, psv, psn, psz} = {nsi, nsc, nsv, nsn, nsz};
228
229       /*********************************************
230        * 440 MIPS CONTROL UNIT (Finite State Machine) *
231        *********************************************/
232       always @(posedge clk, posedge reset)
233         if (reset)
234            begin
235               // ALU_Out <- 32'h3FC
236               @(negedge clk)
237                   inta = 0;   FS = sp_init;
238                   {pc_sel, pc_ld, pc_inc, ir_ld}        = 5'b00_0_0_0;
239                   {im_cs, im_rd, im_wr}                 = 3'b0_0_0;
240                   {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel} = 9'b0_00_00_0_000;
241                   {dm_cs, dm_rd, dm_wr}                 = 3'b0_0_0;
242                   {io_cs, io_rd, io_wr}                 = 3'b0_0_0;
243                   {sp_sel, s_sel}                       = 2'b0_0;
244                #1 {nsi, nsc, nsv, nsn, nsz}             = 5'b0;
245                  state = RESET;
246            end
247         else
248            case (state)
249               FETCH:
250                   if ((inta==0 & (intr==1 & psi==1))) // Recieve Interrupt Signal
251                     begin //*** new interrupt pending; prepare for ISR ***
252                       // control word assignments for "deasserting" everything
253
254                        @(negedge clk)
255                        inta = 0;   FS = 5'h0;
256                        {pc_sel, pc_ld, pc_inc, ir_ld}        = 5'b00_0_0_0;
257                        {im_cs, im_rd, im_wr}                 = 3'b0_0_0;
258                        {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel} = 9'b0_00_00_0_000;
259                        {dm_cs, dm_rd, dm_wr}                 = 3'b0_0_0;
260                        {io_cs, io_rd, io_wr}                 = 3'b0_0_0;
261                        {sp_sel, s_sel}                       = 2'b0_0;
262                     #1 {nsi, nsc, nsv, nsn, nsz}             = {psi, psc, psv, psn, psz};
263                        state = INTR1;
264                     end
265                   else  // No Interrupt Signal involved
266                     begin //*** no new interrupt pending; fetch and instruction ***
267                       if(((inta==1 & intr==1) || (psi==1 & intr==0))) inta=1'b0;
268                       // control word assignments for IR <- iM[PC]; PC <- PC+4
269
270                       @(negedge clk)
271                       inta = 0;   FS = 5'h0;
272                       {pc_sel, pc_ld, pc_inc, ir_ld}        = 5'b00_0_1_1;
273                       {im_cs, im_rd, im_wr}                 = 3'b1_1_0;
274                       {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel} = 9'b0_00_00_0_000;
275                       {dm_cs, dm_rd, dm_wr}                 = 3'b0_0_0;
276                       {io_cs, io_rd, io_wr}                 = 3'b0_0_0;
```

```
277                 {sp_sel, s_sel}                         = 2'b0_0;
278              #1  {nsi, nsc, nsv, nsn, nsz}               = {psi, psc, psv, psn, psz};
279                   state = DECODE;
280                end
281
282           RESET:
283              begin
284                // control word assignments for $sp <- ALU_Out(32'h3FC)
285                @(negedge clk)
286                inta=0;   FS=5'h0;
287                {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
288                {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
289                {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b1_11_00_0_000;
290                {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
291                {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
292                {sp_sel, s_sel}                         = 2'b0_0;
293                #1  {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
294                state = FETCH;
295              end
296
297           DECODE:
298              begin
299                @(negedge clk)
300            // check for MIPS format
301            // R Type
302                if (IR[31:26] == 6'h0)
303                  begin
304                // it is an R-type format
305                  // control word assignments: RS <- $rs, RT <- $rt (default)
306                  inta = 0;   FS = 5'h0;
307                 {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
308                 {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
309                 {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
310                 {dm_cs, dm_rd, dm_wr}                     = 3'b0_0_0;
311                 {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
312                 {sp_sel, s_sel}                         = 2'b0_0;
313              #1 {nsi, nsc, nsv, nsn, nsz}               = {psi, psc, psv, psn, psz};
314                // check for function for R type
315                case (IR[5:0])
316                  6'h00 :  state = SLL;
317                  6'h02 :  state = SRL;
318                  6'h03 :  state = SRA;
319                  6'h08 :  state = JR;
320                  6'h10 :  state = MFHI;
321                  6'h12 :  state = MFLO;
322                  6'h18 :  state = MUL;
323                  6'h1A :  state = DIV;
324                  6'h20 :  state = ADD;
325                  6'h21 :  state = ADDU;
326                  6'h22 :  state = SUB;
327                  6'h23 :  state = SUBU;
328                  6'h24 :  state = AND;
329                  6'h25 :  state = OR;
330                  6'h26 :  state = XOR;
331                  6'h27 :  state = NOR;
332                  6'h2A :  state = SLT;
333                  6'h2B :  state = SLTU;
334                  6'h0D :  state = BREAK;
335                  6'h1F :  state = SETIE;
336                 default: state = ILLEGAL_OP;
337                    endcase
338                end // end of if for R-type Format
339
340            else if(IR[31:26] == 6'h1F)
341                  begin
342                 /*************************************************************
343                 * Enhancement Type instructions
344                 * Since it is still decode state, the default is to load $rs and $rt
345                 *************************************************************/
```

```
                    inta = 0;   FS = 5'h0;
                   {pc_sel, pc_ld, pc_inc, ir_ld}         = 5'b00_0_0_0;
                   {im_cs, im_rd, im_wr}                  = 3'b0_0_0;
                   {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}  = 9'b0_00_00_0_000;
                   {dm_cs, dm_rd, dm_wr}                  = 3'b0_0_0;
                   {io_cs, io_rd, io_wr}                  = 3'b0_0_0;
                   {sp_sel, s_sel}                        = 2'b0_0;
                 #1 {nsi, nsc, nsv, nsn, nsz}             = {psi, psc, psv, psn, psz};

                   // The available states for the enhancement type
                   case (IR[5:0])
                      6'h33  : state = CLR;
                      6'h44  : state = MOV;
                      6'h55  : state = NOP;
                      6'h66  : state = PUSH;
                      6'h77  : state = POP;
                      default: state = ILLEGAL_OP;
                   endcase
                 end  // end of if E-key format
                   else
                      begin
                   // control word assignments: RS <- $rs, RT <- DT(se_16)
                     inta = 0;   FS = 5'h0;
                    {pc_sel, pc_ld, pc_inc, ir_ld}         = 5'b00_0_0_0;
                    {im_cs, im_rd, im_wr}                  = 3'b0_0_0;
                    {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}  = 9'b0_00_01_0_000;
                    {dm_cs, dm_rd, dm_wr}                  = 3'b0_0_0;
                    {io_cs, io_rd, io_wr}                  = 3'b0_0_0;
                    {sp_sel, s_sel}                        = 2'b0_0;
                 #1 {nsi, nsc, nsv, nsn, nsz}             = {psi, psc, psv, psn, psz};

                   // Check opcode for I and J type
                        case (IR[31:26])
                     6'h02 : state = J;
                     6'h03 : state = JAL;
                     6'h04 : state = BEQ;
                     6'h05 : state = BNE;
                     6'h06 : state = BLEZ;
                     6'h07 : state = BGTZ;
                     6'h30 : state = BLT;
                     6'h31 : state = BGE;
                     6'h08 : state = ADDI;
                     6'h0A : state = SLTI;
                     6'h0B : state = SLTIU;
                     6'h0C : state = ANDI;
                     6'h0D : state = ORI;
                     6'h0E : state = XORI;
                     6'h0F : state = LUI;
                     6'h23 : state = LW;
                     6'h2B : state = SW;
                     6'h1C : state = INPUT;
                     6'h1D : state = OUTPUT;
                     6'h1E : state = RETI;

                    default: state = ILLEGAL_OP;
                         endcase

                 /*****************************************************************
                 * Calculations for the branches.
                 * T_Sel selects 0: $RT <= $rt, take the IR[15:0] imme16 to calculate
                 * branch address because if any branch states, the default is to grab T
                 * and calculate branch address.
                 *****************************************************************/

                   if(state == BEQ || state == BNE || state == BLEZ|| state == BGTZ||
                      state == BLT || state == BGE  )
                      T_Sel = 2'b00;
                   else
                      T_Sel = 2'b01;
```

```
415
416                     end
417                 end // end of DECODE
418
419     /*************************************************************************
420     *                    R TYPE INSTRUCTIONS
421     *************************************************************************/
422             SLL:
423                 begin
424                   // control word assignments: ALU_Out <- $rt << shamt
425                   @(negedge clk)
426                     inta=0;  FS=sll;
427                    {pc_sel, pc_ld, pc_inc, ir_ld}         = 5'b00_0_0_0;
428                    {im_cs, im_rd, im_wr}                  = 3'b0_0_0;
429                    {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}  = 9'b0_00_00_0_000;
430                    {dm_cs, dm_rd, dm_wr}                  = 3'b0_0_0;
431                    {io_cs, io_rd, io_wr}                  = 3'b0_0_0;
432                    {sp_sel, s_sel}                        = 2'b0_0;
433                 #1 {nsi, nsc, nsv, nsn, nsz}              = {psi, psc, psv, psn, psz};
434                    state = WB_alu;
435                 end
436
437             SRL:
438                 begin
439                   // control word assignments: ALU_Out <- $rt >> shamt
440                   @(negedge clk)
441                    inta=0;  FS=srl;
442                    {pc_sel, pc_ld, pc_inc, ir_ld}         = 5'b00_0_0_0;
443                    {im_cs, im_rd, im_wr}                  = 3'b0_0_0;
444                    {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}  = 9'b0_00_00_0_000;
445                    {dm_cs, dm_rd, dm_wr}                  = 3'b0_0_0;
446                    {io_cs, io_rd, io_wr}                  = 3'b0_0_0;
447                    {sp_sel, s_sel}                        = 2'b0_0;
448                 #1 {nsi, nsc, nsv, nsn, nsz}              = {psi, psc, psv, psn, psz};
449                    state = WB_alu;
450                 end
451
452             SRA:
453                 begin
454                   // control word assignments: ALU_Out <- $rt >> shamt
455                   inta = 1'b0; FS = sra;
456                   @(negedge clk)
457                   {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
458                   {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
459                   {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
460                   {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
461                   {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
462                   {sp_sel, s_sel}                         = 2'b0_0;
463               #1 {nsi, nsc, nsv, nsn, nsz} = {psi, C, V, N, Z};
464                   state = WB_alu;
465                 end
466
467
468
469              ADD:
470                 begin
471                   // control word assignments: ALU_Out <- $rs + $rt
472                   @(negedge clk)
473                   inta=0;    FS=add;
474                   {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
475                   {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
476                   {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
477                   {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
478                   {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
479                   {sp_sel, s_sel}                         = 2'b0_0;
480               #1 {nsi, nsc, nsv, nsn, nsz} = {psi, C, V, N, Z};
481                   state = WB_alu;
482                 end
483
```

```
484                 ADDU:
485                     begin
486                       // control word assignments: ALU_Out <- $rs + $rt
487                       @(negedge clk)
488                       inta=0;    FS=addu;
489                       {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
490                       {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
491                       {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
492                       {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
493                       {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
494                       {sp_sel, s_sel}                         = 2'b0_0;
495                     #1 {nsi, nsc, nsv, nsn, nsz} = {psi, C, V, N, Z};
496                       state = WB_alu;
497                     end
498
499                 SUB:
500                     begin
501                       // control word assignments: ALU_Out <- $rs - $rt
502                       @(negedge clk)
503                       inta=0;    FS=sub;
504                       {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
505                       {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
506                       {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
507                       {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
508                       {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
509                       {sp_sel, s_sel}                         = 2'b0_0;
510                     #1 {nsi, nsc, nsv, nsn, nsz} = {psi, C, V, N, Z};
511                       state = WB_alu;
512                     end
513
514                 SUBU:
515                     begin
516                       // control word assignments:
517                       @(negedge clk)
518                       // control word assignments: ALU_Out <- $rs - $rt
519                       @(negedge clk)
520                       inta=0;    FS=subu;
521                       {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
522                       {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
523                       {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
524                       {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
525                       {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
526                       {sp_sel, s_sel}                         = 2'b0_0;
527                     #1 {nsi, nsc, nsv, nsn, nsz} = {psi, C, V, N, Z};
528                       state = WB_alu;
529                     end
530
531                 AND:
532                     begin
533                       // control word assignments: ALU_Out <- $rs & $rt
534                       @(negedge clk)
535                       inta=0;    FS=and_op;
536                       {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
537                       {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
538                       {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
539                       {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
540                       {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
541                       {sp_sel, s_sel}                         = 2'b0_0;
542                     #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
543                       state = WB_alu;
544                     end
545
546                 OR:
547                     begin
548                       // control word assignments: ALU_Out <- $rs | $rt
549                       @(negedge clk)
550                       inta=0;    FS=or_op;
551                       {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
552                       {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
```

```
553                 {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
554                 {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
555                 {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
556                 {sp_sel, s_sel}                         = 2'b0_0;
557               #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
558                  state = WB_alu;
559                end
560
561             NOR:
562                begin
563                  // control word assignments: ALU_Out <- ~($rs | $rt)
564                  @(negedge clk)
565                  inta=0;    FS=nor_op;
566                  {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
567                  {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
568                  {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
569                  {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
570                  {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
571                  {sp_sel, s_sel}                         = 2'b0_0;
572               #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
573                  state = WB_alu;
574                end
575
576             XOR:
577                begin
578                  // control word assignments: ALU_Out <- $rs ^ $rt
579                  @(negedge clk)
580                  inta=0;    FS=xor_op;
581                  {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
582                  {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
583                  {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
584                  {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
585                  {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
586                  {sp_sel, s_sel}                         = 2'b0_0;
587               #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
588                  state = WB_alu;
589                end
590
591
592
593             SLT:
594                begin
595                  // control word assignments: ALU_Out <- $rs < $rt ? 1 : 0
596                  @(negedge clk)
597                  inta=0;    FS=slt;
598                  {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
599                  {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
600                  {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
601                  {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
602                  {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
603                  {sp_sel, s_sel}                         = 2'b0_0;
604               #1 {nsi, nsc, nsv, nsn, nsz} = {psi, C, V, N, Z};
605                  state = WB_alu;
606                end
607
608             SLTU:
609                begin
610                  // control word assignments: ALU_Out <- $rs < $rt ? 1 : 0
611                  @(negedge clk)
612                  inta=0;    FS=sltu;
613                  {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
614                  {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
615                  {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
616                  {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
617                  {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
618                  {sp_sel, s_sel}                         = 2'b0_0;
619               #1 {nsi, nsc, nsv, nsn, nsz} = {psi, C, V, N, Z};
620                  state = WB_alu;
621                end
```

```
622
623
624                 MUL:
625                     begin
626                       // control word assignments: {Hi,Lo} <- R[$rs] * R[$rt]
627                       @(negedge clk)
628                       inta=0;    FS=mul;
629                       {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
630                       {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
631                       {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_1_000;
632                       {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
633                       {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
634                       {sp_sel, s_sel}                         = 2'b0_0;
635                     #1 {nsi, nsc, nsv, nsn, nsz} = {psi, C, V, N, Z};
636                        state = FETCH;
637                     end
638
639                 DIV:
640                     begin
641                       // control word assignments:
642                       // Lo <- R[$rs] / R[$rt], Hi <- R[$rs] % R[$rt]
643                       @(negedge clk)
644                       inta=0;    FS=div;
645                       {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
646                       {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
647                       {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_1_000;
648                       {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
649                       {io_cs, io_rd, io_wr}                      = 3'b0_0_0;
650                       {sp_sel, s_sel}                         = 2'b0_0;
651                     #1 {nsi, nsc, nsv, nsn, nsz} = {psi, C, V, N, Z};
652                        state = WB_alu;
653                     end
654
655                 MFHI:
656                     begin
657                       // control word assignments: R[$rd] <- Hi
658
659                       @(negedge clk)
660                       inta=0;    FS=5'h0;
661                       {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
662                       {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
663                       {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b1_00_00_0_001;
664                       {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
665                       {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
666                       {sp_sel, s_sel}                         = 2'b0_0;
667                     #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
668                        state = FETCH;
669                     end
670
671                 MFLO:
672                     begin
673                       // control word assignments: R[$rd] <- Lo
674                       @(negedge clk)
675                       inta=0;    FS=5'h0;
676                       {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
677                       {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
678                       {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b1_00_00_0_010;
679                       {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
680                       {io_cs, io_rd, io_wr}                      = 3'b0_0_0;
681                       {sp_sel, s_sel}                         = 2'b0_0;
682                     #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
683                        state = FETCH;
684                     end
685
686
687
688                 SETIE:
689                     begin
690                       // control word assignments: psi <- 1'b1
```

```
691                   @(negedge clk)
692                   inta=0;    FS=5'h0;
693                   {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
694                   {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
695                   {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
696                   {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
697                   {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
698                   {sp_sel, s_sel}                         = 2'b0_0;
699                 #1 {nsi, nsc, nsv, nsn, nsz} = {1'b1, psc, psv, psn, psz};
700                    state = FETCH;
701                  end
702
703     /*************************************************************************
704     *                     I TYPE INSTRUCTIONS
705     *************************************************************************/
706
707
708          ORI:
709              begin
710              // ALU_OUT <= RS($rs) | {16'h0, RT(SE_16)}
711                   @(negedge clk)
712                   inta=0;    FS=ori;
713                   {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
714                   {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
715                   {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
716                   {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
717                   {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
718                   {sp_sel, s_sel}                         = 2'b0_0;
719                 #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
720                    state = WB_imm;
721                  end
722
723            XORI:
724              begin
725              // ALU_OUT <= RS($rs) ^ {16'h0, RT(SE_16)}
726                   @(negedge clk)
727                   inta=0;    FS=xori;
728                   {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
729                   {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
730                   {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
731                   {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
732                   {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
733                   {sp_sel, s_sel}                         = 2'b0_0;
734                 #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
735                    state = WB_imm;
736                  end
737
738            LUI:
739              begin
740              // ALU_OUT <= {RT(SE_16), 16'h0}
741                   // control word assignments for ALU_Out <- { RT[15:0], 16'h0}
742                   @(negedge clk)
743                   inta=0;    FS=lui;
744                   {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
745                   {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
746                   {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
747                   {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
748                   {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
749                   {sp_sel, s_sel}                         = 2'b0_0;
750                 #1  {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
751                    state = WB_imm;
752                  end
753
754            LW:
755              begin
756              // Effective Address calculation
757              // ALU_OUT <= RS($rs) + RT(SE_16)
758                   @(negedge clk)
759                   inta=0;    FS=add;
```

```
760                 {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
761                 {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
762                 {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
763                 {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
764                 {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
765                 {sp_sel, s_sel}                         = 2'b0_0;
766               #1 {nsi, nsc, nsv, nsn, nsz} = {psi, C, V, N, Z};
767                  state = LW2;
768                end
769            LW2:
770              begin
771              // Read the memory content and store in D_IN
772              // D_IN <= M[ALU_OUT]
773                   @(negedge clk)
774                   inta=0;    FS=5'h0;
775                   {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
776                   {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
777                   {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
778                   {dm_cs, dm_rd, dm_wr}                   = 3'b1_1_0;
779                   {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
780                   {sp_sel, s_sel}                         = 2'b0_0;
781               #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
782                  state = WB_Din;
783                end
784
785              SW:
786                  begin
787                  // Effective Address calculation
788                  // ALU_OUT <= RS($rs) + RT(SE_16)
789                    @(negedge clk)
790                    inta=0;    FS=add;
791                    {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
792                    {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
793                    {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
794                    {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
795                    {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
796                    {sp_sel, s_sel}                         = 2'b0_0;
797               #1 {nsi, nsc, nsv, nsn, nsz} = {psi, C, V, N, Z};
798                    state = WB_mem;
799                  end
800
801              ADDI:
802              begin
803              // ALU_OUT <= RS($rs) + RT(SE_16)
804                   @(negedge clk)
805                   inta=0;    FS=add;
806                   {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
807                   {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
808                   {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
809                   {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
810                   {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
811                   {sp_sel, s_sel}                         = 2'b0_0;
812               #1 {nsi, nsc, nsv, nsn, nsz} = {psi, C, V, N, Z};
813                  state = WB_imm;
814                end
815
816            SLTI:
817              begin
818              // This instruction compares if ( RS($rs) < RT(SE_16) )? 1:0;
819              // ALU_OUT <= ( RS($rs) < RT(SE_16) )? 1:0
820                   @(negedge clk)
821                   inta=0;    FS=slt;
822                   {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
823                   {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
824                   {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
825                   {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
826                   {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
827                   {sp_sel, s_sel}                         = 2'b0_0;
828                   #1 {nsi, nsc, nsv, nsn, nsz} = {psi, C, V, N, Z};
```

```
829                 state = WB_imm;
830               end
831
832            SLTIU:
833              begin
834              // This instruction compares if ( RS($rs) < RT(unsigned(SE_16)) )? 1:0;
835              // ALU_OUT <= ( RS($rs) < RT(SE_16) )? 1:0
836                   @(negedge clk)
837                   inta=0;    FS=sltu;
838                   {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
839                   {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
840                   {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
841                   {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
842                   {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
843                   {sp_sel, s_sel}                         = 2'b0_0;
844                #1 {nsi, nsc, nsv, nsn, nsz} = {psi, C, V, N, Z};
845                   state = WB_imm;
846                 end
847
848            ANDI:
849              begin
850              // ALU_OUT <= RS($rs) & RT(SE_16)
851                   @(negedge clk)
852                   inta=0;    FS=andi;
853                   {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
854                   {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
855                   {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
856                   {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
857                   {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
858                   {sp_sel, s_sel}                         = 2'b0_0;
859                #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
860                   state = WB_imm;
861                 end
862
863
864          /*************************************************************************
865           *                  J TYPE INSTRUCTIONS
866           *************************************************************************/
867
868
869
870            J:
871               begin
872                     // PC <= PC + SE_16(IR[25:0]) << 2 (Calculate the effective address
873                     @(negedge clk)
874                     inta=0;    FS=5'h0;
875                     {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b01_1_0_0;
876                     {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
877                     {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
878                     {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
879                     {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
880                     {sp_sel, s_sel}                         = 2'b0_0;
881                  #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
882                     state = FETCH;
883                   end
884
885            JAL:
886               begin
887               // Update PC with the calculation of Jump effective address and also
888               // save the current PC in $r31
889               // PC  <= PC + SE_16(IR[25:0]) << 2
890               // $ra <= PC
891                     @(negedge clk)
892                     inta=0;    FS=5'h0;
893                     {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b01_1_0_0;
894                     {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
895                     {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b1_10_00_0_100;
896                     {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
897                     {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
```

```
                {sp_sel, s_sel}                          = 2'b0_0;
            #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
               state = FETCH;
              end

          JR:
              begin
                // control word assignments: PC <- [$rt]
                @(negedge clk)
                 inta=0;   FS=5'h0;
                 {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
                 {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
                 {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
                 {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
                 {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
                 {sp_sel, s_sel}                         = 2'b0_0;
            #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
               state = JR2;
              end

          JR2:
              begin
                // control word assignments: PC <- [$rt]
                @(negedge clk)
                 inta=0;   FS=5'h0;
                 {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_1_0_0;
                 {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
                 {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
                 {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
                 {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
                 {sp_sel, s_sel}                         = 2'b0_0;
            #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
               state = FETCH;
              end
        /*************************************************************************
        *                    Branches INSTRUCTIONS
        *************************************************************************/

         BEQ:
            begin
               // The comparison using subtraction
                // ALU_OUT <= RS - RT
                @(negedge clk)
                inta=0;    FS=sub;
                {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
                {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
                {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
                {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
                {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
                {sp_sel, s_sel}                         = 2'b0_0;
                #1  {nsi, nsc, nsv, nsn, nsz} = {psi, C, V, N, Z};
                state = BEQ2;
              end
         BEQ2:
            begin
            // If zero flag is 1 meaning s = t, then branch to the label address
            // else go back to fetch state.
            // PC <= PC + SE_16(IR[25:0] << 2)
                inta=0;   FS=5'h00;
                if(psz == 1)
                       {pc_sel, pc_ld, pc_inc, ir_ld}   = 5'b10_1_0_0;
                else
                       {pc_sel, pc_ld, pc_inc, ir_ld}   = 5'b00_0_0_0;
                {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
                {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
                {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
                {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
                {sp_sel, s_sel}                         = 2'b0_0;
              #1    {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
```

