New Horizon

Architecure For Programmers

CECS 440 Computer Architecture

M/W 10 – 12pm

Instructor: R.W Allison

Due : Nov 14, 2016

Kassandra Flores

ID: 010313626

Adan Hernandez

ID:012232894

I Table of Contents

I Purpose

I.A Abstract

To serve as users' documentation to the New Horizon processor. The following sections will be addressed in the *Architecture For Programmers* document:

1. Registers available to the user
2. Data types
3. Instruction and instruction formats of the baseline MIPS architecture and the enhanced New Horizon architecture
4. Memory Architecture, Interrupts and Reset.
5. Processor verification and test results

I.B Typographical Conventions

The following typographical conventions are used throughout this document. They are intended to help in understanding of all material presented.

I.B.1 Italic Text

- Is used for emphasis

I.B.2 Bold Text

- Indicates that a term is being **defined**

I.B.3 Courier Text

- Will be used to define data sizes and data bus names

I.C Special Symbols Used

Specials symbols throughout this document are used in attempt to demonstrate in a pseudo-language what is being attempted

Symbols	Description
←	Assignment
=	Assignment
==,!=	Tests for equality and inequality
b’	Base 2 representation of a value
h’	Base 16 representation of a value
{#{#z}}	Replication operation, Where # is the number of copies to make of #z
0x#	Base 16 representation of a value
/	Division
%	Modules
+.-	Addition, Subtraction
>>	Logical Shift Right
<<	Logical Shift Left
>>>	Logical Shift Right Arithmetic
$	Preceding register
{a,b}	Concatenation of all sources within brackets. Where resulting value is equal in data size to sum of all sources concatenated together
S.E	Sign Extension to 32 bit value
OP	Operation
PC	Program Counter
PCP4	Program Counter plus 4
>> rot, << rot	Shift left with rotate , Shift right with rotate

II Instruction Set Architecture

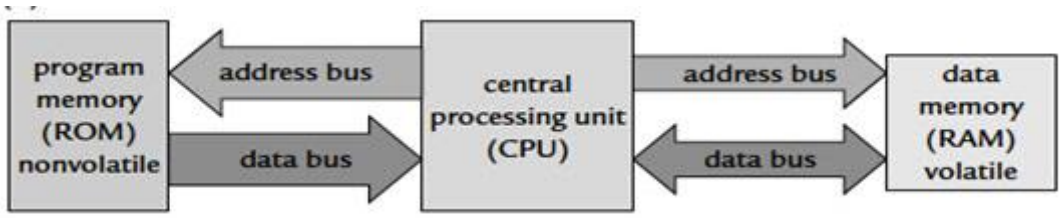
In the following section the below categories will be addressed:

1. Type of memory architecture implement.
2. Memory address space
3. Addressing Memory
4. Machine Register Set
5. Data Types
6. Addressing
7. Instruction Sets

II.A Harvard Memory Architecture and Organization

This New Horizon processor uses the ***Harvard Memory Architecture*** in its memory structure; having two separate memory modules for data and instruction. This architecture allows for instructions to be executed while data transfer is occurring since data transfer and instruction execution use separate busses to compete these actions.

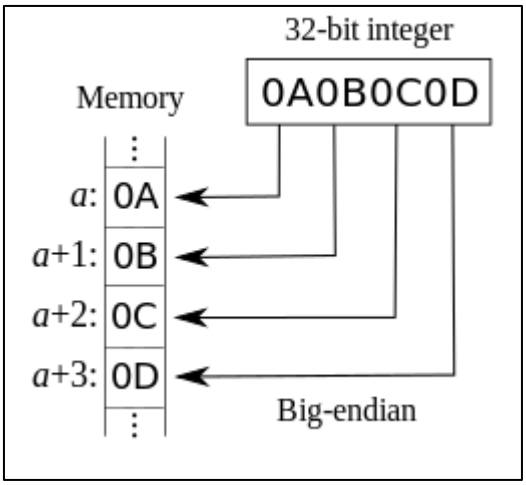
Figure 1



II.A.1 Address Space

The New Horizon uses a 32-bit **address space** with address starting at 0x00 to 0x3FF (minimum of 1024) for running code execution simulations.

Figure 2



II.A.2 Byte Address

All instruction and data memory locations are **byte addressable**, where at each memory address a byte is held. In this memory architecture, all 32 bit data and 32 bit instruction data are saved into 4 consecutive memory byte locations and therefore are **word aligned**

Data is stored into memory in **big endian** format, where the most significant byte is placed into the smaller address value location and the most significant byte is placed in the largest address value location.

II.B Machine Register Set

II.B.1 General Purpose Registers

There are *32 32-bit wide integer registers* available to the user on the New Horizon processor. Only *Register 0* cannot have a value written to it, it will always have a constant value of zero. Aside from *Register 0*, these registers can be written to and are capable of holding 32 bits of data.

The table below shows all register names, use and call convention.