```
967                     state = FETCH;
968                end
969
970             BNE:
971                begin
972                // The comparison using subtraction
973                     // ALU_OUT <= RS - RT
974                  @(negedge clk)
975                  inta=0;    FS=sub;
976                  {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
977                  {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
978                  {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
979                  {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
980                  {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
981                  {sp_sel, s_sel}                         = 2'b0_0;
982                  #1  {nsi, nsc, nsv, nsn, nsz} = {psi, C, V, N, Z};
983                  state = BNE2;
984                    end
985             BNE2:
986                begin
987                // If zero flag is 0 meaning s != t, then branch to the label address
988                // else go back to fetch state.
989                // PC <= PC + SE_16(IR[25:0] << 2)
990
991                     @(negedge clk)
992                                              inta=0;   FS=5'h00;
993                     if(psz == 1'b0)
994                            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b10_1_0_0;
995                     else
996                            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
997                     {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
998                     {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
999                     {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
1000                    {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
1001                    {sp_sel, s_sel}                         = 2'b0_0;
1002                    #1  {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
1003                    state = FETCH;
1004                   end
1005
1006            BLEZ:
1007               begin
1008                  // The comparison using subtraction
1009                     // ALU_OUT <= RS - RT
1010                     @(negedge clk)
1011                     inta=0;    FS=sub;
1012                     {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
1013                     {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
1014                     {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
1015                     {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
1016                     {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
1017                     {sp_sel, s_sel}                         = 2'b0_0;
1018                     #1  {nsi, nsc, nsv, nsn, nsz} = {psi, C, V, N, Z};
1019                     state = BLEZ2;
1020                   end
1021            BLEZ2:
1022               begin
1023               // In this case, we are checking for RS against RT(0x0000_0000) if RS = 0's
1024               // then we will check for both senarios:
1025               // Less than would result in a negative
1026               // Equal would result in a zero
1027               // So if either both is active then branch to the effective address.
1028               // PC <= PC + SE_16(IR[25:0] << 2)
1029                 @(negedge clk)
1030                     inta=0;   FS=5'h00;
1031                     if(psn == 1 || psz == 1)
1032                            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b10_1_0_0;
1033                     else
1034                            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
1035                     inta=0;    FS=sub;
```

```
1036
1037                {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
1038                {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
1039                {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
1040                {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
1041                {sp_sel, s_sel}                         = 2'b0_0;
1042                state = FETCH;
1043              end
1044
1045          BGTZ:
1046             begin
1047                // The comparison using subtraction
1048                // ALU_OUT <= RS - RT
1049                @(negedge clk)
1050                inta=0;    FS=sub;
1051                {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
1052                {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
1053                {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
1054                {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
1055                {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
1056                {sp_sel, s_sel}                         = 2'b0_0;
1057                #1  {nsi, nsc, nsv, nsn, nsz} = {psi, C, V, N, Z};
1058                state = BGTZ2;
1059              end
1060          BGTZ2:
1061             begin
1062             // In this case, we are checking for RS against RT(0x0000_0000) if RS = 0's
1063             // then we will check for both senarios:
1064             // Greater than would result in a positive, meaning negative flag stays low
1065             // Equal would result in a zero
1066             // So if either senarios is true then branch to the effective address.
1067             // PC <= PC + SE_16(IR[25:0] << 2)
1068                @(negedge clk)
1069                inta=0;   FS=5'h00;
1070                if(psn == 0 || psz == 1)
1071                       {pc_sel, pc_ld, pc_inc, ir_ld}   = 5'b10_1_0_0;
1072                else
1073                       {pc_sel, pc_ld, pc_inc, ir_ld}   = 5'b00_0_0_0;
1074                {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
1075                {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
1076                {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
1077                {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
1078                {sp_sel, s_sel}                         = 2'b0_0;
1079                #1  {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
1080                state = WB_imm;
1081              end
1082
1083          BLT:
1084                begin
1085                // The comparison using subtraction
1086                // ALU_OUT <= RS - RT
1087                  @(negedge clk)
1088                  inta=0;    FS=sub;
1089                  {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
1090                  {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
1091                  {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
1092                  {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
1093                  {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
1094                  {sp_sel, s_sel}                         = 2'b0_0;
1095               #1 {nsi, nsc, nsv, nsn, nsz} = {psi, C, V, N, Z};
1096                  state = BLT2;
1097                end
1098          BLT2:
1099                begin
1100                  // If negative flag is 1 meaning s < t, then branch to the ea
1101                  // PC <= PC + SE_16 << 2
1102                  @(negedge clk)
1103                  inta=0;    FS=5'h00;
1104                  if (psn)
```

```
1105                            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b10_1_0_0;
1106                    else
1107                            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
1108                    {im_cs, im_rd, im_wr}                  = 3'b0_0_0;
1109                    {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}  = 9'b0_00_00_0_000;
1110                    {dm_cs, dm_rd, dm_wr}                  = 3'b0_0_0;
1111                    {io_cs, io_rd, io_wr}                  = 3'b0_0_0;
1112                    {sp_sel, s_sel}                        = 2'b0_0;
1113                 #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
1114                    state = FETCH;
1115                  end
1116
1117            BGE:
1118                  begin
1119                    // The comparison using subtraction
1120                    // ALU_OUT <= RS - RT
1121                    @(negedge clk)
1122                    inta=0;    FS=sub;
1123                    {pc_sel, pc_ld, pc_inc, ir_ld}         = 5'b00_0_0_0;
1124                    {im_cs, im_rd, im_wr}                  = 3'b0_0_0;
1125                    {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}  = 9'b0_00_00_0_000;
1126                    {dm_cs, dm_rd, dm_wr}                  = 3'b0_0_0;
1127                    {io_cs, io_rd, io_wr}                  = 3'b0_0_0;
1128                    {sp_sel, s_sel}                        = 2'b0_0;
1129                 #1 {nsi, nsc, nsv, nsn, nsz} = {psi, C, V, N, Z};
1130                    state = BGE2;
1131                  end
1132            BGE2:
1133                  begin
1134                    // If zero flag is 1 meaning s = t,
1135                    // forcing to check for less than case, then branch to the ea
1136                    // PC <= PC + SE_16 << 2
1137                    @(negedge clk)
1138                    inta=0;    FS=5'h00;
1139                    if(psn == 0 || psz == 1)
1140                            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b10_1_0_0;
1141                    else
1142                            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
1143                    {pc_sel, pc_ld, pc_inc, ir_ld}         = 5'b00_0_0_0;
1144                    {im_cs, im_rd, im_wr}                  = 3'b0_0_0;
1145                    {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}  = 9'b0_00_00_0_000;
1146                    {dm_cs, dm_rd, dm_wr}                  = 3'b0_0_0;
1147                    {io_cs, io_rd, io_wr}                  = 3'b0_0_0;
1148                    {sp_sel, s_sel}                        = 2'b0_0;
1149                 #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
1150                    state = FETCH;
1151                  end
1152
1153          /*************************************************************************
1154          *                    Write Backs INSTRUCTIONS
1155          *************************************************************************/
1156                WB_alu:
1157                   begin
1158                   // R($rd) <= ALU_OUT
1159                     @(negedge clk)
1160                     inta=0;    FS=5'h0;
1161                     {pc_sel, pc_ld, pc_inc, ir_ld}         = 5'b00_0_0_0;
1162                     {im_cs, im_rd, im_wr}                  = 3'b0_0_0;
1163                     {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}  = 9'b1_00_00_0_000;
1164                     {dm_cs, dm_rd, dm_wr}                  = 3'b0_0_0;
1165                     {io_cs, io_rd, io_wr}                  = 3'b0_0_0;
1166                     {sp_sel, s_sel}                        = 2'b0_0;
1167                  #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
1168                     state = FETCH;
1169                   end
1170
1171                WB_imm:
1172                   begin
1173                   // R($rt) <= ALU_OUT
```

```
1174                    @(negedge clk)
1175                    inta=0;    FS=5'h0;
1176                    {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
1177                    {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
1178                    {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b1_01_00_0_000;
1179                    {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
1180                    {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
1181                    {sp_sel, s_sel}                         = 2'b0_0;
1182                 #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
1183                    state = FETCH;
1184                  end
1185
1186               WB_mem:
1187                  begin
1188                  // M[ALU_OUT($rs + SE_16) <= RT($rt)
1189                    @(negedge clk)
1190                    inta=0;    FS=5'h0;
1191                    {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
1192                    {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
1193                    {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
1194                    {dm_cs, dm_rd, dm_wr}                   = 3'b1_0_1;
1195                    {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
1196                    {sp_sel, s_sel}                         = 2'b0_0;
1197                 #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
1198                    state = FETCH;
1199                  end
1200
1201               WB_Din:
1202                  begin
1203                  // R($rt) <= D_IN(M[ALU_OUT])
1204                        @(negedge clk)
1205                        inta=0;    FS=5'h0;
1206                        {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
1207                        {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
1208                        {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b1_01_00_0_011;
1209                        {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
1210                        {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
1211                        {sp_sel, s_sel}                         = 2'b0_0;
1212                     #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
1213                        state = FETCH;
1214                      end
1215
1216
1217        /*************************************************************************
1218        *                    Interrupt and other INSTRUCTIONS
1219        *************************************************************************/
1220                  INTR1:
1221                     begin
1222                     // $ra <= PC and RS <= $sp
1223                     // The return addr will get the copy of program counter ($r31)
1224                     // The RS register will get the content of stack pointer ($r29)
1225                       @(negedge clk)
1226                       inta=0;    FS=5'h0;
1227                       {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
1228                       {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
1229                       {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b1_10_00_0_100;
1230                       {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
1231                       {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
1232                       {sp_sel, s_sel}                         = 2'b1_0;
1233                    #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
1234                     state = INTR2;
1235                     end
1236                  INTR2:
1237                     begin
1238
1239                       // The ALU_OUT register will get the content of stack pointer reg
1240                       // ALU_OUT <= $r29
1241                       @(negedge clk)
1242                       inta=0;    FS=5'h0;
```

```
1243                    {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
1244                    {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
1245                    {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
1246                    {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
1247                    {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
1248                    {sp_sel, s_sel}                         = 2'b0_0;
1249                #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
1250                   state = INTR3;
1251                 end
1252           INTR3:
1253                 begin
1254                 // Since we have a fixed interrupt service routine addr of 0x3FC
1255                 // we can now read that address into D_IN
1256                 // D_IN <= M[ALU_OUT(0x3FC)]
1257                   @(negedge clk)
1258                   inta=0;    FS=5'h0;
1259                   {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
1260                   {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
1261                   {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
1262                   {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
1263                   {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
1264                   {sp_sel, s_sel}                         = 2'b0_0;
1265                #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
1266                   state = INTR4;
1267                 end
1268              INTR4:
1269                 begin
1270                 // Copy the content of the memory address of the stack pointer reg to PC
1271                 // ALU_OUT reg gets the pre-decremented by 4 of stack pointer reg
1272                 // RT register is loaded with the current PC
1273                 // PC <= D_IN(M[$sp]), ALU_OUT <= $sp - 4, $RT <= PC
1274                   @(negedge clk)
1275                   inta=0;    FS=dec4;
1276                   {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_1_0_0;
1277                   {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
1278                   {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_10_0_011;
1279                   {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
1280                   {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
1281                   {sp_sel, s_sel}                         = 2'b0_1;
1282                #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
1283                   state = INTR5;
1284                 end
1285           INTR5:
1286                 begin
1287                 // To make a PUSH instruction to PC onto the stack, $r29 gets decremented
1288                 // push the flags on stack too through RT register
1289                 // M[$r29] <= RT(PC), ALU_OUT <= ALU_OUT($r29 - 4) - 4, RT <= 5-bit Flags
1290                   @(negedge clk)
1291                   inta=0;    FS=dec4;
1292                   {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
1293                   {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
1294                   {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_11_0_000;
1295                   {dm_cs, dm_rd, dm_wr}                   = 3'b1_0_1;
1296                   {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
1297                   {sp_sel, s_sel}                         = 2'b0_1;
1298                #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
1299                   state = INTR6;
1300                 end
1301           INTR6:
1302                 begin
1303                 // Again we will push the flags like such {27'h0, 5-bit flags}
1304                 // The stack pointer reg gets the address of current stack pointer after
1305                 // being pushed twice then we can set the acknowledge of interrupt
1306                 // M[ALU_OUT($r29 - 8) <= RT({27'h0, 5-bit flags})
1307                 // R($29) <= ALU_OUT($r29 - 8) and inta = 1'b1
1308                   @(negedge clk)
1309                   inta=1'b1;     FS=add;
1310                   {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
1311                   {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
```

```
1312                    {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}    = 9'b1_11_00_0_000;
1313                    {dm_cs, dm_rd, dm_wr}                    = 3'b1_0_1;
1314                    {io_cs, io_rd, io_wr}                    = 3'b0_0_0;
1315                    {sp_sel, s_sel}                          = 2'b0_0;
1316                 #1 {nsi, nsc, nsv, nsn, nsz} = {1'b0, psc, psv, psn, psz};
1317                    state = FETCH;
1318                  end
1319
1320
1321             ILLEGAL_OP:
1322                   begin
1323                     $display("ILLEGAL OPCODE FETCHED %t",$time);
1324                     // control word assignments for "deasserting" everything
1325                     @(negedge clk)
1326                     inta=0;    FS=5'h0;
1327                     {pc_sel, pc_ld, pc_inc, ir_ld}           = 5'b00_0_0_0;
1328                     {im_cs, im_rd, im_wr}                    = 3'b0_0_0;
1329                     {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}    = 9'b0_00_00_0_000;
1330                     {dm_cs, dm_rd, dm_wr}                    = 3'b0_0_0;
1331                     {io_cs, io_rd, io_wr}                    = 3'b0_0_0;
1332                     {sp_sel, s_sel}                          = 2'b0_0;
1333                 #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
1334                     $display(" ");
1335                     $display("Memory:");
1336                     $display(" ");
1337                     Dump_Registers;
1338                     $display(" ");
1339                     Dump_PC_and_IR;
1340                   $finish;
1341                   end
1342
1343
1344             BREAK:
1345                 begin
1346                   $display("BREAK INSTRUCTION FETCHED %t",$time);
1347                   // control word assignments for "deasserting" everything
1348                   @(negedge clk)
1349                      inta=0;     FS=5'h0;
1350                      {pc_sel, pc_ld, pc_inc, ir_ld}           = 5'b00_0_0_0;
1351                      {im_cs, im_rd, im_wr}                    = 3'b0_0_0;
1352                      {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}    = 9'b0_00_00_0_000;
1353                      {dm_cs, dm_rd, dm_wr}                    = 3'b0_0_0;
1354                      {io_cs, io_rd, io_wr}                    = 3'b0_0_0;
1355                      {sp_sel, s_sel}                          = 2'b0_0;
1356                    #1 {nsi, nsc, nsv, nsn, nsz}               = {psi, psc, psv, psn, psz};
1357                   $display(" R E G I S T E R ' S  A F T E R  B R E A K");
1358
1359                   $display(" ");
1360                   $display("*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*");
1361                   $display(" Register File Content of $r0 - $r31 ");
1362                   $display("*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*");
1363                   $display(" ");
1364                   Dump_Registers; // task to output MIPS RegFile
1365
1366                   $display(" ");
1367                   $display("*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*");
1368                   $display("Memory Location at M[0x3FC]");
1369                   $display("*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*");
1370                   $display(" ");
1371                   $display("Time=%t M[0x3FC]=%h", $time, {MIPS_CPU_TB.dm.DataMem[12'h3F0],
1372                                                           MIPS_CPU_TB.dm.DataMem[12'h3F1],
1373                                                           MIPS_CPU_TB.dm.DataMem[12'h3F2],
1374                                                           MIPS_CPU_TB.dm.DataMem[12'h3F3]});
1375                   $display(" ");
1376                   $display("*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*");
1377                   $display("DATA MEMORY && I/O MEMORY of Memory locations 0xC0h to 0xFFh");
1378                   $display("*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*");
1379                   $display(" ");
1380                   Dump_Memory;
```

```
1381
1382                $finish;
1383             end
1384
1385
1386      /*****************************************************************************
1387       *                    ENHANCEMENT INSTRUCTIONS
1388       *****************************************************************************/
1389           INPUT:
1390                  begin
1391                    // ALU_OUT <= RS($rs) + RT(SE_16) to calculate the effective address.
1392                    @(negedge clk)
1393                    inta=0;    FS=add;
1394                    {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
1395                    {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
1396                    {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
1397                    {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
1398                    {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
1399                    {sp_sel, s_sel}                         = 2'b0_0;
1400                 #1 {nsi, nsc, nsv, nsn, nsz} = {psi, C, V, N, Z};
1401                    state = INPUT2;
1402                  end
1403           INPUT2:
1404                  begin
1405                    // D_IN <= I/OM[ALU_OUT]
1406                    @(negedge clk)
1407                    inta=0;    FS=5'h0;
1408                    {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
1409                    {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
1410                    {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
1411                    {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
1412                    {io_cs, io_rd, io_wr}                   = 3'b1_1_0;
1413                    {sp_sel, s_sel}                         = 2'b0_0;
1414                 #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
1415                    state = WB_Din;
1416                  end
1417
1418           OUTPUT:
1419                  begin
1420                    // ALU_OUT <= RS($rs) + RT(SE_16) to calculate the effective address.
1421                    @(negedge clk)
1422                    inta=0;    FS=add;
1423                    {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
1424                    {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
1425                    {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
1426                    {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
1427                    {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
1428                    {sp_sel, s_sel}                         = 2'b0_0;
1429                 #1 {nsi, nsc, nsv, nsn, nsz} = {psi, C, V, N, Z};
1430                    state = OUTPUT2;
1431                  end
1432           OUTPUT2:
1433                  begin
1434                    // I/OM[ALU_OUT] <= D_IN
1435                    @(negedge clk)
1436                    inta=0;    FS=5'h0;
1437                    {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
1438                    {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
1439                    {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
1440                    {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
1441                    {io_cs, io_rd, io_wr}                   = 3'b1_0_1;
1442                    {sp_sel, s_sel}                         = 2'b0_0;
1443                 #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
1444                    state = FETCH;
1445                  end
1446
1447           RETI:
1448                  begin
1449                  // ALU_OUT <= RS($r29)
```

```
1450                // Pass S, which is the current stack pointer register through the ALU
1451                  @(negedge clk)
1452                  inta=0;    FS=add;
1453                  {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
1454                  {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
1455                  {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
1456                  {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
1457                  {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
1458                  {sp_sel, s_sel}                         = 2'b0_0;
1459              #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
1460                 state = RETI2;
1461                end
1462           RETI2:
1463                begin
1464                // D_IN <= M[$r29], D_IN register carries the flags
1465                  @(negedge clk)
1466                  inta=0;    FS=5'h0;
1467                  {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
1468                  {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
1469                  {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
1470                  {dm_cs, dm_rd, dm_wr}                   = 3'b1_1_0;
1471                  {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
1472                  {sp_sel, s_sel}                         = 2'b0_0;
1473              #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
1474                 state = RETI3;
1475                end
1476          RETI3:
1477
1478                begin
1479                // Since we passed the status flags to the least 5 bit of D_IN, we can
1480                // collect the 5 bit and increment the stack pointer due to a POP
1481                // Y_MUX <= D_IN and ALU_OUT <= $r29 + 4
1482                  @(negedge clk)
1483                  inta=0;    FS=inc4;
1484                  {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
1485                  {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
1486                  {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
1487                  {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
1488                  {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
1489                  {sp_sel, s_sel}                         = 2'b0_1;
1490              #1 {nsi, nsc, nsv, nsn, nsz} = {stack_InFlags};
1491                 state = RETI4;
1492                end
1493           RETI4:
1494                begin
1495                // Pop the current stack of the $ra to D_IN and increment $r29
1496                // D_IN <= M[$r29 + 4] and ALU_OUT <= ALU_OUT($r29 + 4), because
1497                // POP is post increment so after every POP, we increment $sp
1498                  @(negedge clk)
1499                  inta=0;    FS=inc4;
1500                  {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
1501                  {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
1502                  {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
1503                  {dm_cs, dm_rd, dm_wr}                   = 3'b1_1_0;
1504                  {sp_sel, s_sel}                         = 2'b0_1;
1505                  {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
1506              #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
1507                 state = RETI5;
1508                end
1509          RETI5:
1510                begin
1511                // Load the current D_IN carries PC to Program Counter
1512                // PC <= D_IN and ALU_OUT <= $r29
1513                  @(negedge clk)
1514                  inta=0;    FS=5'h0;
1515                  {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_1_0_0;
1516                  {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
1517                  {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_011;
1518                  {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
```

```
1519                {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
1520                {sp_sel, s_sel}                         = 2'b0_1;
1521             #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
1522                state = RETI6;
1523              end
1524          RETI6:
1525              begin
1526              // WB the stack pointer popped twiced
1527              // $r29 <= ALU_OUT($r29)
1528                @(negedge clk)
1529                inta=0;    FS=5'h0;
1530                {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
1531                {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
1532                {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b1_11_00_0_000;
1533                {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
1534                {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
1535                {sp_sel, s_sel}                         = 2'b0_0;
1536             #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
1537                state = FETCH;
1538              end
1539
1540
1541
1542
1543          CLR:
1544              begin
1545                // ALU_OUT <= 0x0000_0000
1546                @(negedge clk)
1547                inta=0;    FS=zeros;
1548                {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
1549                {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
1550                {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
1551                {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
1552                {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
1553                {sp_sel, s_sel}                         = 2'b0_0;
1554             #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
1555                state = CLR2;
1556              end
1557          CLR2:
1558              begin
1559              // WB to T_Addr as the destination whatever the result from ALU_OUT
1560              // R($rt) <= ALU_OUT
1561                @(negedge clk)
1562                inta=0;    FS=5'h0;
1563                {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
1564                {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
1565                {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b1_01_00_0_000;
1566                {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
1567                {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
1568                {sp_sel, s_sel}                         = 2'b0_0;
1569             #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
1570                state = FETCH;
1571              end
1572
1573
1574
1575          MOV:
1576              begin
1577              // $rt <= $rs
1578              // ALU_OUT <= RS($rs)
1579                @(negedge clk)
1580                inta=0;    FS=5'h0;
1581                {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
1582                {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
1583                {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
1584                {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
1585                {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
1586                {sp_sel, s_sel}                         = 2'b0_0;
1587             #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
```

```
1588                    state = MOV2;
1589                end
1590            MOV2:
1591                begin
1592                  // R($rt) <= ALU_OUT($rs)
1593                  @(negedge clk)
1594                  inta=0;    FS=5'h0;
1595                  {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
1596                  {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
1597                  {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b1_01_00_0_000;
1598                  {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
1599                  {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
1600                  {sp_sel, s_sel}                         = 2'b0_0;
1601             #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
1602                  state = FETCH;
1603                end
1604
1605            NOP:
1606                begin
1607                  // Nothing happens during this clock, due to that fact the ALU result
1608                  // gets stored back to itself (register)
1609                  // control word assignments ALU_Out <- ALU_Out
1610                  @(negedge clk)
1611                  inta=0;    FS=5'h0;
1612                  {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
1613                  {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
1614                  {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
1615                  {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
1616                  {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
1617                  {sp_sel, s_sel}                         = 2'b0_1;
1618             #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
1619                  state = FETCH;
1620                end
1621
1622            PUSH:
1623                begin
1624                // Pre decrement the $r29 and save $rt content into that register
1625                // Load RS and RT
1626                // RS <= [$r29] , RT = $rt
1627                  @(negedge clk)
1628                  inta=0;    FS=5'h0;
1629                  {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
1630                  {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
1631                  {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
1632                  {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
1633                  {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
1634                  {sp_sel, s_sel}                         = 2'b0_0;
1635             #1 {nsi, nsc, nsv, nsn, nsz} = {psi, C, V, N, Z};
1636                  state = PUSH2;
1637                end
1638            PUSH2:
1639                begin
1640                // Push content onto stack, and the operation is to subtract 4 from
1641                // the stack address. Load RT with the appropriate content
1642                // ALU_OUT <= ($r29 - 4) and RT <= $rt
1643                  @(negedge clk)
1644                  inta=0;    FS=dec4;
1645                  {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
1646                  {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
1647                  {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
1648                  {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
1649                  {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
1650                  {sp_sel, s_sel}                         = 2'b0_0;
1651             #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
1652                  state = PUSH3;
1653                end
1654            PUSH3:
1655                begin
1656                // The current content in RT is stored into memory location
```

```
1657                // ALU_OUT reg also gets the stack pointer decremented by 4
1658                // M[ALU_OUT] <= RT and ALU_OUT <= $r29 - 4
1659                  @(negedge clk)
1660                  inta=0;    FS=5'h0;
1661                  {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
1662                  {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
1663                  {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
1664                  {dm_cs, dm_rd, dm_wr}                   = 3'b1_0_1;
1665                  {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
1666                  {sp_sel, s_sel}                         = 2'b0_1;
1667             #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
1668                 state = PUSH4;
1669                end
1670          PUSH4:
1671                begin
1672                // Stack pointer register gets the calculation of $r29 decremented by 4
1673                // $r29 <= ALU_OUT($r29 - 4)
1674                  @(negedge clk)
1675                  inta=0;    FS=5'h0;
1676                  {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
1677                  {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
1678                  {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b1_11_00_0_000;
1679                  {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
1680                  {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
1681                  {sp_sel, s_sel}                         = 2'b0_0;
1682             #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
1683                 state = FETCH;
1684                end
1685
1686          POP:
1687                begin
1688                  // R($rt) <= M[$r29] +  The post increment type of POP
1689                  // RS <= $r29
1690                  @(negedge clk)
1691                  inta=0;    FS=5'h0;
1692                  {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
1693                  {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
1694                  {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
1695                  {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
1696                  {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
1697                  {sp_sel, s_sel}                         = 2'b1_0;
1698             #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
1699                 state = POP2;
1700                end
1701          POP2:
1702                begin
1703                // ALU_OUT <= RS($r29)
1704                // The ALU_OUT register gets the calculation through the ALU
1705                // and passes on the data of stack pointer reg.
1706                  @(negedge clk)
1707                  inta=0;    FS=5'h0;
1708                  {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
1709                  {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
1710                  {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}   = 9'b0_00_00_0_000;
1711                  {dm_cs, dm_rd, dm_wr}                   = 3'b0_0_0;
1712                  {io_cs, io_rd, io_wr}                   = 3'b0_0_0;
1713                  {sp_sel, s_sel}                         = 2'b0_0;;
1714             #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
1715                 state = POP3;
1716                end
1717          POP3:
1718                begin
1719                // D_IN reg gets the memory location of ALU_OUT result.
1720                // ALU_OUT gets the calculation with the stack pointer.
1721                // D_IN <= M[ALU_OUT] and ALU_OUT <= ALU_OUT($r29)
1722                  @(negedge clk)
1723                  inta=0;    FS=5'h0;
1724                  {pc_sel, pc_ld, pc_inc, ir_ld}          = 5'b00_0_0_0;
1725                  {im_cs, im_rd, im_wr}                   = 3'b0_0_0;
```

```
1726                    {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}    = 9'b0_00_00_0_000;
1727                    {dm_cs, dm_rd, dm_wr}                    = 3'b1_1_0;
1728                    {io_cs, io_rd, io_wr}                    = 3'b0_0_0;
1729                    {sp_sel, s_sel}                          = 2'b0_1;
1730                 #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
1731                    state = POP4;
1732                 end
1733           POP4:
1734                 begin
1735                 // R($rt) <= D_IN  and ALU_OUT <= ALU_OUT($r29 + 4)
1736                 // D_IN data content gets copied to RT register,
1737                 // ALU_OUT loops back and increment stack pointer by 4.
1738                   @(negedge clk)
1739                   inta=0;    FS=inc4;
1740                   {pc_sel, pc_ld, pc_inc, ir_ld}           = 5'b00_0_0_0;
1741                   {im_cs, im_rd, im_wr}                    = 3'b0_0_0;
1742                   {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}    = 9'b1_01_00_0_011;
1743                   {dm_cs, dm_rd, dm_wr}                    = 3'b0_0_0;
1744                   {io_cs, io_rd, io_wr}                    = 3'b0_0_0;
1745                   {sp_sel, s_sel}                          = 2'b0_1;
1746                 #1 {nsi, nsc, nsv, nsn, nsz} = {psi, C, V, N, Z};
1747                    state = POP5;
1748                 end
1749           POP5:
1750                 begin
1751                 // The final calculation of ALU_OUT($r29 + 4) gets copied to $r29
1752                   // R($r29) <= ALU_OUT($r29 + 4)
1753                   @(negedge clk)
1754                   inta=0;    FS=5'h0;
1755                   {pc_sel, pc_ld, pc_inc, ir_ld}           = 5'b00_0_0_0;
1756                   {im_cs, im_rd, im_wr}                    = 3'b0_0_0;
1757                   {D_En, DA_sel, T_Sel, HILO_ld, Y_Sel}    = 9'b1_11_00_0_000;
1758                   {dm_cs, dm_rd, dm_wr}                    = 3'b0_0_0;
1759                   {io_cs, io_rd, io_wr}                    = 3'b0_0_0;
1760                   {sp_sel, s_sel}                          = 2'b0_0;
1761                 #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
1762                    state = FETCH;
1763                 end
1764
1765       endcase
1766
1767
1768        /*****************************************************************
1769        * DUMP REGISTERS TASK
1770        * Print the first 16 registers contents in the register file
1771        * We have to use the dot method to pinpoint exactly which signal we
1772        * would like to pull out from a module.
1773        *****************************************************************/
1774        task Dump_Registers;
1775
1776            begin
1777                for(i = 0; i < 16; i = i + 1)
1778                begin
1779                    $display ("Time=%t   $r%0d = %h  ||  Time=%t   $r%0d = %h",
1780                    $time, i,    MIPS_CPU_TB.cpu.idp.rf32.data[i],
1781                    $time, i+16, MIPS_CPU_TB.cpu.idp.rf32.data[i + 16]);
1782                 end
1783            end
1784        endtask
1785
1786        /*****************************************************************************
1787        * DUMP CURRENT PROGRAM COUNTER AND INSTRUCTION REGISTER TASK
1788        * The task to print out the current program counter and the instruction register
1789        * Once again we will print these signals with the dot method.
1790        *****************************************************************************/
1791        task Dump_PC_and_IR;
1792            begin
1793                $display(" ");
1794                $display("*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-");
```

```
            $display(" PC and IR Registers");
            $display("*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-");
            $display(" ");
            $display(" "); $display("PC Register:");
            $display("Time=%t PC=%h", $time, MIPS_CPU_TB.cpu.iu.pc.PC_out);
            $display(" ");
            $display("IR Register:");
            $display("Time=%t IR=%h", $time, MIPS_CPU_TB.cpu.iu.ir.Q);
            $display(" "); $display(" ");
        end
    endtask



   /*****************************************************************************
   * DUMP DATA AND INPUT/OUPUT INSTRUCTIONS TASK
   * The task asks to be printed out the input/output memory from the memory location
   * from the 9-bit address of 0xC0h - 0xFFh.
   *****************************************************************************/
   task Dump_Memory;
   begin
      $display("             Data Memory            " );
      for(i = 8'hC0; i <= 8'hFF; i = i + 4) begin
         DM_Dump  = {MIPS_CPU_TB.dm.DataMem[i],
                     MIPS_CPU_TB.dm.DataMem[i+1],
                     MIPS_CPU_TB.dm.DataMem[i+2],
                     MIPS_CPU_TB.dm.DataMem[i+3]};

         IOM_Dump = {MIPS_CPU_TB.io.IOMem[i],
                     MIPS_CPU_TB.io.IOMem[i+1],
                     MIPS_CPU_TB.io.IOMem[i+2],
                     MIPS_CPU_TB.io.IOMem[i+3]};
         $display("Time=%t DM[%h] = %h || Time=%t I/OM[%h] = %h",
         $time, i, DM_Dump, $time, i, IOM_Dump);
      end
   end
   endtask

endmodule
```

```
`timescale 1ns / 1ps
/***************************** C E C S  4 4 0 ****************************
 * File Name:  CPU_IU.v
 * Project:    Lab_Assignment_6
 * Designers:  Mark Aquiapao and TrieuVy Le
 * Rev. No.:   Version 1.0: Original
               Version 1.1: Lab 5
               Version 1.2: Lab 6, add PC_Sel to pass to PC_In
               Version 1.3: Updated the PC_Sel
 * Rev. Date:  April 22, 2019
 *
 * Purpose: This modules serves as the Instruction Unit that instantiates the
 *              Program Counter register, Instruction Memory, and the Instruction
 *              register.
 *
 *
 * ©R.W Allison 2019
 ************************************************************************/
module Instruction_Unit(clk, reset,
                        PC_ld, PC_inc, PC_In, PC_out, PC_sel,
                        im_cs, im_rd, im_wr,
                        ir_ld, IR_out,
                        SE_16);

        input           clk, reset, PC_ld, PC_inc, im_cs, im_rd, im_wr, ir_ld;
        input  [31:0]   PC_In;
        input  [1:0]    PC_sel;

        output [31:0]   PC_out, IR_out, SE_16;

        wire   [31:0]   IM_out, PC_MUX;

        // Calculate the effective address of either Branch or Jump
        assign PC_MUX =   (PC_sel==2'h0) ?  PC_In:
                          (PC_sel==2'h1) ?  {PC_out[31:28],IR_out[25:0],2'b00}:
                          (PC_sel==2'h2) ?  {PC_out + {SE_16[29:0],2'b00}}    :
                                            PC_In;


        // Program Counter to either inc or ld the EA to PC
        ProgramCounter pc    (   .clk(clk),
                                 .reset(reset),
                                 .PC_In(PC_MUX),
                                 .PC_ld(PC_ld),
                                 .PC_inc(PC_inc),
                                 .PC_out(PC_out) );

        // 4096x8 Instruction Memory
        InstructionMemory im (   .clk(clk),
                                 .Address(PC_out[11:0]),
                                 .IM_In(32'b0),
                                 .im_cs(im_cs),
                                 .im_wr(im_wr),
                                 .im_rd(im_rd),
                                 .IM_Out(IM_out));


        // Initialize Register
        InstructionRegister ir  (.clk(clk),
                                 .reset(reset),
                                 .load(ir_ld),
                                 .D(IM_out),
                                 .Q(IR_out)  );

        // Sign Extend 16-bit to 32-bit for I-Type Instructions for RT
        assign SE_16 = { {16{IR_out[15]}}, IR_out[15:0]};

endmodule

```

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Company:
4  // Engineer:
5  //
6  // Create Date:    15:37:59 03/03/2019
7  // Design Name:
8  // Module Name:    ProgramCounter
9  // Project Name:
10 // Target Devices:
11 // Tool versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////////////////////////////////////////////////////////
21 module ProgramCounter(clk, reset, PC_In, PC_ld, PC_inc, PC_out);
22
23     input clk, reset, PC_ld, PC_inc;
24     input [31:0] PC_In;
25     output reg [31:0] PC_out;
26
27     always @ (posedge clk, posedge reset) begin
28         if  (reset)  PC_out <= 32'b0;
29         else
30             case ({PC_inc, PC_ld})
31                 2'b01:  PC_out <= PC_In;
32                 2'b10:  PC_out <= PC_out + 4;
33                 default: PC_out <= PC_out;
34             endcase
35     end
36
37 endmodule
38
```

```
`timescale 1ns / 1ps
/***************************** C E C S  4 4 0 ****************************
 * File Name:  InstructionMemory.v
 * Project:    Lab 4
 * Designers:  Mark Aquiapao and TrieuVy Le
 * Rev. No.:   Version 1.0
 * Rev. Date:  Mar. 5, 2019
 *
 * Purpose:    The module is to be organized as a 4096x8 Byte addressable MEM.
 * When writing to InstructionMemory, it is to be synchronous with the clock.
 * When reading to InstructionMemory, it is to be asynchronous due to that it
 * is not depending on the clock.
 * Either to read/write to memory, the chip select signal (dm_cs) must be
 * asserted with the proper read or write control signals.
 * The output is to have tri-state outputs labeled as z.

 * ©R.W Allison 2019
 ********************************************************************************/
module InstructionMemory(clk, Address, IM_In, im_cs, im_wr, im_rd, IM_Out);

    input         clk, im_cs, im_wr, im_rd;
    input [31:0]  IM_In;
    input [11:0]  Address;
    output [31:0] IM_Out;

    reg [7:0]     IMem [4095:0];

    always @ (posedge clk) begin
        if (im_cs && im_wr)
            {IMem[Address],
             IMem[Address+1],
             IMem[Address+2],
             IMem[Address+3]} <=

            {IM_In[31:24],
             IM_In[23:16],
             IM_In[15:8],
             IM_In[7:0]};

        else
            {IMem[Address],
             IMem[Address+1],
             IMem[Address+2],
             IMem[Address+3]} <=

            {IMem[Address],
             IMem[Address+1],
             IMem[Address+2],
             IMem[Address+3]};
    end

    /*********************************************************************
    * The output is to have tri-state outputs labeled as z.
    * If DataMemory is being read, output will pass 32-bit content at the Address.
    * If not read, output myst in the high impedance state of z.
    **********************************************************************/
    assign IM_Out = {im_cs && im_rd} ?
                                        {IMem[Address],
                                        IMem[Address+1],
                                        IMem[Address+2],
                                        IMem[Address+3]} : 32'hz;


endmodule

```

```
`timescale 1ns / 1ps
/**********************************************************************
*
* Author:   TrieuVy Le
* Filename: LoadReg
* Date:     Feb. 26, 2019
* Version:  1
*
* Notes:    The register allows data in to output only if load is enabled.
* Otherwise data out is held at its previous value. Reset will zeros the array.
*
**********************************************************************/
module InstructionRegister(clk, reset, load, D, Q);

    /*****************************************************************
    * Initilize input, output, and register used within the module.
    * Registers are to use non-blocking assignment.
    * Always block to cycle the loop at positive edge of the clock.
    * Continuous assignment to output the enabled data.
    *****************************************************************/
    input         clk, reset, load;
    input [31:0]  D;

    output reg [31:0] Q;


    always @(posedge clk or posedge reset) begin
        if      (reset)  Q <= 32'h0;
        else if       (load)   Q <= D;
        else               Q <= Q;
    end

    // Continuous assignment

endmodule

```

```
1  `timescale 1ns / 1ps
2  /**********************************************************************
3  *
4  * Author:   TrieuVy Le
5  * Filename: regfile32
6  * Date:     Feb. 26, 2019
7  * Version:  2.0
8  *
9  * Notes:    Reading the register file is asynchronous and
10 *           Writing the register file is to be synchronous
11 *           Both S and T ports are asynch outputs from the RF
12 * Which register to write to is 'synchronously controlled" by the clk in
13 * conjunction with the D_Addr.
14 * A register specified, to be written to only on the rising edge of clk
15 * iff (if and only if) D_En is asserted
16 *
17 **********************************************************************/
18 module regfile32(clk, reset, D, D_Addr, S_Addr, T_Addr, D_En, S, T);
19
20     input           clk, reset, D_En;
21     input    [4:0]  D_Addr, S_Addr, T_Addr;
22     input  [31:0]   D;
23     output [31:0]   S, T;
24
25     reg [31:0]      data [31:0];
26
27     // Read section asynchronusly
28     assign S = data[S_Addr];
29     assign T = data[T_Addr];
30
31     /**********************************************************************
32     * The WRITE section of this module is to be modeled behaviorally and
33     * is sensitive to posedge clk and posedge reset.
34     * The READ section is to be modeled with two contiuous statements.
35     **********************************************************************/
36     always@ (posedge clk, posedge reset) begin
37
38         if (reset)
39             data[0]      <= 32'h0;
40
41         else if ( D_En  && D_Addr != 5'b0)
42             data[D_Addr] <= D;
43
44         else
45             data[D_Addr] <= data[D_Addr];
46     end
47
48 endmodule
49
```

```
1  `timescale 1ns / 1ps
2  /**********************************************************************************
3  *
4  * Author:   TrieuVy Le
5  * Filename: ALU_32
6  * Date:     Feb. 26, 2019
7  * Version:  1
8  *
9  * Notes:    This top module is to instantiate the following:
10 *           1. MIPS_32
11 *           2. DIV_32
12 *           3. MPY_32
13 *           The module includes 2 32-bit S and T inputs, 5-bit opcodes FS,
14 *           2 32-bit Y_LO and Y_HI outputs, and 4 status flags:
15 *           Carry (C), Negative (N), Overflow (V), Zero (Z).
16 *           For MUL and DIV operations, we used the the multiplexer to select
17 *           the correct outputs for {Y_LO, Y_HI, C, N, V, Z} otherwise based on FS,
18 *           the program will choose MIPS settings as the default case.
19 *           There are certain operations that do not require the update of
20 *           the status flags C, N, V, Z due to the types of instructions
21 *           Unsigned #: positive numbers. MSB bit is 2^n - 1.
22 *           Signed #: requires an arithmetic sign. MSB is used to represent
23 *           the sign bit. If MSB is 0 - positive and 1 - negative.
24 *           ADDU, SUBU, STLU do not require signed bit due to unsigned operations.
25 **********************************************************************************/
26 module ALU_32(S, T, FS, shtamt, Y_HI, Y_LO, C, V, N, Z);
27
28     /****************************************
29     * Initilize inputs and outputs to process data
30     * Signals utilize by MUL and DIV
31     * MIPS module interfacing variable
32     ****************************************/
33     input [31:0]  S, T;
34     input [4:0]   FS, shtamt;// shtamt;
35     output [31:0] Y_LO, Y_HI;
36     output        C, N, Z, V;
37
38     wire  [31:0]  Hi_DIV, Lo_DIV, Hi_MUL, Lo_MUL;
39     wire          MUL_N, MUL_Z, DIV_N, DIV_Z;
40
41     wire [31:0]   mipsS, mipsT, mipsY_LO, mipsY_HI;
42     wire          mipsC, mipsN, mipsZ, mipsV;
43     wire [4:0]    mipsFS;
44
45     wire          bs_C, bs_N, bs_Z, bs_V;
46     wire [31:0]   bs_YLO;
47
48     parameter [4:0]      MUL  = 5'h1E,
49                          DIV  = 5'h1F,
50                          SRL = 5'h0C,
51                          SRA = 5'h0D,
52                          SLL = 5'h0E;
53
54
55     MIPS_32  mips32 (   .mipsS(S), .mipsT(T), .mipsFS(FS),
56                         .C(mipsC), .V(mipsV), .N(mipsN), .Z(mipsZ),
57                         .Y_hi(mipsY_HI), .Y_lo(mipsY_LO));
58
59     DIV_32   div32  (       .S_DIV(S), .T_DIV(T), .Quotient(Lo_DIV),
60                             .Remainder(Hi_DIV), .N_DIV(DIV_N), .Z_DIV(DIV_Z));
61
62     MPY_32   mpy32  (.S_MUL(S), .T_MUL(T), .HI_MUL(Hi_MUL),
63                      .LO_MUL(Lo_MUL), .N_MUL(MUL_N), .Z_MUL(MUL_Z));
64
65     BS_32    bs32   (.T(T), .shtamt(shtamt), .S_type(FS),.C(bs_C), .YLO(bs_YLO));
66
67
68
69
```

```
70
71      /*****************************************************************************
72      * The multiplexer is determined based on the FS .
73      * FS = 5'h1E is reserved for MUL op.
74      * FS = 5'h1F is reserved for DIV op.
75      * The respective value of Y_HI, Y_LO, and status flags are interconnected
76      * depending on FS opcodes for MUL & DIV, otherwise default will be MIPS contents.
77      * DIV: Quotient = Y_LO and Remainder = Y_HI
78      * MUL: 64-bit Product = {Y_HI, Y_LO}
79      *****************************************************************************/
80      assign   bs_Z = ~|{Y_LO};
81      assign {Y_HI, Y_LO, N, Z, C, V} =
82                      (FS == MUL) ? {Hi_MUL,   Lo_MUL,   MUL_N, MUL_Z, 1'bx,  1'bx} :
83                      (FS == DIV) ? {Hi_DIV,   Lo_DIV,   DIV_N, DIV_Z, 1'bx,  1'bx} :
84                      (FS == SLL || FS == SRL || FS == SRA) ?
85
86                                      {31'h0,   bs_YLO,    1'bx,  bs_Z,  bs_C,  1'bx} :
87
88                                      {mipsY_HI, mipsY_LO, mipsN, mipsZ, mipsC, mipsV};
89
90  endmodule
91
```

```
1   `timescale 1ns / 1ps
2   /**********************************************************************
3   *
4   * Author:   TrieuVy Le
5   * Filename: MIPS_32
6   * Date:     Feb. 26, 2019
7   * Version:  1
8   *
9   * Notes:    The module includes (all - 2)operations (MUL and DIV)
10  *           The 5-bit FS will select which ALU operation is to be performed.
11  *           N flag is determined by the MSB Y_LO[31].
12  *           Z flag is determined by the content of Y_LO if all 0's .
13  *           Y_HI is always 0's due to data is assigned to Y_LO.
14  *           >>  or <<  : Logical shift and fill with zero.
15  *           >>> or <<< : Arithmetic shift and keep sign  .
16  *           Signed -- Arithmetic -- Integer -- >>> <<< -- MSB
17  *           Unsigned -- Logical -- Regular -- >> << -- No negative
18  *           If a status flag is not affected by the ALU operation, set to X.
19  *           To any unsigned operations including ADDU, SUBU, SLTU, the negative
20  *           flag is set to 0, by definition, it's positive number thus
21  *           do not require an arithmetic sign.
22  **********************************************************************/
23  module MIPS_32(mipsS, mipsT, mipsFS, C, V,N, Z, Y_hi, Y_lo);
24
25      /**********************************************************************
26      * 32 bit inputs S, T and 5 bit OpCode.
27      * Flag C, N, Z, V and lower 32 bit output (do not utilize upper 32 in MIPS).
28      **********************************************************************/
29      input [31:0] mipsS;
30      input [31:0] mipsT;
31      input [4:0]  mipsFS;
32
33      output reg C, V, N, Z;
34      output reg [31:0] Y_lo, Y_hi;
35
36      parameter [4:0] PASS_S  = 5'h00,
37                      PASS_T  = 5'h01,
38                      ADD     = 5'h02,
39                      ADDU    = 5'h03,
40                      SUB     = 5'h04,
41                      SUBU    = 5'h05,
42                      SLT     = 5'h06,
43                      SLTU    = 5'h07,
44                      MUL     = 5'h1E,
45                      DIV     = 5'h1F,
46
47                      AND     = 5'h08,
48                      OR      = 5'h09,
49                      XOR     = 5'h0A,
50                      NOR     = 5'h0B,
51                      SRL     = 5'h0C,
52                      SRA     = 5'h0D,
53                      SLL     = 5'h0E,
54                      ANDI    = 5'h16,
55                      ORI     = 5'h17,
56                      LUI     = 5'h18,
57                      XORI    = 5'h19,
58
59                      INC     = 5'h0F,
60                      INC4    = 5'h10,
61                      DEC     = 5'h11,
62                      DEC4    = 5'h12,
63                      ZEROS   = 5'h13,
64                      ONES    = 5'h14,
65                      SP_INIT = 5'h15;
66
67
68
69
```

```
70
71
72
73          /*****************************************************************************
74          * Casting integers using for MUL, DIV, SLT, or/SRA.
75          * Combinational logic depending on the hex values of FS.
76          *****************************************************************************/
77          integer     INT_S, INT_T;
78
79
80          always @ (*) begin
81              INT_S = mipsS;
82              INT_T = mipsT;
83
84              case (mipsFS)
85                  PASS_S: {C, V, Y_lo}    = {1'bx, 1'bx, mipsS};
86                  PASS_T: {C, V, Y_lo}    = {1'bx, 1'bx, mipsT};
87                  /*********************************************************************
88                  * Signed operations ADD: We compare the MSB of both operands,
89                  * If both MSB are positive and sum is negative => Overflow
90                  * If both MSB are negative and sum is positve  => Overflow
91                  *********************************************************************/
92                  ADD     : begin
93                            {C, Y_lo} = mipsS + mipsT;
94                                case ({mipsS[31], mipsT[31], Y_lo[31]})
95                                      3'b001:     V = 1'b1;
96                                      3'b110:     V = 1'b1;
97                                      default: V = 1'b0;
98                                 endcase
99                            end
100
101                 ADDU    : begin
102                           {C, Y_lo} = mipsS + mipsT;
103                           V = (C == 1'b1) ? 1'b1 : 1'b0;
104                              end
105
106                 /*********************************************************************
107                 * Signed operations SUB: We compare the MSB of both operands & result
108                 * If (+) - (-) and result is negative => Overflow
109                 * If (-) - (+) and result is positive => Overflow
110                 *********************************************************************/
111
112                 SUB     : begin
113                           {C, Y_lo} = mipsS - mipsT;
114                               case ({mipsS[31], mipsT[31], Y_lo[31]})
115                                     3'b011:     V = 1'b1;
116                                     3'b100:     V = 1'b1;
117                                     default: V = 1'b0;
118                                endcase
119                             end
120
121                 SUBU    : begin
122                           {C, Y_lo} = mipsS - mipsT;
123                           V = (C == 1'b1) ? 1'b1 : 1'b0;
124                              end
125
126                 SLT     : {C, V, Y_lo} = {1'bx, 1'bx, ((INT_S < INT_T) ? 32'b1 : 32'b0)};
127                 SLTU    : {C, V, Y_lo} = {1'bx, 1'bx, ((mipsS < mipsT) ? 32'b1 : 32'b0)};
128                 AND     : {C, V, Y_lo} = {1'bx, 1'bx, mipsS & mipsT};
129                 OR      : {C, V, Y_lo} = {1'bx, 1'bx, mipsS | mipsT};
130                 XOR   : {C, V, Y_lo} = {1'bx, 1'bx, mipsS ^ mipsT};
131                 NOR   : {C, V, Y_lo} = {1'bx, 1'bx, ~(mipsS | mipsT)};
132                 SRL     : {C, V, Y_lo} = {mipsT[0], 1'bx, mipsT >> 1};
133                 SRA   : {C, V, Y_lo} = {mipsT[0] , 1'bx, mipsT[31], mipsT[31:1]};
134                 SLL     : {C, V, Y_lo} = {mipsT[31], 1'bx, mipsT << 1};
135                 ANDI    : {C, V, Y_lo} = {1'bx, 1'bx, mipsS & ({16'h0, mipsT[15:0]})};
136                 ORI     : {C, V, Y_lo} = {1'bx, 1'bx, mipsS | ({16'h0, mipsT[15:0]})};
137                 LUI     : {C, V, Y_lo} = {1'bx, 1'bx, ({mipsT[15:0], 16'b0})};
138                 XORI    : {C, V, Y_lo} = {1'bx, 1'bx, mipsS ^({16'b0, mipsT[15:0]})};
```

```
139
140                 INC     : begin
141                             {C, V, Y_lo} = (mipsS + 1);
142                             V = (mipsS[31] == 1'b0 && Y_lo[31] == 1'b1) ? 1'b1 : 1'b0;
143                           end
144
145                 INC4    : begin
146                             {C, V, Y_lo} = (mipsS + 4);
147                             V = (mipsS[31] == 1'b0 && Y_lo[31] == 1'b1) ? 1'b1 : 1'b0;
148                           end
149                 DEC     : begin
150                             {C, V, Y_lo} = (mipsS - 1);
151                             V = (mipsS[31] == 1'b1 && Y_lo[31] == 1'b0) ? 1'b1 : 1'b0;
152                           end
153
154                 DEC4    : begin
155                             {C, V, Y_lo} = (mipsS - 4);
156                             V = (mipsS[31] == 1'b1 && Y_lo[31] == 1'b0) ? 1'b1 : 1'b0;
157                           end
158
159                 ZEROS   : {C, V, Y_lo} = {1'bx, 1'bx, 32'b0};
160                 ONES    : {C, V, Y_lo} = {1'bx, 1'bx, 32'hFFFFFFFF};
161                 SP_INIT : {C, V, Y_lo} = {1'bx, 1'bx, 32'h3FC};
162
163                 default : {C, V, Y_lo} = {1'bx, 1'bx, 32'h0F0F0F0F};
164             endcase
165             /*************************************************************
166             * Negative status flag is determined by Y_LO MSB
167             * Zero status flag is determined if all bits are 0's
168             * Y_HI is default to be = 32'b0 unless specified otherwise.
169             * To differentiate the unsigned operations, we do not take in
170             * consideration of MSB due to that they are positive numbers,
171             * thus do not require the arithmetic sign bit.
172             *************************************************************/
173             N = (mipsFS == ADDU || mipsFS == SUBU || mipsFS == SLTU) ? 1'b0 : Y_lo[31];
174             Z = (Y_lo ==  32'b0) ? 1'b1 : 1'b0;
175             Y_hi = 32'h0;
176         end
177 endmodule
178
```

```
`timescale 1ns / 1ps
/**********************************************************************
*
* Author:   TrieuVy Le
* Filename: DIV_32
* Date:     Feb. 26, 2019
* Version:  1
*
* Notes:    The DIV module yields a 32-bit Quotient (Y_LO)
*           and 32-bit Remainder (Y_HI)
*           The remainder of the operations are to be implemented in ALU_32,
*           it yields a 32-bit result (Y_LO) where ALU_hi will be set to all 0's
*           N status flags is solely dependent on Y_LO/Quotient[31]
*           irrespective to Remainder
*           Z status flags is also dependent on the content of Y_LO/Quotient
*           if all 32'b0 then, Zero flag is set irrespective to Remainder
**********************************************************************/
module DIV_32(S_DIV, T_DIV, Quotient, Remainder, N_DIV, Z_DIV);

    // Initialize inputs and outputs used within the module
    input     [31:0] S_DIV, T_DIV;

    output reg [31:0] Quotient, Remainder;
    output reg        N_DIV, Z_DIV;

    integer           INT_S, INT_T;

    /**********************************************************************
    * Combinational logic, cast integers to be used for MUL and DIV operations
    * due to the signs of MSB so we account for signed bit
    **********************************************************************/
    always @(*) begin
        INT_S = S_DIV;
        INT_T = T_DIV;

        /**********************************************************************
        * Quotient = Y_LO, Remainder = Y_HI
        * Update the Negative status flag based on the MSB bit of the bus
        * Update the Zero status flag if variable content is all 0 -> 1'b0
        **********************************************************************/
        Quotient  = INT_S / INT_T;
        Remainder = INT_S % INT_T;

        N_DIV = Quotient[31];
        Z_DIV = (Quotient == 32'b0) ? 1'b1 : 1'b0;
    end
endmodule