Table 1

Name	Number	Use	Reserved after a call
$zero	0	The Constant Value 0	N.A
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for function results and Express Evaluations	No
$a0 - $a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Temporaries	Yes
$t8-$t9	24-25	Temporaries	No
$k0-$k1	26-27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	Yes

II.B.2 Program Counter

The New Horizon MIPS architecture has a **program counter** (PC) which is a 32-bit register. This register stores the address of the next instruction that will be executed at run time. When the processor fetches and instruction form memory, the PC will increment by 4 which will result in it pointing to the next instruction to be fetched.

II.B.3 Flag Registers

The following flag registers: C, V, N, Z, are registers that store the status of **arithmetic logic unit** when performing arithmetic operations. Below is an explanation of each flag register type and its definition.

Interrupt Enable (I):

Flag indicating if interrupts are enabled and if they will be acknowledged by the processor.

Carry (C):

Indicates if the calculated result of the arithmetic or logical operation causes a carry out to occur.

Overflow (V):

Indicates if the result of an arithmetic operation does not fit within the 32-bit register data size.

Negative (N):

Indicates if the result of a arithmetic operation produced a negative value or if the value passed through the arithmetic logic unit is a negative number.

Zero (Z):

Indicates if the result of a arithmetic operation produced a zero value or if the value passed through the arithmetic logic unit was of a zero value.

II.C Data Types

The New Horizon processor operates on the following data types: 32-bit **signed** and **unsigned** integers.

II.C.1 Signed Integers

A maximum value of – 2,147,483,648 to2147483647 can be represented in the available 32 bit data size registers. Determining if the value stored in the register is by the **most significant bit (msb).** If the msb is 1, then the value stored in the register is a negative number, else it is an unsigned integer.

Determining the value of the negative number present in the register can be accomplished using **2's compliment.**

II.C.2 Unsigned Integers

A maximum value of 0 to 4294967295 can be represented on the 32 bit data registers. Since the msb of the 32 bit value is no longer being used as indicator if the value is negative or positive, reason being that unsigned integers can only be positive, the msb if free to be used to increase the size we are able to represent on the 32 bit registers.

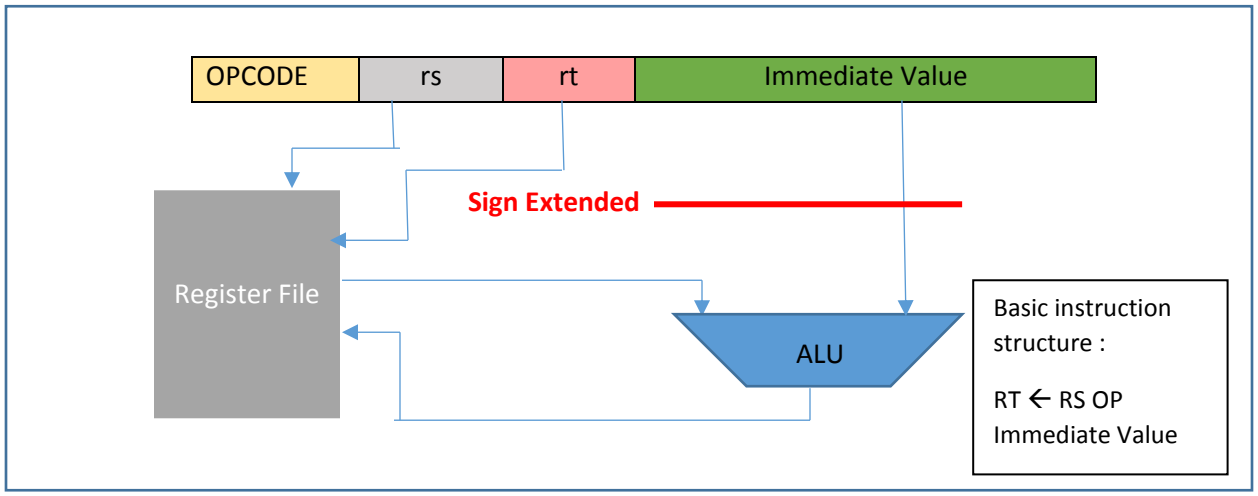
II.D Addressing Modes

II.D.1 Immediate Addressing

Immediate addressing is when one of the operands itself (not the address of a operand) is being used in the instruction. Typical uses of this addressing mode are to initialize contents of registers with data or address values .