```

```
`timescale 1ns / 1ps
/**********************************************************************
*
* Author:   TrieuVy Le
* Filename: MPY_32
* Date:     Feb. 26, 2019
* Version:  1
*
* Notes:    The MUL module yields a 64-bit product {Y_LO, Y_HI}
*               N status flags is solely dependent on the MSB of the 64-bit product
*               Z status flags is also dependent on the content of the 64-bit product
**********************************************************************/
module MPY_32(S_MUL, T_MUL, HI_MUL, LO_MUL, N_MUL, Z_MUL);

    input      [31:0] S_MUL, T_MUL;
    output reg [31:0] HI_MUL, LO_MUL;
    output reg        N_MUL, Z_MUL;

    // Integers used to cast signed operations
    integer       INT_S, INT_T;

    /*********************************************************************
    * Combinational logic, cast integers to be used for MUL and DIV operations
    * due to the signs of MSB so we account for
    * The operation yields a 64-bit product assigned to the concatenation of
    * Y_HI and Y_LO
    *********************************************************************/
    always @(*) begin

        INT_S = S_MUL;
        INT_T = T_MUL;

        {HI_MUL, LO_MUL} = INT_S * INT_T;

        /*********************************************************************
        * Quotient = Y_LO, Remainder = Y_HI
        * Update the Negative status flag based on the MSB bit of the bus
        * Update the Zero status flag if variable content is all 0 -> 1'b0
        *********************************************************************/
        N_MUL = HI_MUL[31];
        Z_MUL = ({HI_MUL, LO_MUL} == 64'b0) ? 1'b1 : 1'b0;
    end
endmodule

```

```
`timescale 1ns / 1ps
/***************************** C E C S  4 4 0 ****************************
 * File Name:  BS_32
 * Project:    Final Project
 * Designers:  TrieuVy Le and Mark Aquiapao
 * Rev. No.:   Version 1.0
 * Rev. Date:  April 22, 2019

 * Purpose:     To perform 3 types of shifting with a specified amount then
    store result back to Register File.
                1. SRL = 5'h0C, logical shift right
                2. SRA = 5'h0D, arithmetic shift right
                3. SLL = 5'h0E, logical left shift

 * ©R.W Allison 2019
 ********************************************************************************/
module BS_32(T, shtamt, S_type, YLO, C);

    // Initialize input and output
    input [31:0]      T;
    input [4:0]       shtamt, S_type;

    output reg [31:0] YLO;
    output reg C;

    parameter   SRL = 5'h0C,
                SRA = 5'h0D,
                SLL = 5'h0E;

    // We care about the carry flag
    always @ (*) begin
    case (S_type)
      // Logical shift right
      SRL:
        case (shtamt)
              5'd0  : {C, YLO} = {1'b0, T};
              5'd1  : {C, YLO} = {T[0], 1'b0,  T[31:1]  };
              5'd2  : {C, YLO} = {T[1], 2'b0,  T[31:2]  };
              5'd3  : {C, YLO} = {T[2], 3'b0,  T[31:3]  };
              5'd4  : {C, YLO} = {T[3], 4'b0,  T[31:4]  };
              5'd5  : {C, YLO} = {T[4], 5'b0,  T[31:5]  };
              5'd6  : {C, YLO} = {T[5], 6'b0,  T[31:6]  };
              5'd7  : {C, YLO} = {T[6], 7'b0,  T[31:7]  };
              5'd8  : {C, YLO} = {T[7], 8'b0,  T[31:8]  };
              5'd9  : {C, YLO} = {T[8], 9'b0,  T[31:9]  };
              5'd10 : {C, YLO} = {T[9], 10'b0, T[31:10] };
              5'd11 : {C, YLO} = {T[10],11'b0, T[31:11] };
              5'd12 : {C, YLO} = {T[11],12'b0, T[31:12] };
              5'd13 : {C, YLO} = {T[12],13'b0, T[31:13] };
              5'd14 : {C, YLO} = {T[13],14'b0, T[31:14] };
              5'd15 : {C, YLO} = {T[14],15'b0, T[31:15] };
              5'd16 : {C, YLO} = {T[15],16'b0, T[31:16] };
              5'd17 : {C, YLO} = {T[16],17'b0, T[31:17] };
              5'd18 : {C, YLO} = {T[17],18'b0, T[31:18] };
              5'd19 : {C, YLO} = {T[18],19'b0, T[31:19] };
              5'd20 : {C, YLO} = {T[19],20'b0, T[31:20] };
              5'd21 : {C, YLO} = {T[20],21'b0, T[31:21] };
              5'd22 : {C, YLO} = {T[21],22'b0, T[31:22] };
              5'd23 : {C, YLO} = {T[22],23'b0, T[31:23] };
              5'd24 : {C, YLO} = {T[23],24'b0, T[31:24] };
              5'd25 : {C, YLO} = {T[24],25'b0, T[31:25] };
              5'd26 : {C, YLO} = {T[25],26'b0, T[31:26] };
              5'd27 : {C, YLO} = {T[26],27'b0, T[31:27] };
              5'd28 : {C, YLO} = {T[27],28'b0, T[31:28] };
              5'd29 : {C, YLO} = {T[28],29'b0, T[31:29] };
              5'd30 : {C, YLO} = {T[29],30'b0, T[31:30] };
              5'd31 : {C, YLO} = {T[30],31'b0, T[31] };
             default :{C ,YLO} = {33'b0};
            endcase
```

```
70
71
72
73          // Arithmetic shift right
74          SRA:
75                case (shtamt)
76                  5'd0  : {C, YLO} = {1'b0, T};
77                  5'd1  : {C, YLO} = {T[0],{1{T[31]}},T[31:1]};
78                  5'd2  : {C, YLO} = {T[1],{2{T[31]}},T[31:2]};
79                  5'd3  : {C, YLO} = {T[2],{3{T[31]}},T[31:3]};
80                  5'd4  : {C, YLO} = {T[3],{4{T[31]}},T[31:4]};
81                  5'd5  : {C, YLO} = {T[4],{5{T[31]}},T[31:5]};
82                  5'd6  : {C, YLO} = {T[5],{6{T[31]}},T[31:6]};
83                  5'd7  : {C, YLO} = {T[6],{7{T[31]}},T[31:7]};
84                  5'd8  : {C, YLO} = {T[7],{8{T[31]}},T[31:8]};
85                  5'd9  : {C, YLO} = {T[8],{9{T[31]}},T[31:9]};
86                  5'd10 : {C, YLO} = {T[9],{10{T[31]}},T[31:10]};
87                  5'd11 : {C, YLO} = {T[10],{11{T[31]}},T[31:11]};
88                  5'd12 : {C, YLO} = {T[11],{12{T[31]}},T[31:12]};
89                  5'd13 : {C, YLO} = {T[12],{13{T[31]}},T[31:13]};
90                  5'd14 : {C, YLO} = {T[13],{14{T[31]}},T[31:14]};
91                  5'd15 : {C, YLO} = {T[14],{15{T[31]}},T[31:15]};
92                  5'd16 : {C, YLO} = {T[15],{16{T[31]}},T[31:16]};
93                  5'd17 : {C, YLO} = {T[16],{17{T[31]}},T[31:17]};
94                  5'd18 : {C, YLO} = {T[17],{18{T[31]}},T[31:18]};
95                  5'd19 : {C, YLO} = {T[18],{19{T[31]}},T[31:19]};
96                  5'd20 : {C, YLO} = {T[19],{20{T[31]}},T[31:20]};
97                  5'd21 : {C, YLO} = {T[20],{21{T[31]}},T[31:21]};
98                  5'd22 : {C, YLO} = {T[21],{22{T[31]}},T[31:22]};
99                  5'd23 : {C, YLO} = {T[22],{23{T[31]}},T[31:23]};
100                 5'd24 : {C, YLO} = {T[23],{24{T[31]}},T[31:24]};
101                 5'd25 : {C, YLO} = {T[24],{25{T[31]}},T[31:25]};
102                 5'd26 : {C, YLO} = {T[25],{26{T[31]}},T[31:26]};
103                 5'd27 : {C, YLO} = {T[26],{27{T[31]}},T[31:27]};
104                 5'd28 : {C, YLO} = {T[27],{28{T[31]}},T[31:28]};
105                 5'd29 : {C, YLO} = {T[28],{29{T[31]}},T[31:29]};
106                 5'd30 : {C, YLO} = {T[29],{30{T[31]}},T[31:30]};
107                 5'd31 : {C, YLO} = {T[30],{31{T[31]}},T[31:31]};
108                 default : {C,YLO} = {33'b0};
109               endcase
110         // Logical shift left
111         SLL:
112               case(shtamt)
113                5'd0  : {C, YLO} = {1'b0, T};
114                5'd1  : {C, YLO} = {T[31],T[30:0],1'b0};
115                5'd2  : {C, YLO} = {T[30],T[29:0],2'b0};
116                5'd3  : {C, YLO} = {T[29],T[28:0],3'b0};
117                5'd4  : {C, YLO} = {T[28],T[27:0],4'b0};
118                5'd5  : {C, YLO} = {T[27],T[26:0],5'b0};
119                5'd6  : {C, YLO} = {T[26],T[25:0],6'b0};
120                5'd7  : {C, YLO} = {T[25],T[24:0],7'b0};
121                5'd8  : {C, YLO} = {T[24],T[23:0],8'b0};
122                5'd9  : {C, YLO} = {T[23],T[22:0],9'b0};
123                5'd10 : {C, YLO} = {T[22],T[21:0],10'b0};
124                5'd11 : {C, YLO} = {T[21],T[20:0],11'b0};
125                5'd12 : {C, YLO} = {T[20],T[19:0],12'b0};
126                5'd13 : {C, YLO} = {T[19],T[18:0],13'b0};
127                5'd14 : {C, YLO} = {T[18],T[17:0],14'b0};
128                5'd15 : {C, YLO} = {T[17],T[16:0],15'b0};
129                5'd16 : {C, YLO} = {T[16],T[15:0],16'b0};
130                5'd17 : {C, YLO} = {T[15],T[14:0],17'b0};
131                5'd18 : {C, YLO} = {T[14],T[13:0],18'b0};
132                5'd19 : {C, YLO} = {T[13],T[12:0],19'b0};
133                5'd20 : {C, YLO} = {T[12],T[11:0],20'b0};
134                5'd21 : {C, YLO} = {T[11],T[10:0],21'b0};
135                5'd22 : {C, YLO} = {T[10],T[9:0], 22'b0};
136                5'd23 : {C, YLO} = {T[9], T[8:0], 23'b0};
137                5'd24 : {C, YLO} = {T[8], T[7:0], 24'b0};
138                5'd25 : {C, YLO} = {T[7], T[6:0], 25'b0};
```

```
139             5'd26 : {C, YLO} = {T[6], T[5:0], 26'b0};
140             5'd27 : {C, YLO} = {T[5], T[4:0], 27'b0};
141             5'd28 : {C, YLO} = {T[4], T[3:0], 28'b0};
142             5'd29 : {C, YLO} = {T[3], T[2:0], 29'b0};
143             5'd30 : {C, YLO} = {T[2], T[1:0], 30'b0};
144             5'd31 : {C, YLO} = {T[1], T[0:0], 31'b0};
145             default : {C,YLO} = {33'b0};
146           endcase
147         endcase //end shift
148     end
149 endmodule
150
```

```
`timescale 1ns / 1ps
/***************************** C E C S  4 4 0 ****************************
 * File Name:  DataMemory.v
 * Project:    Final Project
 * Designers:  Mark Aquiapao and TrieuVy Le
 * Rev. No.:   Version 1.0
 * Rev. Date:  April 22, 2019
 *
 * Purpose:    The module is to be organized as a 4096x8 Byte addressable MEM.
 * When writing to Data_Memory, it is to be synchronous with the clock.
 * When reading to Data_Memory, it is to be asynchronous due to that it is not
 * depending on the clock.
 * Either to read/write to memory, the chip select signal (dm_cs) must be
 * asserted with the proper read or write control signals.
 * The output is to have tri-state outputs labeled as z.

 * ©R.W Allison 2019
 ********************************************************************************/
module Data_Memory(clk, Address, DM_In, dm_cs, dm_wr, dm_rd, DM_Out);

    input         clk, dm_cs, dm_wr, dm_rd;
    input  [31:0] DM_In;
    input  [11:0] Address;
    output [31:0] DM_Out;

    reg    [7:0]  DataMem [4095:0];

    always @ (posedge clk) begin
        if (dm_cs && dm_wr)
            {DataMem[Address],
             DataMem[Address+1],
             DataMem[Address+2],
             DataMem[Address+3]} <=

            {DM_In[31:24],
             DM_In[23:16],
             DM_In[15:8],
             DM_In[7:0]};

        else
            {DataMem[Address],
             DataMem[Address+1],
             DataMem[Address+2],
             DataMem[Address+3]} <=

            {DataMem[Address],
             DataMem[Address+1],
             DataMem[Address+2],
             DataMem[Address+3]};
    end

    /*********************************************************************
    * The output is to have tri-state outputs labeled as z.
    * If DataMemory is being read, output will pass 32-bit content at the Address.
    * If not read, output myst in the high impedance state of z.
    **********************************************************************/
    assign DM_Out = {dm_cs && dm_rd} ?
                                        {DataMem[Address],
                                        DataMem[Address+1],
                                        DataMem[Address+2],
                                        DataMem[Address+3]} : 32'hz;


endmodule

```

```
1   `timescale 1ns / 1ps
2   /***************************** C E C S  4 4 0 *****************************
3    * File Name:  IOMemory.v
4    * Project:    Final Project
5    * Designers:  Mark Aquiapao and TrieuVy Le
6    * Rev. No.:   Version 1.0
7    * Rev. Date:  April 22, 2019
8    *
9    * Purpose:    IO Memory module using Big Endian format,
10   * used to store data and instructions that could be accessed using Address
11   * as the base location ( Address/ + 1, 2, 3)
12
13   * Â©R.W Allison 2019
14   ********************************************************************************/
15  module IOMemory(    clk,
16                      Address, IO_In,
17                      io_cs, io_wr, io_rd,
18                      intr, inta,
19                      IO_Out  );
20
21     input          clk, io_cs, io_wr, io_rd, inta;
22     input  [31:0]  Address, IO_In;
23
24     output  reg    intr;
25     output [31:0]  IO_Out;
26
27     reg    [7:0]   IOMem [4095:0];
28
29     /********************************************************************************
30     * After a certain amount of time has propagated, we can introduce an interrupt
31     * request.
32     * Note that: although the interrupt request is set, if the SETIE instruction is
33     * not called, then the processor will not get interrupted.
34     * Once acknowledged by observing the rising edge of the interrupt acknowledge flag
35     * we can turn off the request.
36     ********************************************************************************/
37     initial begin
38        intr=0;
39        #1000 intr=1;
40        @(posedge inta) intr=0;
41     end
42
43
44     /********************************************************************************
45     * Similar to data memory
46     * Reading the memory asynchronously in chunks of bytes, even if we want to
47     * read a certain byte, we would have to select the whole word
48     ********************************************************************************/
49
50     assign IO_Out = (io_cs & io_rd)? {IOMem[Address + 0],
51                                       IOMem[Address + 1],
52                                       IOMem[Address + 2],
53                                       IOMem[Address + 3] } : 32'hz;
54
55     /********************************************************************************
56     * Similar to data memory
57     * Writing data synchronously with the clock only with the assertion of cs and wr
58
59     ********************************************************************************/
60     always@(posedge clk)
61        if(io_cs & io_wr) // Write Data Input into the Memory
62                                       {IOMem[Address+0],
63                                        IOMem[Address+1],
64                                        IOMem[Address+2],
65                                        IOMem[Address+3]} <= IO_In;
66        else begin
67                                        IOMem[Address+0] <= IOMem[Address+0];
68                                        IOMem[Address+1] <= IOMem[Address+1];
69                                        IOMem[Address+2] <= IOMem[Address+2];
70                                        IOMem[Address+3] <= IOMem[Address+3];
71        end
72
73  endmodule
74
```

C. Memory Modules

See III./D. report

D. Log Files/Annotations

Instruction Memory Module #1

```
@0
3c 01 12 34  // main:      lui  $01, 0x1234
34 21 56 78  //            ori  $01, 0x5678        # LI   R01,  0x12345678
3c 02 87 65  //            lui  $02, 0x8765
34 42 43 21  //            ori  $02, 0x4321        # LI   R02,  0x87654321
00 01 18 20  //            add  $03, $00, $01      # COPY R03, R01

10 22 00 01  //            beq  $01, $02, no_eq    # should not branch
10 23 00 03  //            beq  $01, $03, yes_eq   # should branch
3c 0e ff ff  // no_eq:     lui  $14, 0xFFFF
35 ce ff ff  //            ori  $14, 0xFFFF        # LI   R14,  0xFFFFFFFF  "fail flag"
00 00 00 0d  //            breaK

00 00 70 20  // yes_eq:    add  $14, $0, $0        # CLR  R14  "pass flag"

14 23 00 01  //            bne  $01, $03, no_ne    # should not branch
14 22 00 03  //            bne  $01, $02, yes_ne   # should branch
3c 0f ff ff  // no_ne:     lui  $15, 0xFFFF
35 ef ff ff  //            ori  $15, 0xFFFF        # LI   R15,  0xFFFFFFFF  "fail flag"
00 00 00 0d  //            break

00 00 78 20  // yes_ne:    add  $15, $0, $0        # CLR  R15  "pass flag"
3c 0d 10 01  //            lui  $13, 0x1001
35 ad 00 c0  //            ori  $13, 0x00C0        # LI   R13,  0x100100C0
ad a1 00 00  //            sw   $01, 0($13)        # ST  [R13], R01
00 00 00 0d  //            break
```

```
Log File #1

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
******CECS 440 FINAL MIPS PROJECT RESULTS**********
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

BREAK INSTRUCTION FETCHED 625.0 ns
R E G I S T E R ' S A F T E R B R E A K

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
Register File Content of $r0 - $r31
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
Time= 631.0 ns $r0  = 00000000 || Time= 631.0 ns $r16 = xxxxx
Time= 631.0 ns $r1  = 12345678 || Time= 631.0 ns $r17 = xxxxx
Time= 631.0 ns $r2  = 87654321 || Time= 631.0 ns $r18 = xxxxxxxx
Time= 631.0 ns $r3  = 12345678 || Time= 631.0 ns $r19 = xxxxxxxx
Time= 631.0 ns $r4  = xxxxxxxx || Time= 631.0 ns $r20 = xxxxxxxx
Time= 631.0 ns $r5  = xxxxxxxx || Time= 631.0 ns $r21 = xxxxxxxx
Time= 631.0 ns $r6  = xxxxxxxx || Time= 631.0 ns $r22 = xxxxxxxx
Time= 631.0 ns $r7  = xxxxxxxx || Time= 631.0 ns $r23 = xxxxxxxx
Time= 631.0 ns $r8  = xxxxxxxx || Time= 631.0 ns $r24 = xxxxxxxx
Time= 631.0 ns $r9  = xxxxxxxx || Time= 631.0 ns $r25 = xxxxxxxx
Time= 631.0 ns $r10 = xxxxxxxx || Time= 631.0 ns $r26 = xxxxxxxx
Time= 631.0 ns $r11 = xxxxxxxx || Time= 631.0 ns $r27 = xxxxxxxx
Time= 631.0 ns $r12 = xxxxxxxx || Time= 631.0 ns $r28 = xxxxxxxx
Time= 631.0 ns $r13 = 100100c0 || Time= 631.0 ns $r29 = 000003fc
Time= 631.0 ns $r14 = 00000000 || Time= 631.0 ns $r30 = xxxxxxxx
Time= 631.0 ns $r15 = 00000000 || Time= 631.0 ns $r31 = xxxxxxxx

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
Memory Location at M[0x3FC]
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
Time= 631.0 ns M[3F0]=xxxxxxxx

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
DATA MEMORY && I/O MEMORY of Memory locations 0xC0h to 0xFFh
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
Data Memory
Time= 631.0 ns DM[000000c0] = 12345678
Time= 631.0 ns DM[000000c4] = xxxxxxxx
Time= 631.0 ns DM[000000c8] = xxxxxxxx
Time= 631.0 ns DM[000000cc] = xxxxxxxx
Time= 631.0 ns DM[000000d0] = xxxxxxxx
Time= 631.0 ns DM[000000d4] = xxxxxxxx
Time= 631.0 ns DM[000000d8] = xxxxxxxx
Time= 631.0 ns DM[000000dc] = xxxxxxxx
Time= 631.0 ns DM[000000e0] = xxxxxxxx
Time= 631.0 ns DM[000000e4] = xxxxxxxx
Time= 631.0 ns DM[000000e8] = xxxxxxxx
Time= 631.0 ns DM[000000ec] = xxxxxxxx
Time= 631.0 ns DM[000000f0] = xxxxxxxx
Time= 631.0 ns DM[000000f4] = xxxxxxxx
Time= 631.0 ns DM[000000f8] = xxxxxxxx
Time= 631.0 ns DM[000000fc] = xxxxxxxx
```

Loaded R1, R2, R3

R13: branch to yes_ne then load immediate value
R14: pass first branch then gets cleared
R15: Pass second branch and gets cleared

SW of R1 to M[R13]

Instruction Memory Module #2

```
@0
3c 01 ff ff  // main: lui  $01, 0xFFFF
34 21 ff ff  //       ori  $01, 0xFFFF        # LI   R01,  0xFFFFFFF
20 02 00 10  //       addi $02, $00, 0x10     # LI   R02,  0x10
3c 0f 10 01  //       lui  $15, 0x1001
35 ef 00 c0  //       ori  $15, 0x00C0        # LI   R15,  0x100100C0

00 01 08 42  // top:  srl  $01, $01, 1        # logical shift right 1 bit
ad e1 00 00  //       sw   $01, 0($15)        # ST  [R15], R01
21 ef 00 04  //       addi $15, $15, 4        # inc memory pointer 4 bytes
20 42 ff ff  //       addi $02, $02, -1       # decrement the loop counter
14 40 ff fb  //       bne  $02, $00, top      #  and jmp to top if not finished
08 10 00 0c  //       j    exit               # jump around a halt instruction
00 00 00 0d  //       break

3c 0e 5a 5a  // exit: lui  $14, 0x5A5A
35 ce 3c 3c  //       ori  $14, 0x3C3C        # LI   R14,  0x5A5A3C3C
00 00 00 0d  //       break
```