The following diagram shows the basic logic when executing in immediate addressing mode:

Figure 3



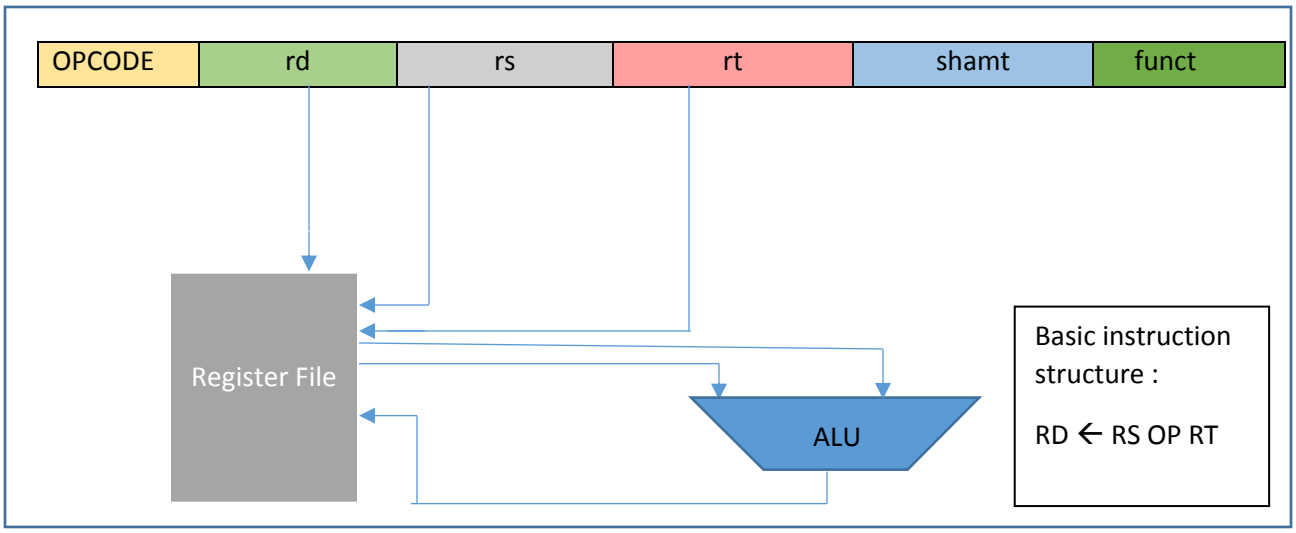
This figure shows that the immediate value is being used directly in a arithmetic operation alongside the register contents of the register specified in the rt field and having the product of this operation being stored in the register specified by the rs field.

Note : At this state of the document , the structure of instructions hasn't been introduced . The instruction structure introduced in this diagram will be further explained in section: I- Type Instructions

II.D.2 Register Addressing

In register addressing mode, all operands and destination location are CPU registers.

Figure 4



This figure shows that values in registers specified by the rs and rt field are being used directly in a arithmetic operation. The product of this operation is then stored in the register specified by the rd field.

Note: At this state of the document , the structure of instructions hasn't been introduced . The instruction structure introduced in this diagram will be further explained in section: R- Type Instructions

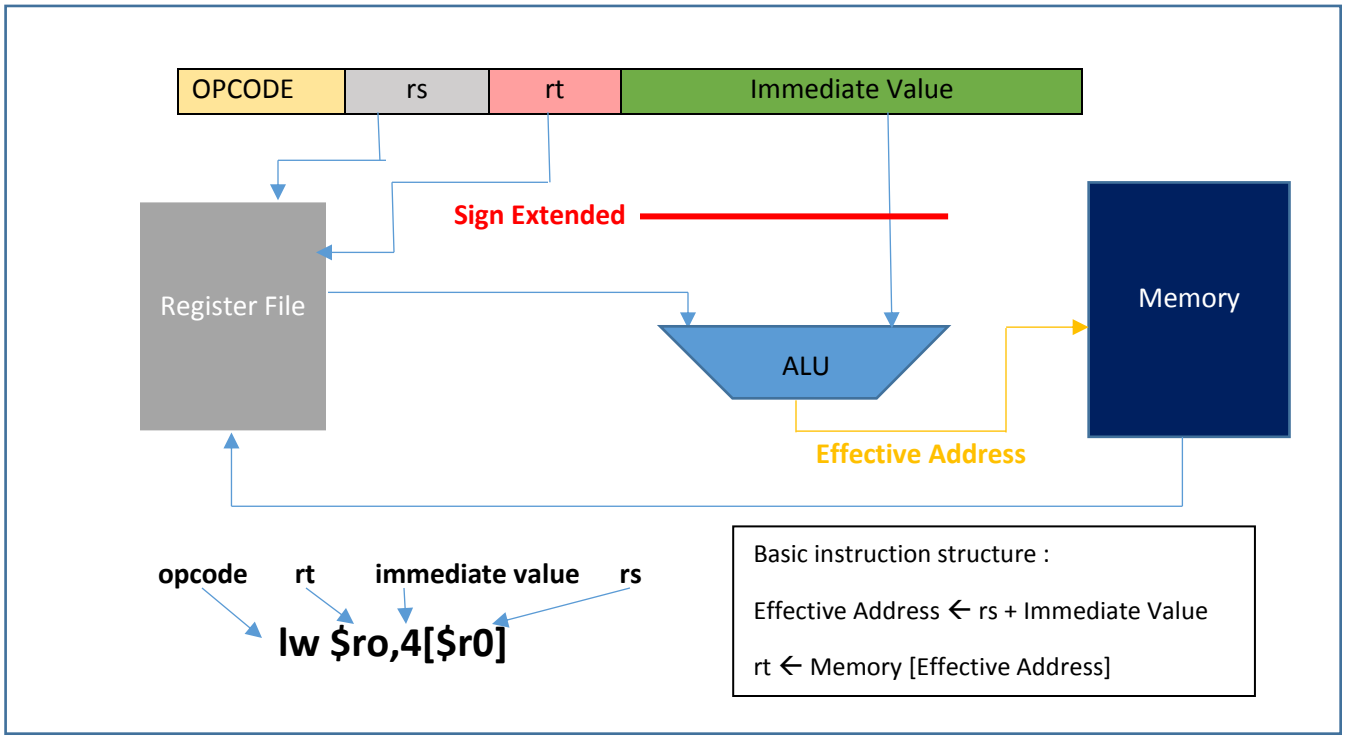
II.D.3 Register Indirect Addressing

This addressing mode is also known as register deferred. This mode is a mode in which a register is holding the address of the operand. This addressing mode is typically used to access operands pointed to by base pointers that are stored in registers.

The following operation and figure will be used to explain Register indirect addressing:

lw $ro,4[$r0] **In this instruction we are to load the memory contents located at the address specified by the value held in $r0 with an added offset of 4 (creating the effective address of where in memory we are going to address) and load the value at this effective memory address location into $r0**

Figure 5



In this figure we are adding the value held in the register specified in the rs field and the sign extended value of the immediate field and creating the effective address that we will use to address a location in memory. The contents inside this memory location specified by the effective address are then to be loaded into the register specified by the rt field

II.D.4 PC-Relative Addressing

This mode is used when a data or instruction memory location is specified as an offset relative to the incremented PC.

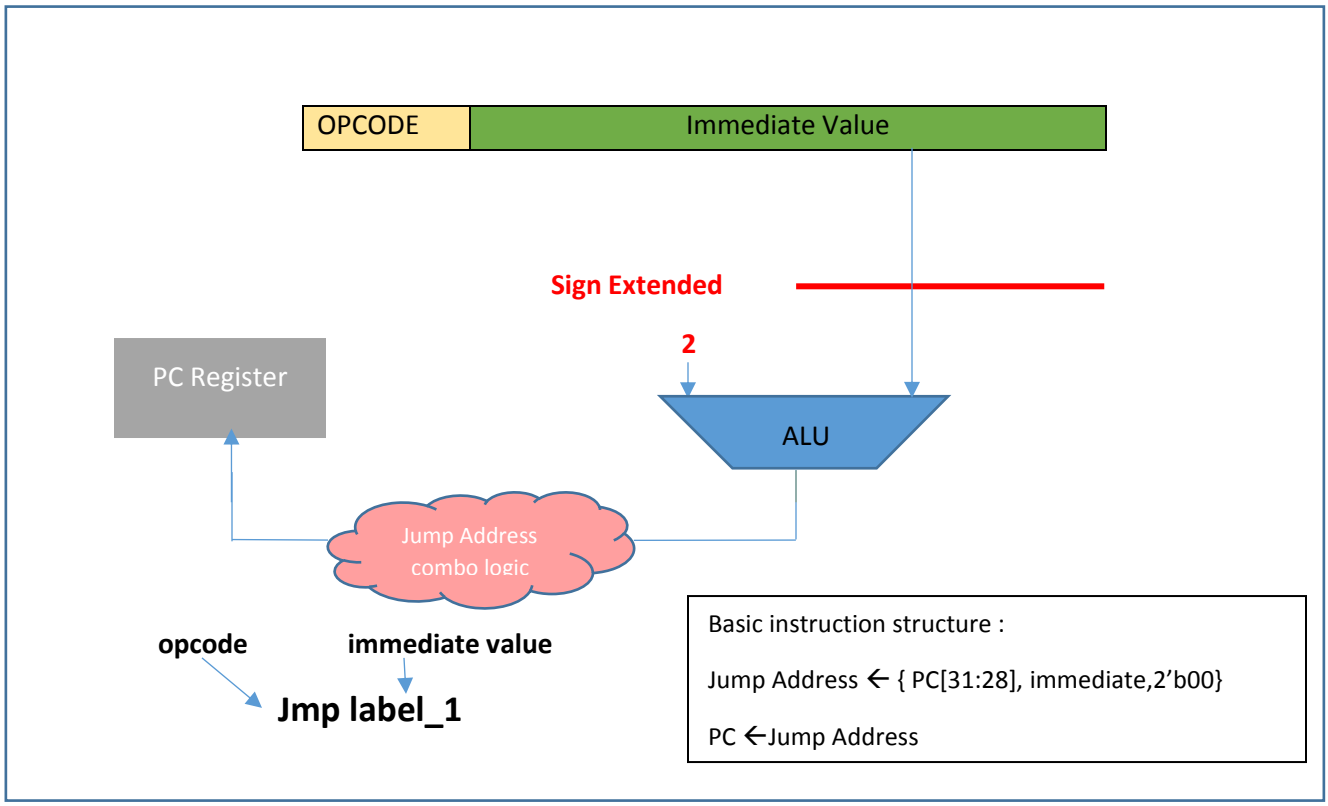
Typical uses of this mode are for branching or jumping from the current location in the program to a specified location in the instruction.