Log File #2

```
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
******CECS 440 FINAL MIPS PROJECT RESULTS**********
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

BREAK INSTRUCTION FETCHED 3575.0 ns
R E G I S T E R ' S A F T E R B R E A K

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
Register File Content of $r0 - $r31
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
Time=3581.0 ns $r0 = 00000000 || Time=3581.0 ns $r16 = xx
Time=3581.0 ns $r1 = 0000ffff || Time=3581.0 ns $r17 = xx
Time=3581.0 ns $r2 = 00000000 || Time=3581.0 ns $r18 = xxxxxxxx
Time=3581.0 ns $r3 = xxxxxxxx || Time=3581.0 ns $r19 = xxxxxxxx
Time=3581.0 ns $r4 = xxxxxxxx || Time=3581.0 ns $r20 = xxxxxxxx
Time=3581.0 ns $r5 = xxxxxxxx || Time=3581.0 ns $r21 = xxxxxxxx
Time=3581.0 ns $r6 = xxxxxxxx || Time=3581.0 ns $r22 = xxxxxxxx
Time=3581.0 ns $r7 = xxxxxxxx || Time=3581.0 ns $r23 = xxxxxxxx
Time=3581.0 ns $r8 = xxxxxxxx || Time=3581.0 ns $r24 = xxxxxxxx
Time=3581.0 ns $r9 = xxxxxxxx || Time=3581.0 ns $r25 = xxxxxxxx
Time=3581.0 ns $r10 = xxxxxxxx || Time=3581.0 ns $r26 = xxxxxxxx
Time=3581.0 ns $r11 = xxxxxxxx || Time=3581.0 ns $r27 = xxxxxxxx
Time=3581.0 ns $r12 = xxxxxxxx || Time=3581.0 ns $r28 =
Time=3581.0 ns $r13 = xxxxxxxx || Time=3581.0 ns $r29 =
Time=3581.0 ns $r14 = 5a5a3c3c || Time=3581.0 ns $r30 =
Time=3581.0 ns $r15 = 10010100 || Time=3581.0 ns $r31 =
```

R1: The content was shifted 16 times from original content
R2: The counter for the loop keeping track of shifts

R14: Broke out of the top loop and jumped to the exit to load R14 with the content
R15: The data memory pointer after breaking out of the loop

```
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
Memory Location at M[0x3FC]
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
Time=3581.0 ns M[3F0]=xxxxxxxx

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
DATA MEMORY && I/O MEMORY of Memory locations 0xC0h to 0xFFh
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
Data Memory
Time=3581.0 ns DM[000000c0] = 7fffffff
Time=3581.0 ns DM[000000c4] = 3fffffff
Time=3581.0 ns DM[000000c8] = 1fffffff
Time=3581.0 ns DM[000000cc] = 0fffffff
Time=3581.0 ns DM[000000d0] = 07ffffff
Time=3581.0 ns DM[000000d4] = 03ffffff
Time=3581.0 ns DM[000000d8] = 01ffffff
Time=3581.0 ns DM[000000dc] = 00ffffff
Time=3581.0 ns DM[000000e0] = 007fffff
Time=3581.0 ns DM[000000e4] = 003fffff
Time=3581.0 ns DM[000000e8] = 001fffff
Time=3581.0 ns DM[000000ec] = 000fffff
Time=3581.0 ns DM[000000f0] = 0007ffff
Time=3581.0 ns DM[000000f4] = 0003ffff
Time=3581.0 ns DM[000000f8] = 0001ffff
Time=3581.0 ns DM[000000fc] = 0000ffff
```

Initially R1 <= 0xFFFF_FFFF, then gets **logically shifted right** for a specified amount of time.
Every loop, the value gets shifted and with a zero filled. The result gets stored after the first shift in the memory from

Instruction Memory Module #3

```
@0
3c 01 80 00  // main:      lui  $01, 0x8000
34 21 ff ff  //            ori  $01, 0xFFFF        # LI   R01,  0x8000FFFF
20 02 00 10  //            addi $02, $00, 0x10     # LI   R02,  0x10
3c 0f 10 01  //            lui  $15, 0x1001
35 ef 00 c0  //            ori  $15, 0x00C0        # LI   R15,  0x100100C0

00 01 08 43  // top:       sra  $01, $01, 1        # logical shift right 1 bit
ad e1 00 00  //            sw   $01, 0($15)        # ST  [R15], R01
21 ef 00 04  //            addi $15, $15, 4        # increment the memory pointer 4 bytes
20 42 ff ff  //            addi $02, $02, -1       # decrement the loop counter
14 40 ff fb  //            bne  $02, $00, top      #  and jmp to top if not finished

08 10 00 0c  //            j    exit               # jump around a halt instruction
00 00 00 0d  //            break

3c 0e 5a 5a  // exit:      lui  $14, 0x5A5A
35 ce 3c 3c  //            ori  $14, 0x3C3C        # LI   R14,  0x5A5A3C3C
00 00 00 0d  //            break
```

```
Log File #3

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
******CECS 440 FINAL MIPS PROJECT RESULTS**********
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

BREAK INSTRUCTION FETCHED 3575.0 ns
R E G I S T E R ' S A F T E R B R E A K
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
Register File Content of $r0 - $r31
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
Time=3581.0 ns $r0  = 00000000 || Time=3581.0 ns $r16 =
Time=3581.0 ns $r1  = ffff8000 || Time=3581.0 ns $r17 =
Time=3581.0 ns $r2  = 00000000 || Time=3581.0 ns $r18 = xxxxxxxx
Time=3581.0 ns $r3  = xxxxxxxx || Time=3581.0 ns $r19 = xxxxxxxx
Time=3581.0 ns $r4  = xxxxxxxx || Time=3581.0 ns $r20 = xxxxxxxx
Time=3581.0 ns $r5  = xxxxxxxx || Time=3581.0 ns $r21 = xxxxxxxx
Time=3581.0 ns $r6  = xxxxxxxx || Time=3581.0 ns $r22 = xxxxxxxx
Time=3581.0 ns $r7  = xxxxxxxx || Time=3581.0 ns $r23 = xxxxxxxx
Time=3581.0 ns $r8  = xxxxxxxx || Time=3581.0 ns $r24 = xxxxxxxx
Time=3581.0 ns $r9  = xxxxxxxx || Time=3581.0 ns $r25 = xxxxxxxx
Time=3581.0 ns $r10 = xxxxxxxx || Time=3581.0 ns $r26 = xxxxxxxx
Time=3581.0 ns $r11 = xxxxxxxx || Time=3581.0 ns $r27 = xxxxxxxx
Time=3581.0 ns $r12 = xxxxxxxx || Time=3581.0 ns $r28 =
Time=3581.0 ns $r13 = xxxxxxxx || Time=3581.0 ns $r29 =
Time=3581.0 ns $r14 = 5a5a3c3c || Time=3581.0 ns $r30 =
Time=3581.0 ns $r15 = 10010100 || Time=3581.0 ns $r31 =

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
Memory Location at M[0x3FC]
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
Time=3581.0 ns M[3F0]=xxxxxxxx

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
DATA MEMORY && I/O MEMORY of Memory locations 0xC0h to 0xFFh
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
Data Memory
Time=3581.0 ns DM[000000c0] = c0007fff
Time=3581.0 ns DM[000000c4] = e0003fff
Time=3581.0 ns DM[000000c8] = f0001fff
Time=3581.0 ns DM[000000cc] = f8000fff
Time=3581.0 ns DM[000000d0] = fc0007ff
Time=3581.0 ns DM[000000d4] = fe0003ff
Time=3581.0 ns DM[000000d8] = ff0001ff
Time=3581.0 ns DM[000000dc] = ff8000ff
Time=3581.0 ns DM[000000e0] = ffc0007f
Time=3581.0 ns DM[000000e4] = ffe0003f
Time=3581.0 ns DM[000000e8] = fff0001f
Time=3581.0 ns DM[000000ec] = fff8000f
Time=3581.0 ns DM[000000f0] = fffc0007
Time=3581.0 ns DM[000000f4] = fffe0003
Time=3581.0 ns DM[000000f8] = ffff0001
Time=3581.0 ns DM[000000fc] = ffff8000
```

R1: The content was shifted 16 times from original content
R2: The counter for the loop keeping track of shifts

R14: Broke out of the top loop and jumped to the exit to load R14 with the content
R15: The data memory pointer after breaking out of the loop

Initially R1 <= 0xFFFF_FFFF, then gets **arithmetically shifted right** for a specified amount of time.
Every loop, the result gets stored after the first shift in the memory from 0x8000_FFFF to 0x0000_FFFF

Instruction Memory Module #4

```
@0
3c 01 ff ff  // main:      lui  $01, 0xFFFF
34 21 ff ff  //            ori  $01, 0xFFFF        # LI   R01,  0xFFFFFFF
20 02 00 10  //            addi $02, $00, 0x10     # LI   R02,  0x10
3c 0f 10 01  //            lui  $15, 0x1001
35 ef 00 c0  //            ori  $15, 0x00C0        # LI   R15,  0x100100C0

00 01 08 40  // top:       sll  $01, $01, 1        # logical shift left 1 bit
ad e1 00 00  //            sw   $01, 0($15)        # ST  [R15], R01
21 ef 00 04  //            addi $15, $15, 4        # increment the memory pointer 4 bytes
20 42 ff ff  //            addi $02, $02, -1       # decrement the loop counter
00 02 18 2a  //            slt  $03, $00, $02      # r3 <--1 if r0 < r2
14 60 ff fa  //            bne  $03, $00, top      # jmp if r3==1

08 10 00 0d  //            j    exit               # jump around a halt instruction
00 00 00 0d  //            break

3c 0e 5a 5a  // exit:      lui  $14, 0x5A5A
35 ce 3c 3c  //            ori  $14, 0x3C3C        # LI   R14,  0x5A5A3C3C
00 00 00 0d  //            break
```

```
Log File #4
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
******CECS 440 FINAL MIPS PROJECT RESULTS**********
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

BREAK INSTRUCTION FETCHED 4215.0 ns
R E G I S T E R ' S A F T E R B R E A K

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
Register File Content of $r0 - $r31
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

Time=4221.0 ns $r0  = 00000000 || Time=4221.0 ns $r16 = x
Time=4221.0 ns $r1  = ffff0000 || Time=4221.0 ns $r17 = xxxxxxxx
Time=4221.0 ns $r2  = 00000000 || Time=4221.0 ns $r18 = xxxxxxxx
Time=4221.0 ns $r3  = 00000000 || Time=4221.0 ns $r19 = xxxxxxxx
Time=4221.0 ns $r4  = xxxxxxxx || Time=4221.0 ns $r20 = xxxxxxxx
Time=4221.0 ns $r5  = xxxxxxxx || Time=4221.0 ns $r21 = xxxxxxxx
Time=4221.0 ns $r6  = xxxxxxxx || Time=4221.0 ns $r22 = xxxxxxxx
Time=4221.0 ns $r7  = xxxxxxxx || Time=4221.0 ns $r23 = xxxxxxxx
Time=4221.0 ns $r8  = xxxxxxxx || Time=4221.0 ns $r24 = xxxxxxxx
Time=4221.0 ns $r9  = xxxxxxxx || Time=4221.0 ns $r25 = xxxxxxxx
Time=4221.0 ns $r10 = xxxxxxxx || Time=4221.0 ns $r26 = xxxxxxxx
Time=4221.0 ns $r11 = xxxxxxxx || Time=4221.0 ns $r27 =
Time=4221.0 ns $r12 = xxxxxxxx || Time=4221.0 ns $r28 =
Time=4221.0 ns $r13 = xxxxxxxx || Time=4221.0 ns $r29 =
Time=4221.0 ns $r14 = 5a5a3c3c || Time=4221.0 ns $r30 =
Time=4221.0 ns $r15 = 10010100 || Time=4221.0 ns $r31 = xxxxxxxx

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
Memory Location at M[0x3FC]
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
Time=4221.0 ns M[3F0]=xxxxxxxx

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
DATA MEMORY && I/O MEMORY of Memory locations 0xC0h to 0xFFh
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
Data Memory
Time=4221.0 ns DM[000000c0] = fffffffe
Time=4221.0 ns DM[000000c4] = fffffffc
Time=4221.0 ns DM[000000c8] = fffffff8
Time=4221.0 ns DM[000000cc] = fffffff0
Time=4221.0 ns DM[000000d0] = ffffffe0
Time=4221.0 ns DM[000000d4] = ffffffc0
Time=4221.0 ns DM[000000d8] = ffffff80
Time=4221.0 ns DM[000000dc] = ffffff00
Time=4221.0 ns DM[000000e0] = fffffe00
Time=4221.0 ns DM[000000e4] = fffffc00
Time=4221.0 ns DM[000000e8] = fffff800
Time=4221.0 ns DM[000000ec] = fffff000
Time=4221.0 ns DM[000000f0] = ffffe000
Time=4221.0 ns DM[000000f4] = ffffc000
Time=4221.0 ns DM[000000f8] = ffff8000
Time=4221.0 ns DM[000000fc] = ffff0000
```

R1: The content was shifted 16 times from original content
R2: The counter for the loop keeping track of shifts

R14: Broke out of the top loop and jumped to the exit to load R14 with the content
R15: The data memory pointer after breaking out of the loop

Initially R1 <= 0xFFFF_FFFF, then gets **logically shifted left** for a specified amount of time.
Every loop, the result gets stored after the first shift with the content of 0x8000_FFFF to 0x0000_FFFF in memory location from 0x0000_00C0 to 0x0000_00FF

Instruction Memory Module #5

```
@0
3c 01 ff ff  // main:      lui  $01, 0xFFFF
34 21 ff ff  //            ori  $01, 0xFFFF        # LI   R01,  0xFFFFFFF
20 02 ff f0  //            addi $02, $00, -16      # LI   R02,  -16
3c 0f 10 01  //            lui  $15, 0x1001
35 ef 00 c0  //            ori  $15, 0x00C0        # LI   R15,  0x100100C0

00 01 08 40  // top:       sll  $01, $01, 1        # logical shift left 1 bit
ad e1 00 00  //            sw   $01, 0($15)        # ST  [R15], R01
21 ef 00 04  //            addi $15, $15, 4        # increment the memory pointer 4 bytes
20 42 00 01  //            addi $02, $02, 1        # increment the loop counter
28 43 00 00  //            slti $03, $02, 0        # r3 <--1 if r2 < 0
14 60 ff fa  //            bne  $03, $00, top      # jmp if r3==1

08 10 00 0d  //            j    exit               # jump around a halt instruction
00 00 00 0d  //            break

3c 0e 5a 5a  // exit:      lui  $14, 0x5A5A
35 ce 3c 3c  //            ori  $14, 0x3C3C        # LI   R14,  0x5A5A3C3C
00 00 00 0d  //            break
```

```
Log File #5

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
******CECS 440 FINAL MIPS PROJECT RESULTS**********
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

BREAK INSTRUCTION FETCHED 4215.0 ns
R E G I S T E R ' S  A F T E R  B R E A K
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
Register File Content of $r0 - $r31
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

Time=4221.0 ns $r0 = 00000000 || Time=4221.0 ns $r16 = xxxxxxxx
Time=4221.0 ns $r1 = ffff0000 || Time=4221.0 ns $r17 = xxxxxxxx
Time=4221.0 ns $r2 = 00000000 || Time=4221.0 ns $r18 = xxxxxxxx
Time=4221.0 ns $r3 = 00000000 || Time=4221.0 ns $r19 = xxxxxxxx
Time=4221.0 ns $r4 = xxxxxxxx || Time=4221.0 ns $r20 = xxxxxxxx
Time=4221.0 ns $r5 = xxxxxxxx || Time=4221.0 ns $r21 = xxxxxxxx
Time=4221.0 ns $r6 = xxxxxxxx || Time=4221.0 ns $r22 = xxxxxxxx
Time=4221.0 ns $r7 = xxxxxxxx || Time=4221.0 ns $r23 = xxxxxxxx
Time=4221.0 ns $r8 = xxxxxxxx || Time=4221.0 ns $r24 = xxxxxxxx
Time=4221.0 ns $r9 = xxxxxxxx || Time=4221.0 ns $r25 = x
Time=4221.0 ns $r10 = xxxxxxxx || Time=4221.0 ns $r26 =
Time=4221.0 ns $r11 = xxxxxxxx || Time=4221.0 ns $r27 =
Time=4221.0 ns $r12 = xxxxxxxx || Time=4221.0 ns $r28 =
Time=4221.0 ns $r13 = xxxxxxxx || Time=4221.0 ns $r29 =
Time=4221.0 ns $r14 = 5a5a3c3c || Time=4221.0 ns $r30 =
Time=4221.0 ns $r15 = 10010100 || Time=4221.0 ns $r31 =

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
Memory Location at M[0x3FC]
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
Time=4221.0 ns M[3F0]=xxxxxxxx

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
DATA MEMORY && I/O MEMORY of Memory locations 0xC0h to 0xFFh
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
Data Memory
Time=4221.0 ns DM[000000c0] = fffffffe
Time=4221.0 ns DM[000000c4] = fffffffc
Time=4221.0 ns DM[000000c8] = fffffff8
Time=4221.0 ns DM[000000cc] = fffffff0
Time=4221.0 ns DM[000000d0] = ffffffe0
Time=4221.0 ns DM[000000d4] = ffffffc0
Time=4221.0 ns DM[000000d8] = ffffff80
Time=4221.0 ns DM[000000dc] = ffffff00
Time=4221.0 ns DM[000000e0] = fffffe00
Time=4221.0 ns DM[000000e4] = fffffc00
Time=4221.0 ns DM[000000e8] = fffff800
Time=4221.0 ns DM[000000ec] = fffff000
Time=4221.0 ns DM[000000f0] = ffffe000
Time=4221.0 ns DM[000000f4] = ffffc000
Time=4221.0 ns DM[000000f8] = ffff8000
Time=4221.0 ns DM[000000fc] = ffff0000
```

R1: The content was shifted 16 times from original content
R2: The counter for the loop keeping track of shifts
R3: R3 is set low when slti $03, $02, 0 executed. The comparison R2 < 0 was false.

R14: Broke out of the sll top loop and jumped to the exit to load R14 with the content of 0x5A5A_3C3C
R15: The data memory pointer after breaking out of the loop. Every loop, R15 get incremented by 4 so the variable has a new base address every time sw is executed.

Initially R1 <= 0xFFFF_FFFF, then gets **logically shifted left** for a specified amount of time.
Every loop, the result gets stored after the first shift with the content of 0xFFFF_FFFF to 0xFFFF_0000 in memory location from 0x0000_00C0 to 0x0000_00FF

Instruction Memory Module #6

```
@0
3c 0f 10 01  // lui  $15, 0x1001
35 ef 00 00  // ori  $15, 0x0000        # LI   R15,  0x10010000  dest data pointer
3c 0e 10 01  // lui  $14, 0x1001
35 ce 00 c0  // ori  $14, 0x00C0        # LI   R14,  0x100100C0  dest data pointer
20 0d 00 10  // addi $13, $00, 16       # LI   R13,  16          loop counter
8d e1 00 04  // lw   $01, 04($15)       # Load
8d e2 00 08  // lw   $02, 08($15)       #   R01
8d e3 00 0c  // lw   $03, 12($15)       #     to
8d e4 00 10  // lw   $04, 16($15)       #      R12
8d e5 00 14  // lw   $05, 20($15)
8d e6 00 18  // lw   $06, 24($15)
8d e7 00 1c  // lw   $07, 28($15)
8d e8 00 20  // lw   $08, 32($15)
8d e9 00 24  // lw   $09, 36($15)
8d ea 00 28  // lw   $10, 40($15)
8d eb 00 2c  // lw   $11, 44($15)
8d ec 00 30  // lw   $12, 48($15)

             // mem2mem:
8d f1 00 00  // lw   $17, 00($15)       # do mem to
ad d1 00 00  // sw   $17, 00($14)       #   mem transfer
21 ef 00 04  // addi $15, $15, 04       # bump both source
21 ce 00 04  // addi $14, $14, 04       #   and dest pointers
21 ad ff ff  // addi $13, $13, -1       # dec the loop counter
15 a0 ff fa  // bne  $13, $00, mem2mem  #   and continue till done
00 00 00 0d  // break
```

```
Log File #6

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
******CECS 440 FINAL MIPS PROJECT RESULTS**********
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

BREAK INSTRUCTION FETCHED 4865.0 ns
R E G I S T E R ' S  A F T E R  B R E A K

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
Register File Content of $r0 - $r31
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

Time=4871.0 ns $r0  = 00000000 || Time=4871.0 ns $r16 = xxxxxxxx
Time=4871.0 ns $r1  = 12345678 || Time=4871.0 ns $r17 = 000075cc
Time=4871.0 ns $r2  = 89abcdef || Time=4871.0 ns $r18 = xxxxxxxx
Time=4871.0 ns $r3  = a5a5a5a5 || Time=4871.0 ns $r19 = xxxxxxxx
Time=4871.0 ns $r4  = 5a5a5a5a || Time=4871.0 ns $r20 = xxxxxxxx
Time=4871.0 ns $r5  = 2468ace0 || Time=4871.0 ns $r21 = xxxxxxxx
Time=4871.0 ns $r6  = 13579bdf || Time=4871.0 ns $r22 = xxxxxxxx
Time=4871.0 ns $r7  = 0f0f0f0f || Time=4871.0 ns $r23 = xxxxxxxx
Time=4871.0 ns $r8  = f0f0f0f0 || Time=4871.0 ns $r24 = xxxxxxxx
Time=4871.0 ns $r9  = 00000009 || Time=4871.0 ns $r25 = xxxxxxxx
Time=4871.0 ns $r10 = 0000000a || Time=4871.0 ns $r26 = xxxxxxxx
Time=4871.0 ns $r11 = 0000000b || Time=4871.0 ns $r27 = xxxxxxxx
Time=4871.0 ns $r12 = 0000000c || Time=4871.0 ns $r28 = xxxxxxxx
Time=4871.0 ns $r13 = 00000000 || Time=4871.0 ns $r29 = 000003fc
Time=4871.0 ns $r14 = 10010100 || Time=4871.0 ns $r30 = xxxxxxxx
Time=4871.0 ns $r15 = 10010040 || Time=4871.0 ns $r31 = xxxxxxxx

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
Memory Location at M[0x3FC]
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

Time=4871.0 ns M[3F0]=xxxxxxxx

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
DATA MEMORY && I/O MEMORY of Memory locations 0xC0h to 0xFFh
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

Data Memory
Time=4871.0 ns DM[000000c0] = c3c3c3c3
Time=4871.0 ns DM[000000c4] = 12345678
Time=4871.0 ns DM[000000c8] = 89abcdef
Time=4871.0 ns DM[000000cc] = a5a5a5a5
Time=4871.0 ns DM[000000d0] = 5a5a5a5a
Time=4871.0 ns DM[000000d4] = 2468ace0
Time=4871.0 ns DM[000000d8] = 13579bdf
Time=4871.0 ns DM[000000dc] = 0f0f0f0f
Time=4871.0 ns DM[000000e0] = f0f0f0f0
Time=4871.0 ns DM[000000e4] = 00000009
Time=4871.0 ns DM[000000e8] = 0000000a
Time=4871.0 ns DM[000000ec] = 0000000b
Time=4871.0 ns DM[000000f0] = 0000000c
Time=4871.0 ns DM[000000f4] = 0000000d
Time=4871.0 ns DM[000000f8] = fffffff8
Time=4871.0 ns DM[000000fc] = 000075cc
```

R1-R12 get loaded with the LW instruction pointing at different base addresses
R13: The counter for the loop keeping track of shifts
R14: To perform a sw as the base address
R15: To perform a lw as the base address

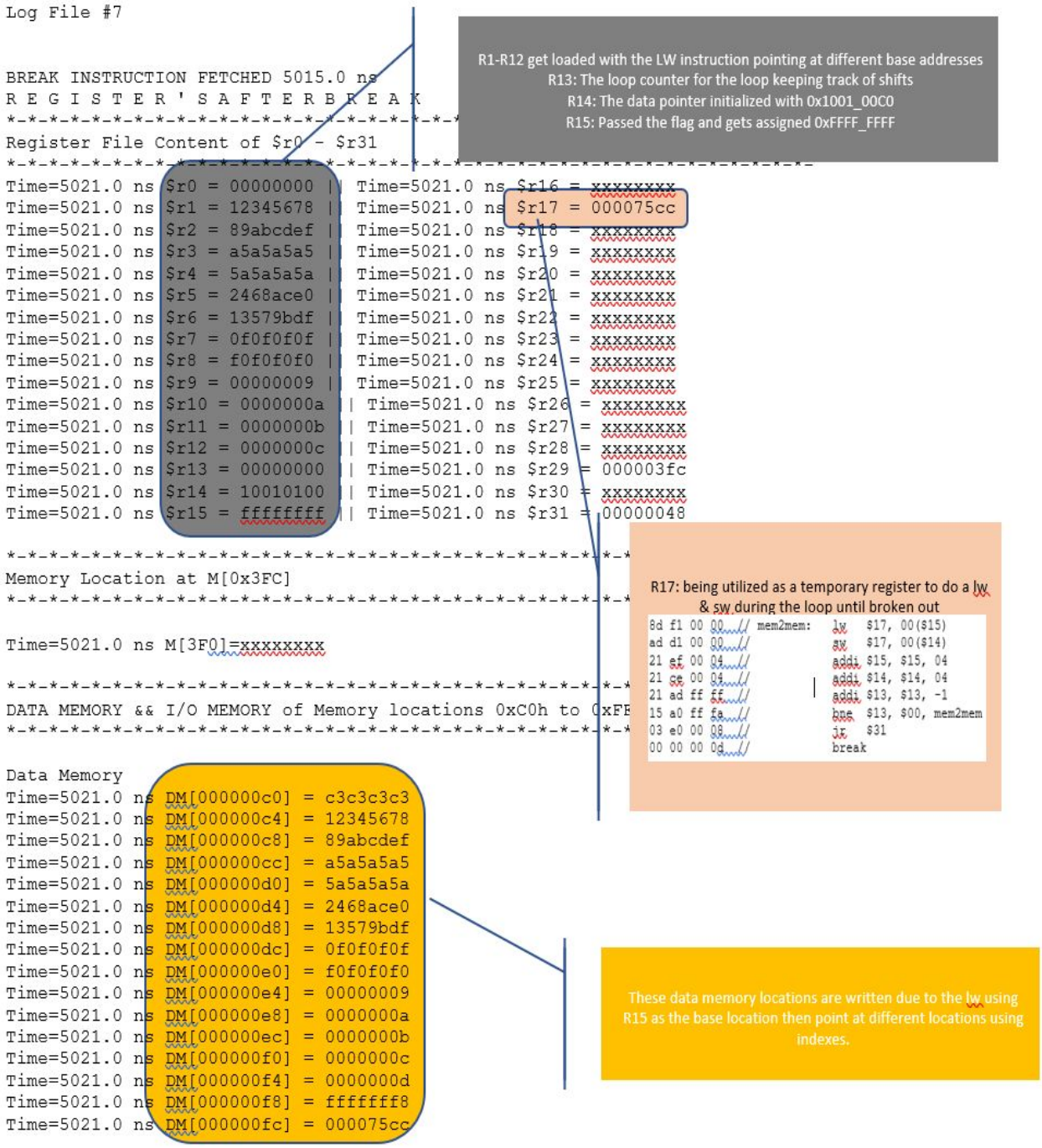
R17: being utilized as a temporary register to do a lw & sw during the loop until broken out

```
                // mem2mem:
8d f1 00 00   // lw   $17, 00($15)
ad d1 00 00   // sw   $17, 00($14)
21 ef 00 04   // addi $15, $15, 04
21 ce 00 04   // addi $14, $14, 04
21 ad ff ff   // addi $13, $13, -1
15 a0 ff fa   // bne  $13, $00, mem2mem
00 00 00 0d   // break
```

These data memory locations are written due to the lw using R15 as the base location then point at different locations using indexes.