The following instructions and figures will be used to explain PC-Relative addressing:

Jmp label_1 **beq $r3,$r2, label_1**

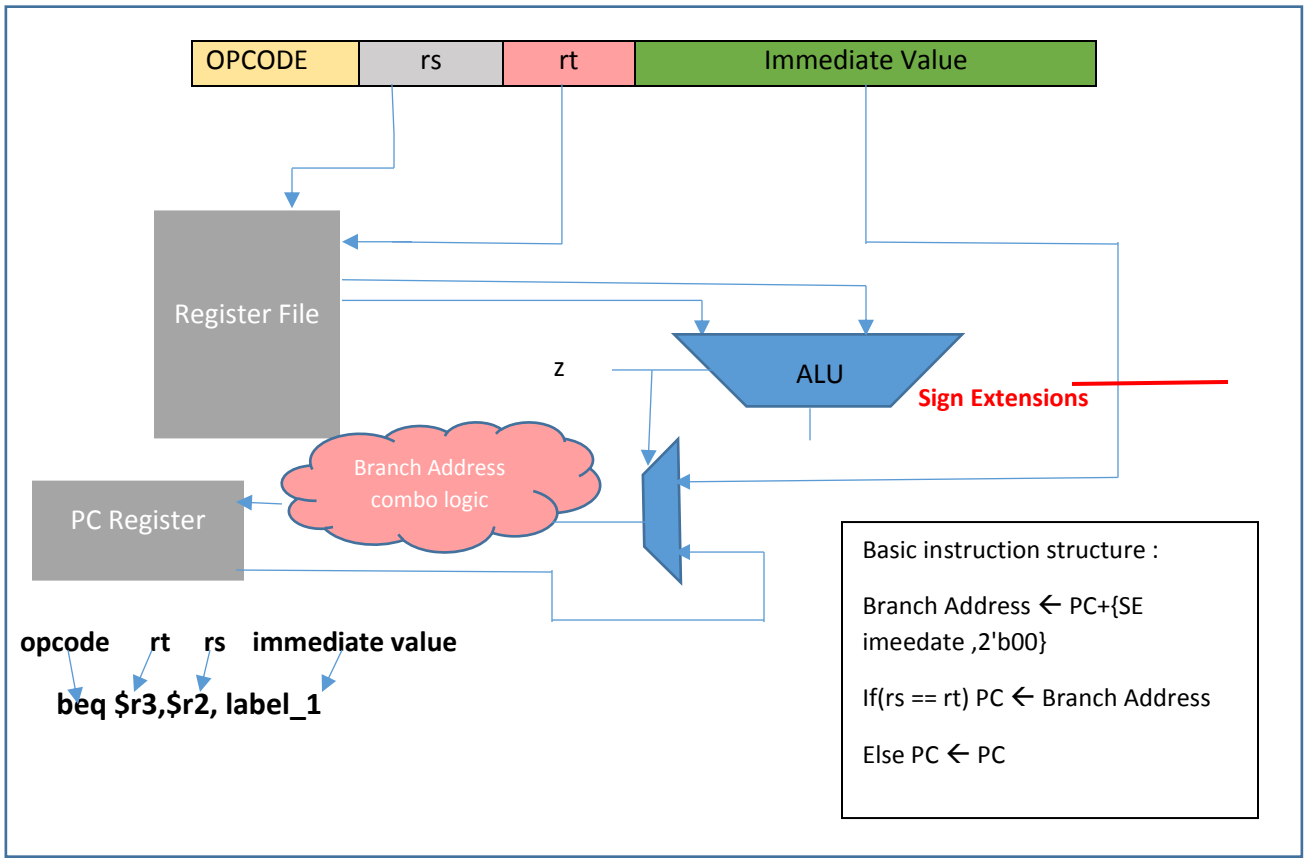
***Note : jump (jmp) instruction will be able to to jump to farther locations in the program because the immediate value that specifies where to jump to is 26 bits wide and the immediate value that specifies where to jump to in branches is only 16 bits wide.**

Figure 6



In this figure we are taking the immediate value and shifting its value to right twice and concatenating it with the 4 most significant bits of the program counter (now pointing to the next instruction) and loading the program counter with the effective jump address.

Figure 5

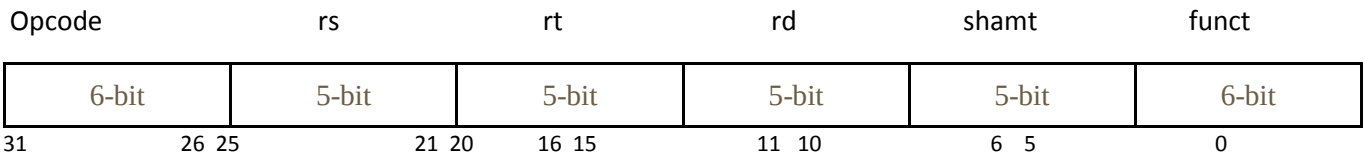


In this figure we are taking the immediate value and shifting its value to right twice and adding it to the program counter (now pointing to the next instruction) and loading the program counter with the effective branch address if rs and rt equal to each other (equality check is performed by checking the z flag , since both rs and rt are being subtracted to test for equality, if they are equivalent, their subtraction product will be a zero) or loading PC with itself.

II.E Instruction Set

II.E.1 R-Type Instructions

- All R-type instructions will use the following format:



Description of each bit field summarization is as follows:

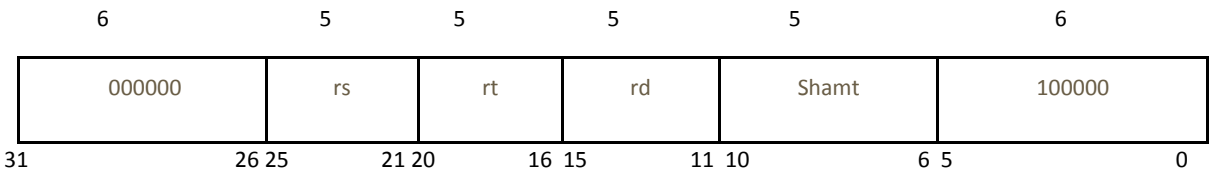
- **opcode** : 6 bit value that will always be 0x00 for R-type instructions. These are stored at bit locations 31 to 26.
- **rs** : 5 bit value that specifies the first source register address. These are stored at bit locations 25 to 21.
- **rt** : 5 bit value that specifies the second source register address. These are stored at bit locations 20 to 16.
- **rd** : 5 bit value that specifies the destination register address. These are stored at bit locations 15 to 11.
- **Shamt** : 5 bit value that specifies the amount of times to shift the register selected in the rt field. This fields value is only used in shift instructions. These are stored at bit locations 10 to 6.
- **funct** : 6 bit value that is the opcode for r-type instructions. This location specifies the instruction to be used. Location bits are bits 5 to 0.

Available R-Type Instructions

Name	funct[5:0]	**Name**	funct[5:0]
SLL	0x00	***SUB***	0x22
SRL	0x02	***SUBU***	0x23
SRA	0x03	***AND***	0x24
JR	0x08	***OR***	0x25
MFHI	0x10	***XOR***	0x26
MFLO	0x12	***NOR***	0x27
MULT	0x18	***SLT***	0x2A
DIV	0x1A	***SLTU***	0x2B
ADD	0x20	***BREAK***	0x0D
ADDU	0x21	***SETIE***	0x1F

II.E.1.a ADD

ADD	**Add signed**



Format | add rd, rs,rt, **New Horizon**

Purpose | To add contents of rs and rt and place the product into register rd.

Description | The contents of rs is added to the contents of rt and placed into rd.If rs is less than rt ,carry will occur .If rs and rt are both positive values or negative values , overflow can occur.The end result of this addition will be placed into the register specified in rd.

Since this is not a shifting operation, values in shamt field will go unused.

Operation | temp ← rs + rt
rd ← temp

Restrictions | Valid values are from -2,147,483,648 to 2,147,483,648 since this is a signed data type operation.

Exception | None.

Programmer's Notes | add rd,rs,$zero : Can use to swap contents of rd with rs.

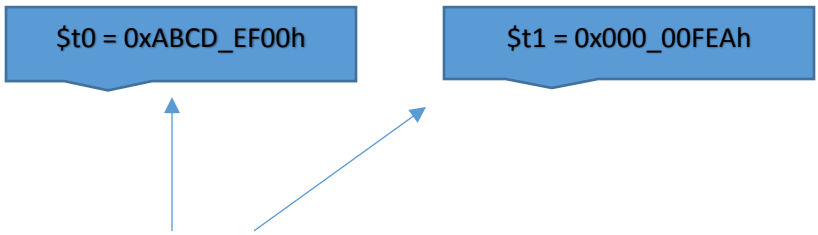
N,V,C,Z Flags | Carry can occur in situations where rs is less than rt. Overflow can occur when rs and rt both hold positive values or both hold negative values.

II.E.1.a.1 Example

Add Signed Instruction

Instruction Description: This instruction is an R-type instruction where temporary registers $t0 & $t1, being 32 bits each, are added together with their appropriate contents. The results are then stored back in temporary register $t0.