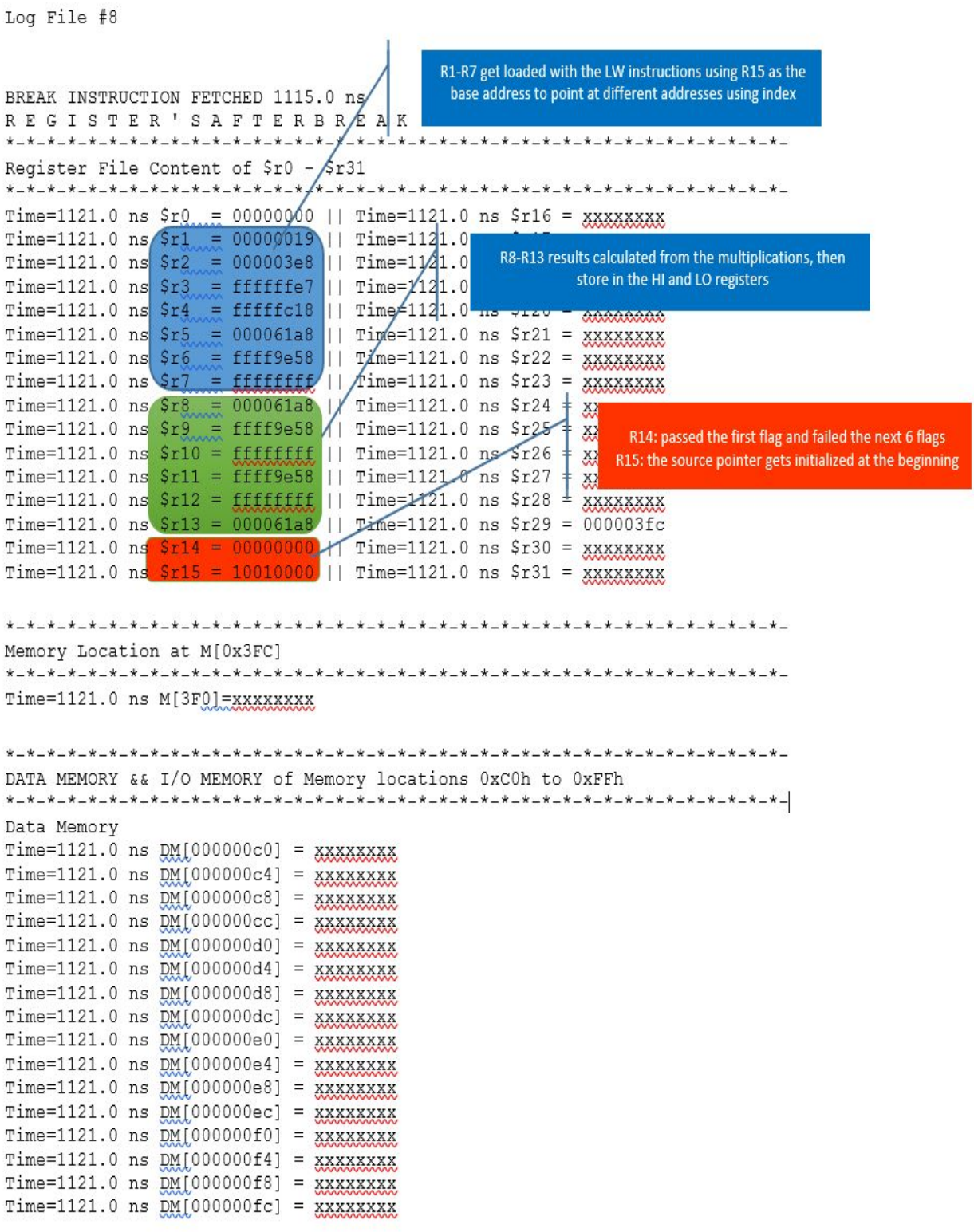
Instruction Memory Module #7

```
@0
3c 0f 10 01  // main:      lui  $15, 0x1001
35 ef 00 00  //             ori  $15, 0x0000        # LI   R15,  0x10010000  dest data pointer
3c 0e 10 01  //             lui  $14, 0x1001
35 ce 00 c0  //             ori  $14, 0x00C0        # LI   R14,  0x100100C0  dest data pointer
20 0d 00 10  //             addi $13, $00, 16       # LI   R13,  16          loop counter
8d e1 00 04  //             lw   $01, 04($15)       # Load
8d e2 00 08  //             lw   $02, 08($15)       #   R01
8d e3 00 0c  //             lw   $03, 12($15)       #     to
8d e4 00 10  //             lw   $04, 16($15)       #      R12
8d e5 00 14  //             lw   $05, 20($15)
8d e6 00 18  //             lw   $06, 24($15)
8d e7 00 1c  //             lw   $07, 28($15)
8d e8 00 20  //             lw   $08, 32($15)
8d e9 00 24  //             lw   $09, 36($15)
8d ea 00 28  //             lw   $10, 40($15)
8d eb 00 2c  //             lw   $11, 44($15)
8d ec 00 30  //             lw   $12, 48($15)

0c 10 00 15  //             jal  mem2mem
3c 0f ff ff  //             lui  $15, 0xFFFF
35 ef ff ff  //             ori  $15, 0xFFFF        # LI   R15,  0xFFFFFFFF  "pass flag"
00 00 00 0d  //             break

8d f1 00 00  // mem2mem:    lw   $17, 00($15)       # do mem to
ad d1 00 00  //             sw   $17, 00($14)       #   mem transfer
21 ef 00 04  //             addi $15, $15, 04       # bump both source
21 ce 00 04  //             addi $14, $14, 04       #   and dest pointers
21 ad ff ff  //             addi $13, $13, -1       # dec the loop counter
15 a0 ff fa  //             bne  $13, $00, mem2mem  #   and continue till done
03 e0 00 08  //             jr   $31                # return to calling code
00 00 00 0d  //             break                   # safety net
```


Log File #7
BREAK INSTRUCTION FETCHED 5015.0 ns
R E G I S T E R ' S A F T E R B R E A K
Register File Content of $r0 - $r31
Time=5021.0 ns $r0 = 00000000 || Time=5021.0 ns $r16 = xxxxxxxx
Time=5021.0 ns $r1 = 12345678 || Time=5021.0 ns $r17 = 000075cc
Time=5021.0 ns $r2 = 89abcdef || Time=5021.0 ns $r18 = xxxxxxxx
Time=5021.0 ns $r3 = a5a5a5a5 || Time=5021.0 ns $r19 = xxxxxxxx
Time=5021.0 ns $r4 = 5a5a5a5a || Time=5021.0 ns $r20 = xxxxxxxx
Time=5021.0 ns $r5 = 2468ace0 || Time=5021.0 ns $r21 = xxxxxxxx
Time=5021.0 ns $r6 = 13579bdf || Time=5021.0 ns $r22 = xxxxxxxx
Time=5021.0 ns $r7 = 0f0f0f0f || Time=5021.0 ns $r23 = xxxxxxxx
Time=5021.0 ns $r8 = f0f0f0f0 || Time=5021.0 ns $r24 = xxxxxxxx
Time=5021.0 ns $r9 = 00000009 || Time=5021.0 ns $r25 = xxxxxxxx
Time=5021.0 ns $r10 = 0000000a || Time=5021.0 ns $r26 = xxxxxxxx
Time=5021.0 ns $r11 = 0000000b || Time=5021.0 ns $r27 = xxxxxxxx
Time=5021.0 ns $r12 = 0000000c || Time=5021.0 ns $r28 = xxxxxxxx
Time=5021.0 ns $r13 = 00000000 || Time=5021.0 ns $r29 = 000003fc
Time=5021.0 ns $r14 = 10010100 || Time=5021.0 ns $r30 = xxxxxxxx
Time=5021.0 ns $r15 = ffffffff || Time=5021.0 ns $r31 = 00000048
Memory Location at M[0x3FC]
Time=5021.0 ns M[3F0]=xxxxxxxx
DATA MEMORY && I/O MEMORY of Memory locations 0xC0h to 0xFF
Data Memory
Time=5021.0 ns DM[000000c0] = c3c3c3c3
Time=5021.0 ns DM[000000c4] = 12345678
Time=5021.0 ns DM[000000c8] = 89abcdef
Time=5021.0 ns DM[000000cc] = a5a5a5a5
Time=5021.0 ns DM[000000d0] = 5a5a5a5a
Time=5021.0 ns DM[000000d4] = 2468ace0
Time=5021.0 ns DM[000000d8] = 13579bdf
Time=5021.0 ns DM[000000dc] = 0f0f0f0f
Time=5021.0 ns DM[000000e0] = f0f0f0f0
Time=5021.0 ns DM[000000e4] = 00000009
Time=5021.0 ns DM[000000e8] = 0000000a
Time=5021.0 ns DM[000000ec] = 0000000b
Time=5021.0 ns DM[000000f0] = 0000000c
Time=5021.0 ns DM[000000f4] = 0000000d
Time=5021.0 ns DM[000000f8] = fffffff8
Time=5021.0 ns DM[000000fc] = 000075cc
R1-R12 get loaded with the LW instruction pointing at different base addresses
R13: The loop counter for the loop keeping track of shifts
R14: The data pointer initialized with 0x1001_00C0
R15: Passed the flag and gets assigned 0xFFFF_FFFF
R17: being utilized as a temporary register to do a lw & sw during the loop until broken out
8d f1 00 00 // mem2mem: lw $17, 00($15)
ad d1 00 00 // sw $17, 00($14)
21 ef 00 04 // addi $15, $15, 04
21 ce 00 04 // addi $14, $14, 04
21 ad ff ff // addi $13, $13, -1
15 a0 ff fa // bne $13, $00, mem2mem
03 e0 00 08 // jr $31
00 00 00 0d // break
These data memory locations are written due to the lw using R15 as the base location then point at different locations using indexes.

Instruction Memory Module #8

```
@0
3c 0f 10 01  // main:      lui  $15, 0x1001
35 ef 00 00  //            ori  $15, 0x0000        # $r15 <-- 0x10010000 (src pointer)
8d e1 00 00  //            lw   $01, 00($15)       # $r01 <--     25
8d e2 00 04  //            lw   $02, 04($15)       # $r02 <--   1000
8d e3 00 08  //            lw   $03, 08($15)       # $r03 <--    -25
8d e4 00 0c  //            lw   $04, 12($15)       # $r04 <--  -1000
8d e5 00 10  //            lw   $05, 16($15)       # $r05 <--  25000
8d e6 00 14  //            lw   $06, 20($15)       # $r06 <-- -25000
8d e7 00 18  //            lw   $07, 24($15)       # $r07 <--     -1
00 22 00 18  //            mult $01, $02
00 00 40 12  //            mflo $08                # rs=pos rt=pos rd=pos
14 a8 00 10  //            bne  $05, $08, fail1
00 62 00 18  //            mult $03, $02
00 00 48 12  //            mflo $09                # rs=neg rt=pos rd=neg
00 00 50 10  //            mfhi $10
14 c9 00 0f  //            bne  $06, $09, fail2L
14 ea 00 11  //            bne  $07, $10, fail2H
00 24 00 18  //            mult $01, $04
00 00 58 12  //            mflo $11                # rs=pos rt=neg rd=neg
00 00 60 10  //            mfhi $12
14 cb 00 10  //            bne  $06, $11, fail3L
14 ec 00 12  //            bne  $07, $12, fail3H
00 64 00 18  //            mult $03, $04
00 00 68 12  //            mflo $13                # rs=neg rt=neg rd=pos
14 ad 00 12  //            bne  $05, $13, fail4

3c 0e 00 00  // pass:      lui  $14, 0x0000
35 ce 00 00  //            ori  $14, 0x0000        # $r14 <-- 0x00000000  (Pass flag)
00 00 00 0d  //            break
3c 0e ff ff  // fail1:     lui  $14, 0xFFFF
35 ce ff ff  //            ori  $14, 0xFFFF        # $r14 <-- 0xFFFFFFFF  (Fail flag 1)
00 00 00 0d  //            break
3c 0e ff ff  // fail2L:    lui  $14, 0xFFFF
35 ce ff fe  //            ori  $14, 0xFFFE        # $r14 <-- 0xFFFFFFFE  (Fail flag 2L)
00 00 00 0d  //            break
3c 0e ff ff  // fail2H:    lui  $14, 0xFFFF
35 ce ff fd  //            ori  $14, 0xFFFD        # $r14 <-- 0xFFFFFFFD  (Fail flag 2H)
00 00 00 0d  //            break
3c 0e ff ff  // fail3L:    lui  $14, 0xFFFF
35 ce ff fc  //            ori  $14, 0xFFFC        # $r14 <-- 0xFFFFFFFC  (Fail flag 3L)
00 00 00 0d  //            break
3c 0e ff ff  // fail3H:    lui  $14, 0xFFFF
35 ce ff fb  //            ori  $14, 0xFFFB        # $r14 <-- 0xFFFFFFFB  (Fail flag 3H)
00 00 00 0d  //            break
3c 0e ff ff  // fail4:     lui  $14, 0xFFFF
35 ce ff fa  //            ori  $14, 0xFFFA        # $r14 <-- 0xFFFFFFFA  (Fail flag 4)
00 00 00 0d  //            break
```


Log File #8
R1-R7 get loaded with the LW instructions using R15 as the base address to point at different addresses using index
BREAK INSTRUCTION FETCHED 1115.0 ns
R E G I S T E R ' S A F T E R B R E A K
Register File Content of $r0 - $r31
Time=1121.0 ns $r0 = 00000000 || Time=1121.0 ns $r16 = xxxxxxxx
Time=1121.0 ns $r1 = 00000019 || Time=1121.0
Time=1121.0 ns $r2 = 000003e8 || Time=1121.0
Time=1121.0 ns $r3 = fffffe7 || Time=1121.0
Time=1121.0 ns $r4 = fffffc18 || Time=1121.0
Time=1121.0 ns $r5 = 000061a8 || Time=1121.0 ns $r21 = xxxxxxxx
Time=1121.0 ns $r6 = ffff9e58 || Time=1121.0 ns $r22 = xxxxxxxx
Time=1121.0 ns $r7 = ffffffff || Time=1121.0 ns $r23 = xxxxxxxx
Time=1121.0 ns $r8 = 000061a8 || Time=1121.0 ns $r24 = xx
Time=1121.0 ns $r9 = ffff9e58 || Time=1121.0 ns $r25 = xx
Time=1121.0 ns $r10 = ffffffff || Time=1121.0 ns $r26 = xx
Time=1121.0 ns $r11 = ffff9e58 || Time=1121.0 ns $r27 = xx
Time=1121.0 ns $r12 = ffffffff || Time=1121.0 ns $r28 = xxxxxxxx
Time=1121.0 ns $r13 = 000061a8 || Time=1121.0 ns $r29 = 000003fc
Time=1121.0 ns $r14 = 00000000 || Time=1121.0 ns $r30 = xxxxxxxx
Time=1121.0 ns $r15 = 10010000 || Time=1121.0 ns $r31 = xxxxxxxx
R8-R13 results calculated from the multiplications, then store in the HI and LO registers
R14: passed the first flag and failed the next 6 flags
R15: the source pointer gets initialized at the beginning
Memory Location at M[0x3FC]
Time=1121.0 ns M[3F0]=xxxxxxxx
DATA MEMORY && I/O MEMORY of Memory locations 0xC0h to 0xFFh
Data Memory
Time=1121.0 ns DM[000000c0] = xxxxxxxx
Time=1121.0 ns DM[000000c4] = xxxxxxxx
Time=1121.0 ns DM[000000c8] = xxxxxxxx
Time=1121.0 ns DM[000000cc] = xxxxxxxx
Time=1121.0 ns DM[000000d0] = xxxxxxxx
Time=1121.0 ns DM[000000d4] = xxxxxxxx
Time=1121.0 ns DM[000000d8] = xxxxxxxx
Time=1121.0 ns DM[000000dc] = xxxxxxxx
Time=1121.0 ns DM[000000e0] = xxxxxxxx
Time=1121.0 ns DM[000000e4] = xxxxxxxx
Time=1121.0 ns DM[000000e8] = xxxxxxxx
Time=1121.0 ns DM[000000ec] = xxxxxxxx
Time=1121.0 ns DM[000000f0] = xxxxxxxx
Time=1121.0 ns DM[000000f4] = xxxxxxxx
Time=1121.0 ns DM[000000f8] = xxxxxxxx
Time=1121.0 ns DM[000000fc] = xxxxxxxx

Instruction Memory Module #9

```
@0
3c 0f 10 01  // main:      lui  $15, 0x1001
35 ef 00 c0  //            ori  $15, 0x00C0        # $r15 <-- 0x100100C0  (dest pointer)
20 01 ff 8a  //            addi $01, $00, -118     # $r01 <-- 0xFFFFFF8A
20 02 00 8a  //            addi $02  $00,  138     # $r02 <-- 0x0000008A
0c 10 00 22  //            jal  slt_tests

3c 0d 77 88  //            lui  $13, 0x7788
35 ad 77 88  //            ori  $13, 0x7788        # $r13 <-- 0x77887788  (pattern1)
3c 0c 88 77  //            lui  $12, 0x8877
35 8c 88 77  //            ori  $12, 0x8877        # $r12 <-- 0x88778877  (pattern2)
3c 0b ff ff  //            lui  $11, 0xFFFF
35 6b ff ff  //            ori  $11, 0xFFFF        # $r11 <-- 0xFFFFFFFF  (pattern3)

01 ac 50 26  //            xor  $10, $13, $12      # $r10 <-- 0xFFFFFFFF
11 4b 00 02  //            beq  $10, $11, xor_pass
20 0e ff fb  //            addi $14, $00, -5       # fail flag5 r14 <-- 0xFFFF_FFFB
00 00 00 0d  //            break
01 ac 48 24  // xor_pass:  and  $09, $13, $12      # $r09 <-- 0x00000000
11 20 00 02  //            beq  $09, $00, and_pass
20 0e ff fa  //            addi $14, $00, -6       # fail flag6 r14 <-- 0xFFFF_FFFA
00 00 00 0d  //            break
01 e2 48 25  // and_pass:  or   $09, $15, $02      # $r09 <-- 0x100100CA
3c 08 10 01  //            lui  $08, 0x1001
35 08 00 ca  //            ori  $08, 0x00CA        # $r08 <-- 0x100100CA
11 09 00 02  //            beq  $08, $09, or_pass
20 0e ff f9  //            addi $14, $00, -7       # fail flag7 r14 <-- 0xFFFF_FFF9
00 00 00 0d  //            break
01 e2 48 27  // or_pass:   nor  $09, $15, $02      # $r09 <-- 0xEFFEFF35
3c 08 ef fe  //            lui  $08, 0xEFFE
35 08 ff 35  //            ori  $08, 0xFF35        # $r08 <-- 0xEFFEFF35
11 09 00 02  //            beq  $08, $09, nor_pass
20 0e ff f8  //            addi $14, $00, -8       # fail flag8 r14 <-- 0xFFFF_FFF8
00 00 00 0d  //            break
ad e8 00 10  // nor_pass:  sw   $08, 0x10($15)     # M[D0] <-- 0xEFFEFF35
00 00 70 20  //            add  $14, $00, $00      # clear r14 indicating "passed all"
00 00 00 0d  //            break                   # should stop here, having
             //                                    #   completed all the tests

00 22 18 2a  // slt_tests: slt  $03, $01, $02      # for signed# r01 < r02
14 60 00 02  //            bne  $03, $00, slt1     #   thus, we should branch
20 0e ff ff  //            addi $14, $00, -1       # fail flag1 r14 <-- FFFF_FFFF
00 00 00 0d  //            break
20 04 00 c0  // slt1:      addi $04, $00, 0xC0   # pass flag1 M[C0] <-- C0
ad e4 00 00  //            sw   $04, 0x00($15)

00 41 18 2b  //            sltu $03, $02, $01      # for unsigned# r02 < r01
14 60 00 02  //            bne  $03, $00, slt2     #   thus, we should branch
20 0e ff fe  //            addi $14, $00, -2       # fail flag2 r14 <-- FFFF_FFFE
00 00 00 0d  //            break
20 05 00 c4  // slt2:      addi $05, $00, 0xC4   # pass flag1 M[C4] <-- C4
ad e5 00 04  //            sw   $05, 0x04($15)

00 41 18 2a  //            slt  $03, $02, $01      # for signed# r02 !< r01
10 60 00 02  //            beq  $03, $00, slt3     #   thus, we should branch
20 0e ff fd  //            addi $14, $00, -3       # fail flag3 r14 <-- FFFF_FFFD
00 00 00 0d  //            break
20 06 00 c8  // slt3:      addi $06, $00, 0xC8   # pass flag3 M[C8] <-- C8
ad e6 00 08  //            sw   $06, 0x08($15)

00 22 18 2b  //            sltu $03, $01, $02      # for unsigned# r01 !< r02
10 60 00 02  //            beq  $03, $00, slt4     #   thus, we should branch
20 0e ff fc  //            addi $14, $00, -4       # fail flag4 r14 <-- FFFF_FFFC
00 00 00 0d  //            break
20 07 00 cc  // slt4:      addi $07, $00, 0xCC     # pass flag4 M[CC] <-- CC
ad e7 00 0c  //            sw   $07, 0x0C($15)
03 e0 00 08  //            jr   $31                # return from subroutine
```

Log File #9

```
BREAK INSTRUCTION FETCHED 1735.0 ns
R E G I S T E R ' S A F T E R B R E A K
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
Register File Content of $r0 - $r31
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
Time=1741.0 ns $r0  = 00000000 || Time=1741.0 ns $r16 = xxxxxxxx
Time=1741.0 ns $r1  = ffffff8a || Time=1741.0 ns $r17 = xxxxxxxx
Time=1741.0 ns $r2  = 0000008a || Time=1741.0 ns $r18 = xxx
Time=1741.0 ns $r3  = 00000000 || Time=1741.0 ns $r19 = xxx
Time=1741.0 ns $r4  = 000000c0 || Time=1741.0 ns $r20 = xxx
Time=1741.0 ns $r5  = 000000c4 || Time=1741.0 ns $r21 = xxxxxxxx
Time=1741.0 ns $r6  = 000000c8 || Time=1741.0 ns $r22 = xxxxxxxx
Time=1741.0 ns $r7  = 000000cc || Time=1741.0 ns $r23 = xxxxxxxx
Time=1741.0 ns $r8  = effeff35 || Time=1741.0 ns $r24 = xxxxxxxx
Time=1741.0 ns $r9  = effeff35 || Time=1741.0 ns $r25 = xxxxx
Time=1741.0 ns $r10 = ffffffff || Time=1741.0 ns $r26 = xxxxx
Time=1741.0 ns $r11 = ffffffff || Time=1741.0 ns $r27 = xxxxx
Time=1741.0 ns $r12 = 88778877 || Time=1741.0 ns $r28 = xxxxx
Time=1741.0 ns $r13 = 77887788 || Time=1741.0 ns $r29 = 00000
Time=1741.0 ns $r14 = 00000000 || Time=1741.0 ns $r30 = xxxxxxxx
Time=1741.0 ns $r15 = 100100c0 || Time=1741.0 ns $r31 = 00000014

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
Memory Location at M[0x3FC]
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-

Time=1741.0 ns M[3F0]=xxxxxxxx

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
DATA MEMORY && I/O MEMORY of Memory locations 0xC0h to 0xFFh
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-

Data Memory
Time=1741.0 ns DM[000000c0] = 000000c0
Time=1741.0 ns DM[000000c4] = 000000c4
Time=1741.0 ns DM[000000c8] = 000000c8
Time=1741.0 ns DM[000000cc] = 000000cc
Time=1741.0 ns DM[000000d0] = effeff35
Time=1741.0 ns DM[000000d4] = xxxxxxxx
Time=1741.0 ns DM[000000d8] = xxxxxxxx
Time=1741.0 ns DM[000000dc] = xxxxxxxx
Time=1741.0 ns DM[000000e0] = xxxxxxxx
Time=1741.0 ns DM[000000e4] = xxxxxxxx
Time=1741.0 ns DM[000000e8] = xxxxxxxx
Time=1741.0 ns DM[000000ec] = xxxxxxxx
Time=1741.0 ns DM[000000f0] = xxxxxxxx
Time=1741.0 ns DM[000000f4] = xxxxxxxx
Time=1741.0 ns DM[000000f8] = xxxxxxxx
Time=1741.0 ns DM[000000fc] = xxxxxxxx
```

R1 and R2 were initialized at the beginning and used to compare for loops
R3: register used set either 0/1 for a comparison.