***Initial contents:**



add $t0, $t0, $t1 #$t0 = $t0 + $t1

$t0 = 0xABCD_EF00h

$t1 = 0x000_00FEAh

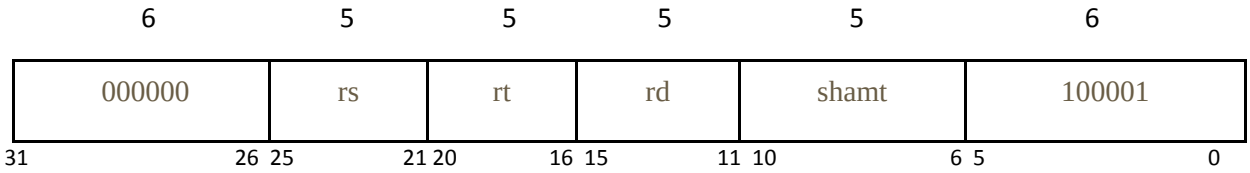
***Contents after Operation Results:**

Flags affected after operation(if any)

- **Carry(C) : 1'b0**
- **Overflow(V): 1'b0**
- **Negative(N): 1'b1**
- **Zero(Z) : 1'b0**

II.E.1.b ADDU

ADDU	**Add unsigned**



Format | addu rd, rs,rt, **New Horizon**

Purpose | To add contents of rs and rt and place the product into register rd.

Description | The contents of rs is added to the contents of rt and placed into rd.If rs and rt produce a number greater than $2^{31} - 1$, carry and overflow will occur .

Operation | temp ← rs + rt
rd ← temp

Restrictions | Valid values are from 0 to $2^{32} - 1$ since this is a unsigned data type operation.

Exception | None.

Programmer's Notes | addu rd,rs,$zero : Can use to swap contents of rd with rs.

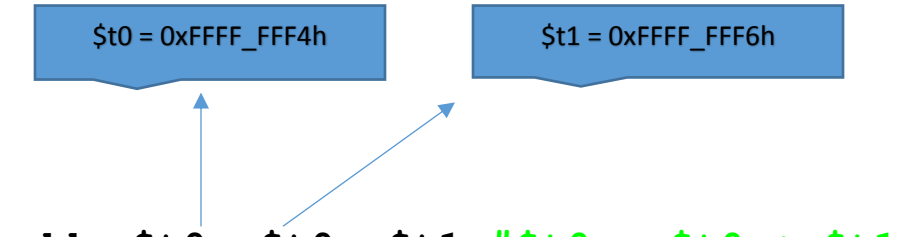
N,V,C,Z Flags | Carry can occur in situations where the addition of rs and rt produce a number greater than $2^{31} - 1$.

II.E.1.b.1 Example

Add Unsigned Instruction #1

Instruction Description: This instruction is an add unsigned R-type instruction where temporary registers $t0 & $t1, being 32 bits each, are added together with their appropriate contents. The results are then stored back in temporary register $t0.

***Initial contents:**

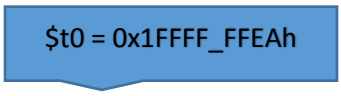



```
addu $t0, $t0, $t1 #$t0 = $t0 + $t1
```

$t0 = 0xFFFF_FFF4h

$t1 = 0xFFFF_FFF6h

***Contents after Operation Results:**

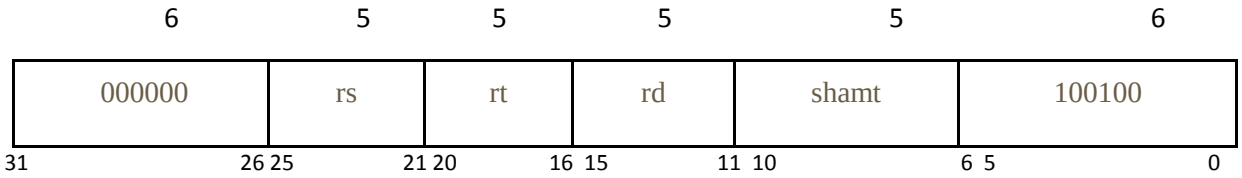


Flags affected after operation(if any)

- Carry(C) : 1'b1
- Overflow(V): 1'b1
- Negative(N): 1'b0
- Zero(Z) : 1'b0

II.E.1.c AND

AND	And



Format | and rd, rs,rt, **New Horizon**

Purpose | To perform a bitwise logical AND

Description | To bitwise AND the contents of rs with the contents of rt and place the product into rd.

Operation | temp ← rs AND rt
rd ← temp

Restrictions | None.

Exception | None.

Programmer's Notes | and rd,rs,$zero : Can use to clear the contents of rd.

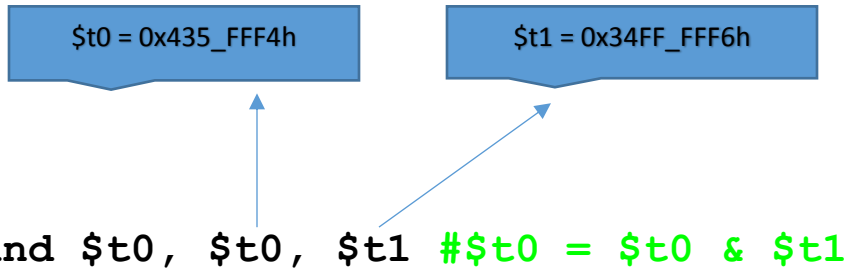
N,V,C,Z Flags | Only zero flag and negative flag are affected.