These registers get written by passing through the loops with either branches or jumps

R14: content stays at 0x0000_0000 if passed all tests mostly branches
R15: The destination pointer acts as the base address to store data to memory

Data get written to memory due to all slt tests because they are all linked up

Instruction Memory Module #10

```
@0
3c 0f 10 01  // main:      lui  $15, 0x1001
35 ef 00 00  //            ori  $15, 0x0000       # $r15 <-- 0x10010000  (source pointer)
8d e1 00 00  //            lw   $01, 00($15)      # $r01 <--  264465
8d e2 00 04  //            lw   $02, 04($15)      # $r02 <--    1000
8d e3 00 08  //            lw   $03, 08($15)      # $r03 <-- -264465
8d e4 00 0c  //            lw   $04, 12($15)      # $r04 <--   -1000
8d e5 00 10  //            lw   $05, 16($15)      # $r05 <--     264   Quot1,4    w01 div w02, w03 div w04
8d e6 00 14  //            lw   $06, 20($15)      # $r06 <--     465   Rem 1,3    w01 rem w02, w01 rem w04
8d e7 00 18  //            lw   $07, 24($15)      # $r07 <--    -264   Quot2,3    w03 div w02, w01 div w04
8d e8 00 1c  //            lw   $08, 28($15)      # $r08 <--    -465   Rem 2,4    w03 re0 w02, w03 rem w04

00 22 00 1a  //            div  $01, $02
00 00 48 12  //            mflo $09               # rs=pos / rt=pos, rem=pos quot=pos
00 00 50 10  //            mfhi $10
15 25 00 16  //            bne  $09, $05, fail1Q
15 46 00 18  //            bne  $10, $06, fail1R

00 62 00 1a  //            div  $03, $02
00 00 48 12  //            mflo $09               # rs=neg / rt=pos, rem=neg quot=neg
00 00 50 10  //            mfhi $10
15 27 00 17  //            bne  $09, $07, fail2Q
15 48 00 19  //            bne  $10, $08, fail2R

00 24 00 1a  //            div  $01, $04
00 00 48 12  //            mflo $09               # rs=pos / rt=neg, rem=pos quot=neg
00 00 50 10  //            mfhi $10
15 27 00 18  //            bne  $09, $07, fail3Q
15 46 00 1a  //            bne  $10, $06, fail3R

00 64 00 1a  //            div  $03, $04
00 00 48 12  //            mflo $09               # rs=neg / rt=neg, rem=neg quot=pos
00 00 50 10  //            mfhi $10
15 25 00 19  //            bne  $09, $05, fail4Q
15 48 00 1b  //            bne  $10, $08, fail4R

3c 0b 00 00  // pass:      lui  $11, 0x0000
35 6b 00 00  //            ori  $11, 0x0000       # $r11 <-- 0x00000000  (Pass flag)
00 0b 60 20  //            add  $12, $00, $11     # $r12 <-- Pass
00 0b 68 20  //            add  $13, $00, $11     # $r13 <-- Pass
00 0b 70 20  //            add  $14, $00, $11     # $r14 <-- Pass
00 00 00 0d  //            break

3c 0e ff ff  // fail1Q:    lui  $14, 0xFFFF
35 ce ff ff  //            ori  $14, 0xFFFF       # $r14 <-- 0xFFFFFFFF  (Fail flag 1 Quot)
00 00 00 0d  //            break
3c 0e ff ff  // fail1R:    lui  $14, 0xFFFF
35 ce ff fe  //            ori  $14, 0xFFFE       # $r14 <-- 0xFFFFFFFE  (Fail flag 1 Rem)
00 00 00 0d  //            break
3c 0e ff ff  // fail2Q:    lui  $14, 0xFFFF
35 ce ff fd  //            ori  $14, 0xFFFD       # $r14 <-- 0xFFFFFFFD  (Fail flag 2 Quot)
00 00 00 0d  //            break
3c 0e ff ff  // fail2R:    lui  $14, 0xFFFF
35 ce ff fc  //            ori  $14, 0xFFFC       # $r14 <-- 0xFFFFFFFC  (Fail flag 2 Rem)
00 00 00 0d  //            break
3c 0e ff ff  // fail3Q:    lui  $14, 0xFFFF
35 ce ff fb  //            ori  $14, 0xFFFB       # $r14 <-- 0xFFFFFFFB  (Fail flag 3 Quot)
00 00 00 0d  //            break
3c 0e ff ff  // fail3R:    lui  $14, 0xFFFF
35 ce ff fa  //            ori  $14, 0xFFFA       # $r14 <-- 0xFFFFFFFA  (Fail flag 3 Rem)
00 00 00 0d  //            break
3c 0e ff ff  // fail4Q:    lui  $14, 0xFFFF
35 ce ff f9  //            ori  $14, 0xFFF9       # $r14 <-- 0xFFFFFFF9  (Fail flag 4 Quot)
00 00 00 0d  //            break
3c 0e ff ff  // fail4R:    lui  $14, 0xFFFF
35 ce ff f8  //            ori  $14, 0xFFF8       # $r14 <-- 0xFFFFFFF8  (Fail flag 4 Rem)
00 00 00 0d  //            break
```

```
Log File #10

BREAK INSTRUCTION FETCHED 1465.0 ns
R E G I S T E R ' S A F T E R B R E A K
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
Register File Content of $r0 - $r31
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
Time=1471.0 ns $r0  = 00000000 || Time=1471.0 ns $r16 = xxxxxxxx
Time=1471.0 ns $r1  = 00040911 || Time=1471.0 ns $r17 = xxxxxxxx
Time=1471.0 ns $r2  = 000003e8 || Time=1471.0 ns $r18 = xxxxxxxx
Time=1471.0 ns $r3  = fffbf6ef || Time=1471.0 ns $r19 = xxxxxxxx
Time=1471.0 ns $r4  = fffffc18 || Time=1471.0 ns $r20 = xxxxxxxx
Time=1471.0 ns $r5  = 00000108 || Time=1471.0 ns $r21 = xxxxxxxx
Time=1471.0 ns $r6  = 000001d1 || Time=1471.0 ns $r22 = xxxxxxxx
Time=1471.0 ns $r7  = fffffef8 || Time=1471.0 ns $
Time=1471.0 ns $r8  =  fffffe2f || Time=1471.0 ns
Time=1471.0 ns $r9  = 00000108 || Time=1471.0 ns $
Time=1471.0 ns $r10 = fffffe2f || Time=1471.0 ns $
Time=1471.0 ns $r11 = 00000000 || Time=1471.0 ns $
Time=1471.0 ns $r12 = 00000000 || Time=1471.0 ns $
Time=1471.0 ns $r13 = 00000000 || Time=1471.0 ns $r29 = 000003fc
Time=1471.0 ns $r14 = 00000000 || Time=1471.0 ns $r30 = xxxxxxxx
Time=1471.0 ns $r15 = 10010000 || Time=1471.0 ns $r31 = xxxxxxxx
```

R1 – R8 were loaded using the LW instructions with the content from data memory.

R9 & R10 were used to store data from the multiplications then get compared for the branches.
R11-R14: passed all the tests and get assigned with these values
R15: the source pointer, used as the base address to LW content from memory

```
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
Memory Location at M[0x3FC]
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

Time=1471.0 ns M[3F0]=xxxxxxxx

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
DATA MEMORY && I/O MEMORY of Memory locations 0xC0h to 0xFFh
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

Data Memory
Time=1471.0 ns DM[000000c0]  = xxxxxxxx
Time=1471.0 ns DM[000000c4]  = xxxxxxxx
Time=1471.0 ns DM[000000c8]  = xxxxxxxx
Time=1471.0 ns DM[000000cc]  = xxxxxxxx
Time=1471.0 ns DM[000000d0]  = xxxxxxxx
Time=1471.0 ns DM[000000d4]  = xxxxxxxx
Time=1471.0 ns DM[000000d8]  = xxxxxxxx
Time=1471.0 ns DM[000000dc]  = xxxxxxxx
Time=1471.0 ns DM[000000e0]  = xxxxxxxx
Time=1471.0 ns DM[000000e4]  = xxxxxxxx
Time=1471.0 ns DM[000000e8]  = xxxxxxxx
Time=1471.0 ns DM[000000ec]  = xxxxxxxx
Time=1471.0 ns DM[000000f0]  = xxxxxxxx
Time=1471.0 ns DM[000000f4]  = xxxxxxxx
Time=1471.0 ns DM[000000f8]  = xxxxxxxx
Time=1471.0 ns DM[000000fc]  = xxxxxxxx
```

Instruction Memory Module #11

```
@0
3c 0f 10 01  // main:      lui  $15, 0x1001
35 ef 00 c0  //            ori  $15, 0x00C0          # $r15 <-- 0x100100C0  (dest pointer)
20 01 ff 8a  //            addi $01, $00, -118       # $r01 <-- 0xFFFFFF8A
20 02 00 8a  //            addi $02  $00,  138       # $r02 <-- 0x0000008A
0c 10 00 1a  //            jal  sltiu_tests

3c 0d ff ff  //            lui  $13, 0xFFFF
35 ad 55 55  //            ori  $13, 0x5555          # $r13 <-- 0xFFFF5555  (pattern1)
3c 0c ff ff  //            lui  $12, 0xFFFF
35 8c fa f5  //            ori  $12, 0xFAF5          # $r12 <-- 0xFFFFFAF5  (pattern2)
3c 0b ff ff  //            lui  $11, 0xFFFF
35 6b ff ff  //            ori  $11, 0xFFFF          # $r11 <-- 0xFFFFFFFF  (pattern3)
3c 0a 00 00  //            lui  $10, 0x0000
35 4a f0 f0  //            ori  $10, 0xF0F0          # $r10 <-- 0x0000F0F0  (pattern4)

39 a9 aa aa  //            xori $09, $13, 0xAAAA     # $r09 <-- 0xFFFFFFFF
01 2b 40 22  //            sub  $08, $09, $11        # $r00 <-- 0
11 00 00 02  //            beq  $08, $00, xor_p1     # should branch
20 0e ff f9  //            addi $14, $00, -7         # fail flag7 r14 <-- FFFF_FFF9
00 00 00 0d  //            break
31 87 f5 fa  // xor_p1:    andi $07, $12, 0xF5FA     # $r07 <-- 0x0000F0F0
00 ea 40 22  //            sub  $08, $07, $10
11 00 00 02  //            beq  $08, $00, xor_p2     # should branch
20 0e ff f8  //            addi $14, $00, -8         # fail flag8 r14 <-- FFFF_FFF8
00 00 00 0d  //            break
ad e1 00 18  // xor_p2:    sw   $01, 0x18($15)       # M[D8] <-- FFFFFF8A
00 00 00 0d  //            break                     # should stop here, having
00 00 00 0d  //            break                     #   completed all the tests

             // sltiu_tests:
2c 23 ff 8b  //            sltiu  $03, $01, -117     # for unsigned# r01 < se(0xFF8B)
14 60 00 02  //            bne    $03, $00, slt1_p1  #   thus, we should branch
20 0e ff ff  //            addi   $14, $00, -1       # fail flag1 r14 <-- FFFF_FFFF
00 00 00 0d  //            break
20 04 00 c0  // slt1_p1:   addi   $04, $00, 0xC0     # pass flag1 M[C0] <-- C0
ad e4 00 00  //            sw     $04, 0x00($15)

2c 23 ff 89  //            sltiu  $03, $01, -119     # for unsigned# r01 !< se(0xFF89)
10 60 00 02  //            beq    $03, $00, slt_p2   #   thus, we should branch
20 0e ff fe  //            addi   $14, $00, -2       # fail flag2 r14 <-- FFFF_FFFE
00 00 00 0d  //            break
20 05 00 c4  // slt_p2:    addi   $05, $00, 0xC4     # pass flag2 M[C4] <-- C4
ad e5 00 04  //            sw     $05, 0x04($15)

2c 23 ff 8a  //            sltiu  $03, $01, -118     # for unsigned# r01 !< se(0xFF8A)
10 60 00 02  //            beq    $03, $00, slt_p3   #   thus, we should branch
20 0e ff fd  //            addi   $14, $00, -3       # fail flag3 r14 <-- FFFF_FFFD
00 00 00 0d  //            break
20 06 00 c8  // slt_p3:    addi   $06, $00, 0xC8     # pass flag3 M[C8] <-- C8
ad e6 00 08  //            sw     $06, 0x08($15)

2c 43 00 8b  //            sltiu  $03, $02, 0x008B   # for unsigned# r02 < se(0x008B)
14 60 00 02  //            bne    $03, $00, slt1_p4  #   thus, we should branch
20 0e ff fc  //            addi   $14, $00, -4       # fail flag4 r14 <-- FFFF_FFFC
00 00 00 0d  //            break
20 07 00 cc  // slt1_p4:   addi   $07, $00, 0xCC     # pass flag4 M[CC] <-- CC
ad e7 00 0c  //            sw     $07, 0x0C($15)

2c 43 00 89  //            sltiu  $03, $02, 0x0089   # for unsigned# r02 !< se(0x0089)
10 60 00 02  //            beq    $03, $00, slt_p5   #   thus, we should branch
20 0e ff fb  //            addi   $14, $00, -5       # fail flag5 r14 <-- FFFF_FFFB
00 00 00 0d  //            break
20 08 00 d0  // slt_p5:    addi   $08, $00, 0xD0     # pass flag5 M[D0] <-- D0
ad e8 00 10  //            sw     $08 0x10($15)

2c 43 00 8a  //            sltiu  $03, $02, 0x008A   # for unsigned# r02 !< se(0x008A)
10 60 00 02  //            beq    $03, $00, slt_p6   #   thus, we should branch
20 0e ff fa  //            addi   $14, $00, -6       # fail flag6 r14 <-- FFFF_FFFA
00 00 00 0d  //            break
20 06 00 d4  // slt_p6:    addi   $06, $00, 0xD4     # pass flag6 M[D4] <-- D4
ad e6 00 14  //            sw     $06, 0x14($15)
20 0e 00 00  //            addi   $14, $00, 0        # set $r14 to 0000_0000
03 e0 00 08  //            jr   $31                  # return from subroutine
```

Log File #11

```
BREAK INSTRUCTION FETCHED 1895.0 ns
R E G I S T E R ' S A F T E R B R E A K
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
Register File Content of $r0 - $r31
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
Time=1901.0 ns $r0  = 00000000 || Time=1901.0 ns $r16 = xxxxxxxx
Time=1901.0 ns $r1  = ffffff8a || Time=1901.0 ns $r17 = xxxxxxxx
Time=1901.0 ns $r2  = 0000008a || Time=1901.0 ns $r18 = xxxxxxxx
Time=1901.0 ns $r3  = 00000000 || Time=1901.0 ns $r19 = xxxxxxxx
Time=1901.0 ns $r4  = 000000c0 || Time=1901.0 ns $r20 = xxxxxxxx
Time=1901.0 ns $r5  = 000000c4 || Time=1901.0 ns $r21 = xxxxxxxx
Time=1901.0 ns $r6  = 000000d4 || Time=1901.0 ns $r22 = xxxxxxxx
Time=1901.0 ns $r7  = 0000f0f0 || Time=1901.0 ns $r23 = xxxxxxxx
Time=1901.0 ns $r8  = 00000000 || Time=1901.0 ns $r24 = xxxxxxxx
Time=1901.0 ns $r9  = ffffffff || Time=1901.0 ns $r25 = xxxxxxxx
Time=1901.0 ns $r10 = 0000f0f0 || Time=1901.0 ns $r26 = xxxxxxxx
Time=1901.0 ns $r11 = ffffffff || Time=1901.0 ns $r27 = xxxxxxxx
Time=1901.0 ns $r12 = ffffaf5  || Time=1901.0 ns $r28 = xxxxxxxx
Time=1901.0 ns $r13 = ffff5555 || Time=1901.0 ns $r29 = 000003fc
Time=1901.0 ns $r14 = 00000000 || Time=1901.0 ns $r30 = xxxxxxxx
Time=1901.0 ns $r15 = 100100c0 || Time=1901.0 ns $r31 = 00000014

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
Memory Location at M[0x3FC]
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-

Time=1901.0 ns M[3F0]=xxxxxxxx

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
DATA MEMORY && I/O MEMORY of Memory locations 0xC0h to 0xFFh
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-

Data Memory
Time=1901.0 ns DM[000000c0]  = 000000c0
Time=1901.0 ns DM[000000c4]  = 000000c4
Time=1901.0 ns DM[000000c8]  = 000000c8
Time=1901.0 ns DM[000000cc]  = 000000cc
Time=1901.0 ns DM[000000d0]  = 000000d0
Time=1901.0 ns DM[000000d4]  = 000000d4
Time=1901.0 ns DM[000000d8]  = ffffff8a
Time=1901.0 ns DM[000000dc]  = xxxxxxxx
Time=1901.0 ns DM[000000e0]  = xxxxxxxx
Time=1901.0 ns DM[000000e4]  = xxxxxxxx
Time=1901.0 ns DM[000000e8]  = xxxxxxxx
Time=1901.0 ns DM[000000ec]  = xxxxxxxx
Time=1901.0 ns DM[000000f0]  = xxxxxxxx
Time=1901.0 ns DM[000000f4]  = xxxxxxxx
Time=1901.0 ns DM[000000f8]  = xxxxxxxx
Time=1901.0 ns DM[000000fc]  = xxxxxxxx
```

R1 and R2 were initialized at the beginning and used to compare for loops

R3: the place holder register to assign either 0/1 when executing slt, slti, sltiu

These registers are temporary registers

These values get initialized to use for the branches

R14: when passed all stl test, R14 is reset to 0x0000_0000
R15: the source pointer, used as the base address to LW

@0x0000_00C0: used R3 content to store in memory by passing the first blt_test.
Only passing these flags will write the respective register to a specified memory location.

@0x0000_00D8: data from R1 get written to the specified memory location

Instruction Memory Module #12

```
@0
3c 0f 10 01  // main:      lui  $15, 0x1001
35 ef 00 c0  //            ori  $15, 0x00C0       # $r15 <-- 0x100100C0  (dest pointer)

20 01 ff 8a  //            addi $01, $00, -118    # $r01 <-- 0xFFFFFF8A
20 02 00 8a  //            addi $02  $00,  138    # $r02 <-- 0x0000008A
0c 10 00 08  //            jal  blt_tests
ad e1 00 18  //            sw   $01, 0x18($15)    # M[D8] <-- 0xFFFFFF8A
ad e2 00 1c  //            sw   $02, 0x1C($15)    # M[DC] <-- 0x0000008A
00 00 00 0d  //            break

18 20 00 02  // blt_tests: blez $01, blez_p1       # this should branch
20 0e ff ff  //            addi $14, $00, -1      # fail flag1 r14 <-- FFFF_FFFF
00 00 00 0d  //            break
20 03 00 c0  // blez_p1:   addi $03, $00, 0xC0    # pass flag1 M[C0] <-- C0
ad e3 00 00  //            sw   $03, 0x00($15)
18 40 00 03  //            blez $02, blez_f2      # this should not branch
20 04 00 c4  //            addi $04, $00, 0xC4    # pass flag2 M[C4] <-- C4
ad e4 00 04  //            sw   $04, 0x04($15)
08 10 00 13  //            j    blez_p2
20 0e ff fe  // blez_f2:   addi $14, $00, -2      # fail flag2 r14 <-- FFFF_FFFE
00 00 00 0d  //            break
18 00 00 02  // blez_p2:   blez $0, blez_p3       # this should branch
20 0e ff fd  //            addi $14, $00, -3      # fail flag3 r14 <-- FFFF_FFFD
00 00 00 0d  //            break
20 05 00 c8  // blez_p3:   addi $05, $00, 0xC8    # pass flag3 M[C8] <-- C8
ad e5 00 08  //            sw   $05, 0x08($15)

1c 40 00 02  //            bgtz $02, bgtz_p1      # this should pass
20 0e ff fc  //            addi $14, $00, -4      # fail flag3 r14 <-- FFFF_FFFC
00 00 00 0d  //            break
20 06 00 cc  // bgtz_p1:   addi $06, $00, 0xCC    # pass flag4 M[C0] <-- CC
ad e6 00 0c  //            sw   $06, 0x0C($15)
1c 20 00 03  //            bgtz $01, bgtz_f2      # this should not branch
20 07 00 d0  //            addi $07, $00, 0xD0    # pass flag5 M[D0] <-- D0
ad e7 00 10  //            sw   $07, 0x10($15)
08 10 00 23  //            j    bgtz_p2
20 0e ff fb  // bgtz_f2:   addi $14, $00, -5      # fail flag5 r14 <-- FFFF_FFFB
00 00 00 0d  //            break
1c 20 00 03  // bgtz_p2:   bgtz $01, bgtz_f3      # this should not branch
20 08 00 d4  //            addi $08, $00, 0xD4    # pass flag6 M[D0] <-- D4
ad e8 00 14  //            sw   $08, 0x14($15)
08 10 00 29  //            j    bgtz_p3
20 0e ff fa  // bgtz_f3:   addi $14, $00, -6      # fail flag6 r14 <-- FFFF_FFFA
00 00 00 0d  //            break
20 0e 00 00  // bgtz_p3:   addi $14, $00, 0       # set $r14 to 0000_0000
03 e0 00 08  //            jr   $31               # return from subroutine
```

```
Log File #12

BREAK INSTRUCTION FETCHED 1225.0 ns
R E G I S T E R ' S A F T E R B R E A K*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
Register File Content of $r0 - $r31
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
Time=1231.0 ns $r0  = 00000000 || Time=1231.0 ns $r16 = xxxxxxxx
Time=1231.0 ns $r1  = ffffff8a || Time=1231.0 ns $r17 = xxxxxxxx
Time=1231.0 ns $r2  = 0000008a || Time=1231.0 ns $r18 = xxxxxxxx
Time=1231.0 ns $r3  = 000000c0 || Time=1231.0 ns $r19 = xxxxxxxx
Time=1231.0 ns $r4  = 000000c4 || Time=1231.0 ns $r20 = xxxxxxxx
Time=1231.0 ns $r5  = 000000c8 || Time=1231.0 ns $r21 = xxxxxxxx
Time=1231.0 ns $r6  = 000000cc || Time=1231.0 ns $r22 = xxxxxxxx
Time=1231.0 ns $r7  = 000000d0 || Time=1231.0 ns $r23 = xxxxxxxx
Time=1231.0 ns $r8  = 000000d4 || Time=1231.0 ns $r24 = xxxxxxxx
Time=1231.0 ns $r9  = xxxxxxxx || Time=1231.0 ns $r25 = xxxxxxxx
Time=1231.0 ns $r10 = xxxxxxxx || Time=1231.0 ns $r26 = xxxxxxxx
Time=1231.0 ns $r11 = xxxxxxxx || Time=1231.0 ns $r27 = xxxxxxxx
Time=1231.0 ns $r12 = xxxxxxxx || Time=1231.0 ns $r28 = xxxxxxxx
Time=1231.0 ns $r13 = xxxxxxxx || Time=1231.0 ns $r29 = 000003fc
Time=1231.0 ns $r14 = 00000000 || Time=1231.0 ns $r30 = xxxxxxxx
Time=1231.0 ns $r15 = 100100c0 || Time=1231.0 ns $r31 = 00000014

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
Memory Location at M[0x3FC]
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

Time=1231.0 ns M[3F0]=xxxxxxxx

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-
DATA MEMORY && I/O MEMORY of Memory locations 0xC0h to 0xFFh
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-

Data Memory
Time=1231.0 ns DM[000000c0] = 000000c0
Time=1231.0 ns DM[000000c4] = 000000c4
Time=1231.0 ns DM[000000c8] = 000000c8
Time=1231.0 ns DM[000000cc] = 000000cc
Time=1231.0 ns DM[000000d0] = 000000d0
Time=1231.0 ns DM[000000d4] = 000000d4
Time=1231.0 ns DM[000000d8] = ffffff8a
Time=1231.0 ns DM[000000dc] = 0000008a
Time=1231.0 ns DM[000000e0] = xxxxxxxx
Time=1231.0 ns DM[000000e4] = xxxxxxxx
Time=1231.0 ns DM[000000e8] = xxxxxxxx
Time=1231.0 ns DM[000000ec] = xxxxxxxx
Time=1231.0 ns DM[000000f0] = xxxxxxxx
Time=1231.0 ns DM[000000f4] = xxxxxxxx
Time=1231.0 ns DM[000000f8] = xxxxxxxx
Time=1231.0 ns DM[000000fc] = xxxxxxxx
```

R1 and R2 were initialized at the beginning and used to compare for loops

These registers are temporary registers, only get written when passing certain branches

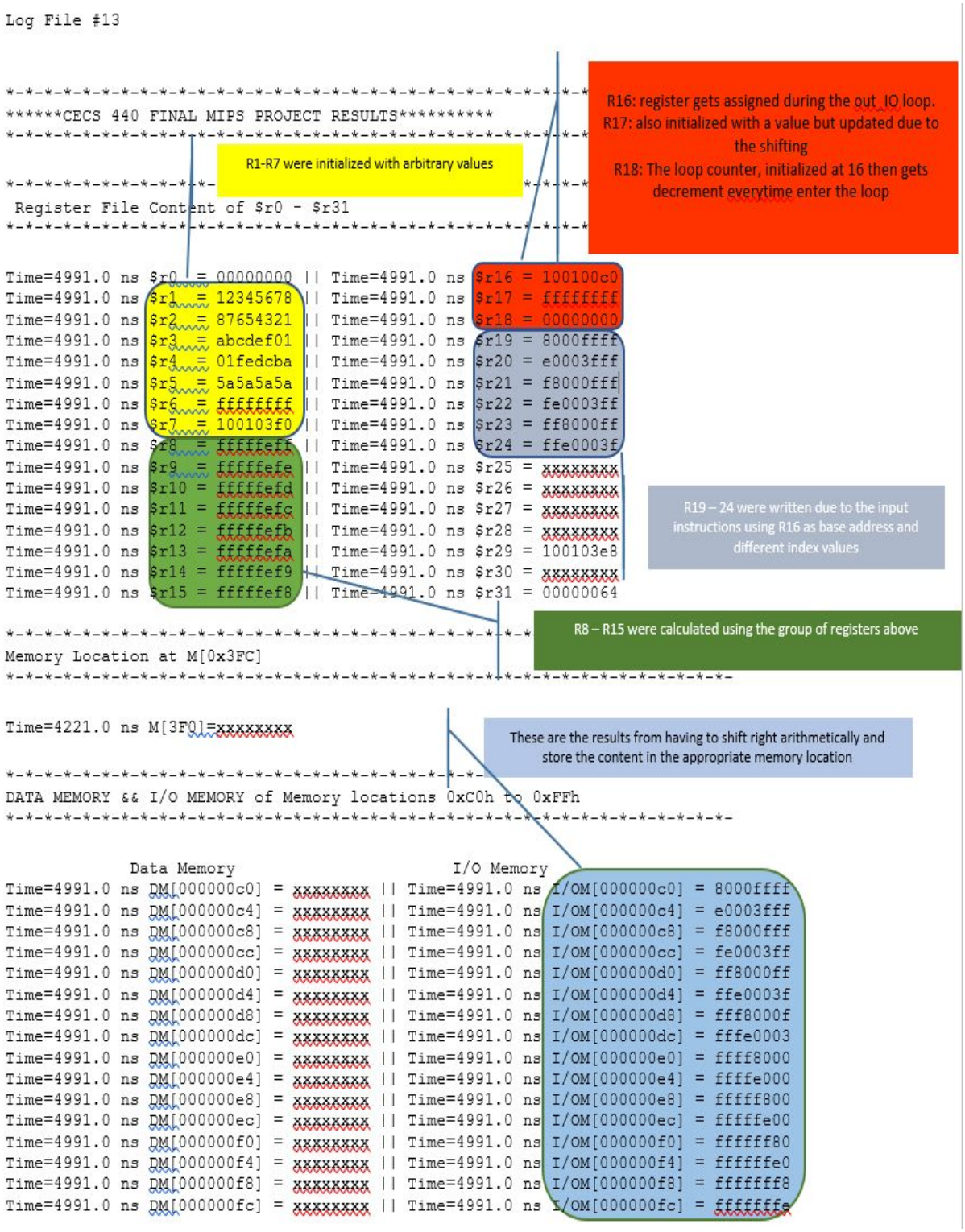
R14: If passed all branch tests, then set the register to 0x0000_0000
R15: the source pointer, used as the base address to LW content

@0x0000_00C0: used R3 content to store in memory by passing the first blt_test
Only passing these flags will write the respective register to a specified memory location.

@0x0000_00D8 & 0x0000_00DC only get written to afterwards. The data stored are from R1 and R2.

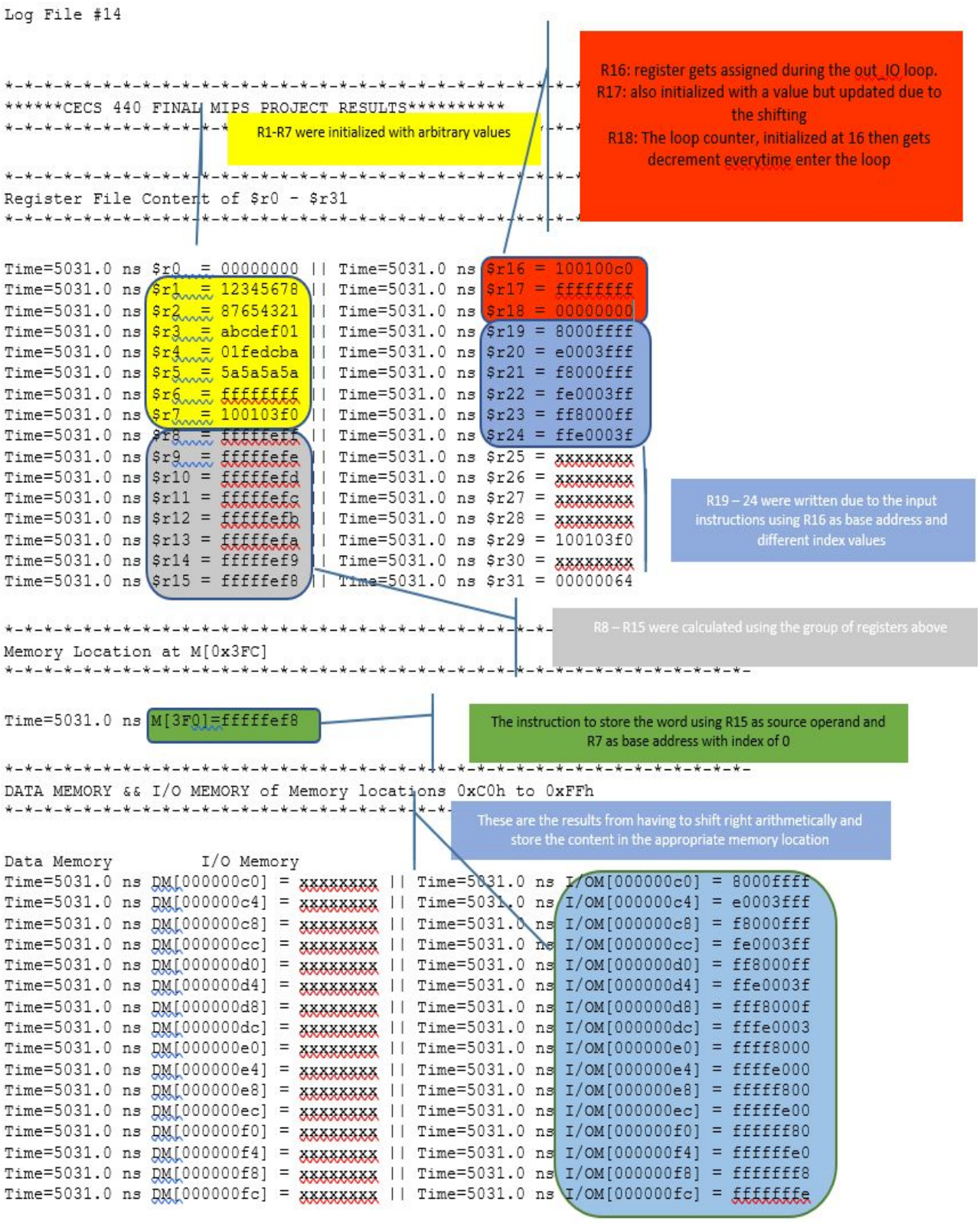
Instruction Memory Module #13

```
@0
00 00 00 1f  // main:      setie
3c 01 12 34  //            lui  $01, 0x1234
34 21 56 78  //            ori  $01, 0x5678      # LI  R01,  0x12345678
3c 02 87 65  //            lui  $02, 0x8765
34 42 43 21  //            ori  $02, 0x4321      # LI  R02,  0x87654321
3c 03 ab cd  //            lui  $03, 0xABCD
34 63 ef 01  //            ori  $03, 0xEF01      # LI  R03,  0xABCDEF01
3c 04 01 fe  //            lui  $04, 0x01FE
34 84 dc ba  //            ori  $04, 0xDCBA      # LI  R04,  0x01FEDCBA
3c 05 5a 5a  //            lui  $05, 0x5A5A
34 a5 5a 5a  //            ori  $05, 0x5A5A      # LI  R05,  0x5A5A5A5A
3c 06 ff ff  //            lui  $06, 0xFFFF
34 c6 ff ff  //            ori  $06, 0xFFFF      # LI  R06,  0xFFFFFFFF
3c 07 ff ff  //            lui  $07, 0xFFFF
34 e7 ff 00  //            ori  $07, 0xFF00      # LI  R07,  0xFFFFFF00

00 c7 40 20  //            add  $08, $06, $07
00 c8 48 20  //            add  $09, $06, $08
00 c9 50 20  //            add  $10, $06, $09
00 ca 58 20  //            add  $11, $06, $10
00 cb 60 20  //            add  $12, $06, $11
00 cc 68 20  //            add  $13, $06, $12
00 cd 70 20  //            add  $14, $06, $13
00 ce 78 20  //            add  $15, $06, $14

3c 07 10 01  //            lui  $07, 0x1001
34 e7 03 f0  //            ori  $07, 0x03F0      # LI   R07,  0x100103F0
ac ef 00 00  //            sw   $15, 0($07)      # ST  [R07], R15
00 00 00 0d  //            break

@200
                        //*********************************************
                        // In this ISR, we will implement writing
                        // some patterns to the IO space, and then
                        // reading them back.
                        // Note: this "ISR" expects the "return address"
                        //       to have been saved in $ra (not the stack)
                        //*********************************************
3c 10 10 01  // isr:       lui    $16, 0x1001      #load destination IO address
36 10 00 c0  //             ori    $16, 0x00C0      #  0x100100C0 into r16
3c 11 80 00  //             lui    $17, 0x8000      #initialize the pattern of
36 31 ff ff  //             ori    $17, 0xFFFF      #  0x8000FFFF into r17
20 12 00 10  //             addi   $18, $0, 0x10    #loop counter set to 16

76 11 00 00  // out_IO:    output $17, 0($16)      # output  [R16], R17
00 11 88 83  //             sra    $17, $17, 2      # change the pattern by shifting twice
22 10 00 04  //             addi   $16, $16, 4      # increment the memory pointer 4 bytes
22 52 ff ff  //             addi   $18, $18, -1     # decrement the loop counter
16 40 ff fb  //             bne    $18, $00, out_IO #  and jmp to top if not finished

3c 10 10 01  //             lui    $16, 0x1001      #load source IO address
36 10 00 c0  //             ori    $16, 0x00C0      #  0x100100C0 into r16
72 13 00 00  //             input  $19,  0($16)     #  and input from 6
72 14 00 04  //             input  $20,  4($16)     #    the IO locations,
72 15 00 08  //             input  $21,  8($16)     #    starting from 0xC0
72 16 00 0c  //             input  $22, 12($16)
72 17 00 10  //             input  $23, 16($16)
72 18 00 14  //             input  $24, 20($16)
03 e0 00 08  //             jr     $31              # return from interrupt (v1, using $ra)
```


Log File #13
******CECS 440 FINAL MIPS PROJECT RESULTS**********
R1-R7 were initialized with arbitrary values
R16: register gets assigned during the out_IO loop.
R17: also initialized with a value but updated due to the shifting
R18: The loop counter, initialized at 16 then gets decrement everytime enter the loop
Register File Content of $r0 - $r31
Time=4991.0 ns $r0 = 00000000 || Time=4991.0 ns $r16 = 100100c0
Time=4991.0 ns $r1 = 12345678 || Time=4991.0 ns $r17 = ffffffff
Time=4991.0 ns $r2 = 87654321 || Time=4991.0 ns $r18 = 00000000
Time=4991.0 ns $r3 = abcdef01 || Time=4991.0 ns $r19 = 8000ffff
Time=4991.0 ns $r4 = 01fedcba || Time=4991.0 ns $r20 = e0003fff
Time=4991.0 ns $r5 = 5a5a5a5a || Time=4991.0 ns $r21 = f8000fff
Time=4991.0 ns $r6 = ffffffff || Time=4991.0 ns $r22 = fe0003ff
Time=4991.0 ns $r7 = 100103f0 || Time=4991.0 ns $r23 = ff8000ff
Time=4991.0 ns $r8 = fffffeff || Time=4991.0 ns $r24 = ffe0003f
Time=4991.0 ns $r9 = fffffefe || Time=4991.0 ns $r25 = xxxxxxxx
Time=4991.0 ns $r10 = fffffefd || Time=4991.0 ns $r26 = xxxxxxxx
Time=4991.0 ns $r11 = fffffefc || Time=4991.0 ns $r27 = xxxxxxxx
Time=4991.0 ns $r12 = fffffefb || Time=4991.0 ns $r28 = xxxxxxxx
Time=4991.0 ns $r13 = fffffefa || Time=4991.0 ns $r29 = 100103e8
Time=4991.0 ns $r14 = fffffef9 || Time=4991.0 ns $r30 = xxxxxxxx
Time=4991.0 ns $r15 = fffffef8 || Time=4991.0 ns $r31 = 00000064
R19 – 24 were written due to the input instructions using R16 as base address and different index values
R8 – R15 were calculated using the group of registers above
Memory Location at M[0x3FC]
Time=4221.0 ns M[3F0]=xxxxxxxx
These are the results from having to shift right arithmetically and store the content in the appropriate memory location
DATA MEMORY && I/O MEMORY of Memory locations 0xC0h to 0xFFh
Data Memory
I/O Memory
Time=4991.0 ns DM[000000c0] = xxxxxxxx || Time=4991.0 ns I/OM[000000c0] = 8000ffff
Time=4991.0 ns DM[000000c4] = xxxxxxxx || Time=4991.0 ns I/OM[000000c4] = e0003fff
Time=4991.0 ns DM[000000c8] = xxxxxxxx || Time=4991.0 ns I/OM[000000c8] = f8000fff
Time=4991.0 ns DM[000000cc] = xxxxxxxx || Time=4991.0 ns I/OM[000000cc] = fe0003ff
Time=4991.0 ns DM[000000d0] = xxxxxxxx || Time=4991.0 ns I/OM[000000d0] = ff8000ff
Time=4991.0 ns DM[000000d4] = xxxxxxxx || Time=4991.0 ns I/OM[000000d4] = ffe0003f
Time=4991.0 ns DM[000000d8] = xxxxxxxx || Time=4991.0 ns I/OM[000000d8] = fff8000f
Time=4991.0 ns DM[000000dc] = xxxxxxxx || Time=4991.0 ns I/OM[000000dc] = fffe0003
Time=4991.0 ns DM[000000e0] = xxxxxxxx || Time=4991.0 ns I/OM[000000e0] = ffff8000
Time=4991.0 ns DM[000000e4] = xxxxxxxx || Time=4991.0 ns I/OM[000000e4] = ffffe000
Time=4991.0 ns DM[000000e8] = xxxxxxxx || Time=4991.0 ns I/OM[000000e8] = fffff800
Time=4991.0 ns DM[000000ec] = xxxxxxxx || Time=4991.0 ns I/OM[000000ec] = ffffe00
Time=4991.0 ns DM[000000f0] = xxxxxxxx || Time=4991.0 ns I/OM[000000f0] = ffffff80
Time=4991.0 ns DM[000000f4] = xxxxxxxx || Time=4991.0 ns I/OM[000000f4] = fffffe0
Time=4991.0 ns DM[000000f8] = xxxxxxxx || Time=4991.0 ns I/OM[000000f8] = ffffff8
Time=4991.0 ns DM[000000fc] = xxxxxxxx || Time=4991.0 ns I/OM[000000fc] = fffffe

Instruction Memory Module #14

```
@0
00 00 00 1f  // main:      setie
3c 01 12 34  //            lui  $01, 0x1234
34 21 56 78  //            ori  $01, 0x5678      # LI  R01,  0x12345678
3c 02 87 65  //            lui  $02, 0x8765
34 42 43 21  //            ori  $02, 0x4321      # LI  R02,  0x87654321
3c 03 ab cd  //            lui  $03, 0xABCD
34 63 ef 01  //            ori  $03, 0xEF01      # LI  R03,  0xABCDEF01
3c 04 01 fe  //            lui  $04, 0x01FE
34 84 dc ba  //            ori  $04, 0xDCBA      # LI  R04,  0x01FEDCBA
3c 05 5a 5a  //            lui  $05, 0x5A5A
34 a5 5a 5a  //            ori  $05, 0x5A5A      # LI  R05,  0x5A5A5A5A
3c 06 ff ff  //            lui  $06, 0xFFFF
34 c6 ff ff  //            ori  $06, 0xFFFF      # LI  R06,  0xFFFFFFFF
3c 07 ff ff  //            lui  $07, 0xFFFF
34 e7 ff 00  //            ori  $07, 0xFF00      # LI  R07,  0xFFFFFF00

00 c7 40 20  //            add  $08, $06, $07
00 c8 48 20  //            add  $09, $06, $08
00 c9 50 20  //            add  $10, $06, $09
00 ca 58 20  //            add  $11, $06, $10
00 cb 60 20  //            add  $12, $06, $11
00 cc 68 20  //            add  $13, $06, $12
00 cd 70 20  //            add  $14, $06, $13
00 ce 78 20  //            add  $15, $06, $14

3c 07 10 01  //            lui  $07, 0x1001
34 e7 03 f0  //            ori  $07, 0x03F0      # LI   R07,  0x100103F0
ac ef 00 00  //            sw   $15, 0($07)      # ST  [R07], R15
00 00 00 0d  //            break

@200
                        //*********************************************
                        // In this ISR, we will implement writing
                        // some patterns to the IO space, and then
                        // reading them back.
                        // Note: this "ISR" expects the "return address"
                        //       to have been saved on the stack (not $ra)
                        //*********************************************
3c 10 10 01  // isr:       lui    $16, 0x1001      #load destination IO address
36 10 00 c0  //             ori    $16, 0x00C0      #  0x100100C0 into r16
3c 11 80 00  //             lui    $17, 0x8000      #initialize the pattern of
36 31 ff ff  //             ori    $17, 0xFFFF      #  0x8000FFFF into r17
20 12 00 10  //             addi   $18, $0, 0x10    #loop counter set to 16

76 11 00 00  // out_IO:    output $17, 0($16)       # output  [R16], R17
00 11 88 83  //             sra    $17, $17, 2      # change the pattern by shifting twice
22 10 00 04  //             addi   $16, $16, 4      # increment the memory pointer 4 bytes
22 52 ff ff  //             addi   $18, $18, -1     # decrement the loop counter
16 40 ff fb  //             bne    $18, $00, out_IO #  and jmp to top if not finished

3c 10 10 01  //             lui    $16, 0x1001      #load source IO address
36 10 00 c0  //             ori    $16, 0x00C0      #  0x100100C0 into r16
72 13 00 00  //             input  $19,  0($16)     #  and input from 6
72 14 00 04  //             input  $20,  4($16)     #    the IO locations,
72 15 00 08  //             input  $21,  8($16)     #    starting from 0xC0
72 16 00 0c  //             input  $22, 12($16)
72 17 00 10  //             input  $23, 16($16)
72 18 00 14  //             input  $24, 20($16)
7B A0 00 00  //             reti                    # return from interrupt (v2, using M[sp] as saved PC)
                                                    # Note opcode=0x1E and $rs=0x1D, i.e. $sp
```


Log File #14
******CECS 440 FINAL MIPS PROJECT RESULTS**********
R1-R7 were initialized with arbitrary values
R16: register gets assigned during the out_IO loop.
R17: also initialized with a value but updated due to the shifting
R18: The loop counter, initialized at 16 then gets decrement everytime enter the loop
Register File Content of $r0 - $r31
Time=5031.0 ns $r0 = 00000000 || Time=5031.0 ns $r16 = 100100c0
Time=5031.0 ns $r1 = 12345678 || Time=5031.0 ns $r17 = ffffffff
Time=5031.0 ns $r2 = 87654321 || Time=5031.0 ns $r18 = 00000000
Time=5031.0 ns $r3 = abcdef01 || Time=5031.0 ns $r19 = 8000ffff
Time=5031.0 ns $r4 = 01fedcba || Time=5031.0 ns $r20 = e0003fff
Time=5031.0 ns $r5 = 5a5a5a5a || Time=5031.0 ns $r21 = f8000fff
Time=5031.0 ns $r6 = ffffffff || Time=5031.0 ns $r22 = fe0003ff
Time=5031.0 ns $r7 = 100103f0 || Time=5031.0 ns $r23 = ff8000ff
Time=5031.0 ns $r8 = fffffeff || Time=5031.0 ns $r24 = ffe0003f
Time=5031.0 ns $r9 = fffffefe || Time=5031.0 ns $r25 = xxxxxxxx
Time=5031.0 ns $r10 = fffffefd || Time=5031.0 ns $r26 = xxxxxxxx
Time=5031.0 ns $r11 = fffffefc || Time=5031.0 ns $r27 = xxxxxxxx
Time=5031.0 ns $r12 = fffffefb || Time=5031.0 ns $r28 = xxxxxxxx
Time=5031.0 ns $r13 = fffffefa || Time=5031.0 ns $r29 = 100103f0
Time=5031.0 ns $r14 = fffffef9 || Time=5031.0 ns $r30 = xxxxxxxx
Time=5031.0 ns $r15 = fffffef8 || Time=5031.0 ns $r31 = 00000064
R19 – 24 were written due to the input instructions using R16 as base address and different index values
R8 – R15 were calculated using the group of registers above
Memory Location at M[0x3FC]
Time=5031.0 ns M[3F0]=fffffef8
The instruction to store the word using R15 as source operand and R7 as base address with index of 0
DATA MEMORY && I/O MEMORY of Memory locations 0xC0h to 0xFFh
These are the results from having to shift right arithmetically and store the content in the appropriate memory location
Data Memory I/O Memory
Time=5031.0 ns DM[000000c0] = xxxxxxxx || Time=5031.0 ns I/OM[000000c0] = 8000ffff
Time=5031.0 ns DM[000000c4] = xxxxxxxx || Time=5031.0 ns I/OM[000000c4] = e0003fff
Time=5031.0 ns DM[000000c8] = xxxxxxxx || Time=5031.0 ns I/OM[000000c8] = f8000fff
Time=5031.0 ns DM[000000cc] = xxxxxxxx || Time=5031.0 ns I/OM[000000cc] = fe0003ff
Time=5031.0 ns DM[000000d0] = xxxxxxxx || Time=5031.0 ns I/OM[000000d0] = ff8000ff
Time=5031.0 ns DM[000000d4] = xxxxxxxx || Time=5031.0 ns I/OM[000000d4] = ffe0003f
Time=5031.0 ns DM[000000d8] = xxxxxxxx || Time=5031.0 ns I/OM[000000d8] = fff8000f
Time=5031.0 ns DM[000000dc] = xxxxxxxx || Time=5031.0 ns I/OM[000000dc] = fffe0003
Time=5031.0 ns DM[000000e0] = xxxxxxxx || Time=5031.0 ns I/OM[000000e0] = ffff8000
Time=5031.0 ns DM[000000e4] = xxxxxxxx || Time=5031.0 ns I/OM[000000e4] = ffffe000
Time=5031.0 ns DM[000000e8] = xxxxxxxx || Time=5031.0 ns I/OM[000000e8] = fffff800
Time=5031.0 ns DM[000000ec] = xxxxxxxx || Time=5031.0 ns I/OM[000000ec] = fffffe00
Time=5031.0 ns DM[000000f0] = xxxxxxxx || Time=5031.0 ns I/OM[000000f0] = ffffff80
Time=5031.0 ns DM[000000f4] = xxxxxxxx || Time=5031.0 ns I/OM[000000f4] = fffffffe0
Time=5031.0 ns DM[000000f8] = xxxxxxxx || Time=5031.0 ns I/OM[000000f8] = fffffff8
Time=5031.0 ns DM[000000fc] = xxxxxxxx || Time=5031.0 ns I/OM[000000fc] = fffffffe

IV. Hardware Implementation

MIPS Test Bench

CPU

Instruction Unit

Datapath

ALU

MCU_1

MCU_2

MCU_3

MCU_4

MCU_5

MCU_6

MCU_7

MCU_8

MCU_9

MCU_10

V. Additional Discussion and/or Comments

Overall, the project was completed on time with some additional instructions and more details to the ISA. We were able to add extra instructions like CLR, MOV, NOP, POP, PUSH, and branches like BLT(Branch Less Than) and BGE(Branch Greater Than or Equal). The block diagrams that we created had detailed all the inputs, outputs, and wires in a “top-down” hierarchical approach. We also added a state diagram of each state in our MCU(MIPS Control Unit) which is found in end of the block diagrams. If we had more time, we would would redesigned our project to have pipelining to make the processor obtain a more efficient CPI and have additional instructions that would have required the datapath to be changed. The other task we could have done is add virtual memory and cache memory within the IO Memory and Data Memory.