II.E.1.c.1 Example

AND Instruction #1

Instruction Description: This instruction is an and R-type instruction where temporary registers $t0 & $t1, being 32 bits each, are added together with their appropriate contents. The results are then stored back in temporary register $t0.

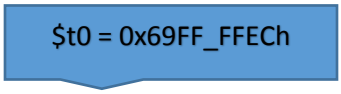
***Initial register contents:**



$t0 = 0x435_FFF4h

$t1 = 0x34FF_FFF6h

***Register contents after operation:**

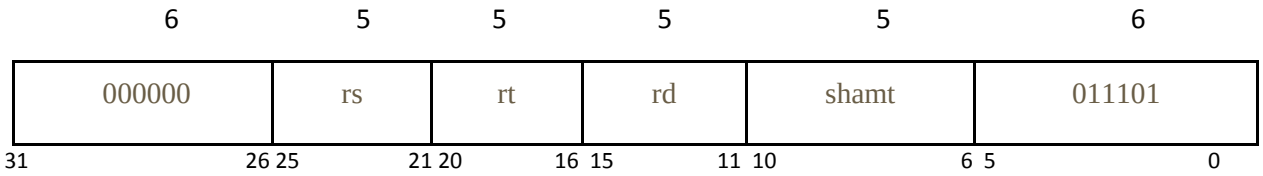


Flags affected after operation(if any)

- **Carry(C) : 1'b0**
- **Overflow(V): 1'b0**
- **Negative(N): 1'b0**
- **Zero(Z) : 1'b0**

II.E.1.d BREAK

BREAK	**Break**



Format | Break **New Horizon**

Purpose | To stop execution of instructions.

Description | Break instruction halts the program from fetching any further instructions.

Operation | To end the program.

Restrictions | None.

Exception | None.

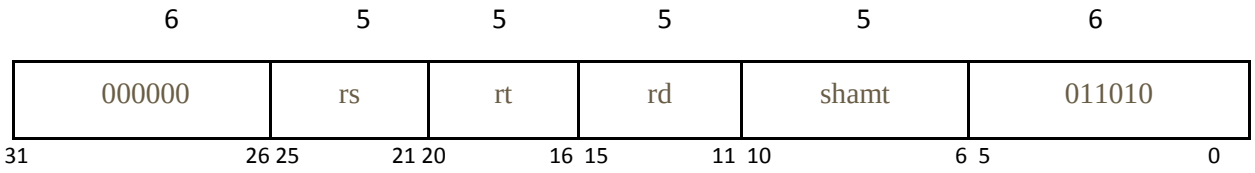
Programmer's Notes | Don’t use to end loops, because as soon as the break instruction gets executed the entire program will stop and no further iterations of the loop will occur.

Use to terminate program.

N,V,C,Z Flags | None.

II.E.1.e DIV

DIV	**Division**



Format | div rs,rt **New Horizon**

Purpose | To divide the values of rs and rt

Description | {LO,HI}← rs / rt

The 32 bit values of rs and rt are divided to produce two 32 bit values, a 32 bit quotient and a 32 bit remainder. The 32 bit quotientis placed into register LO and the 32 remainder is placed into register HI.

Operation | quotient ← rs / rt
remainder← rs%rt
LO← quotient
HI← remainder

Restrictions | None.

Exception | None.

Programmer's Notes | None.

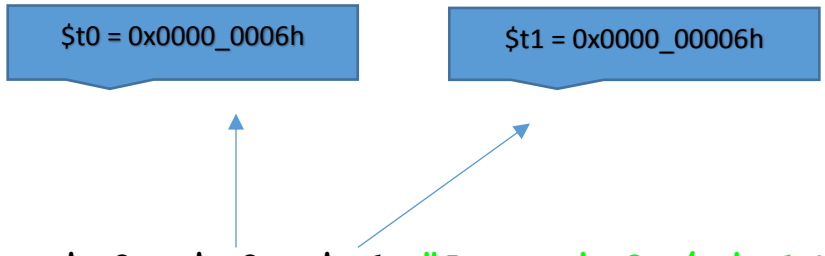
N,V,C,Z Flags | Negative and zero flags . Carry can never occur.

II.E.1.e.1 Example

DIV Instruction #1

Instruction Description: This instruction is an and R-type instruction where temporary registers $t0 & $t1, being 32 bits each, are divided together with their appropriate contents. The results are then stored back in temporary 32 bit registers HI and LO.

***Initial register contents:**




```
DIV $t0, $t0, $t1 #lo = $t0 / $t1(Quotient Only)
                  #hi = $t0 % $t1
```

$t0 = 0x0000_0002h

$t1 = 0x34FF_FFF6h

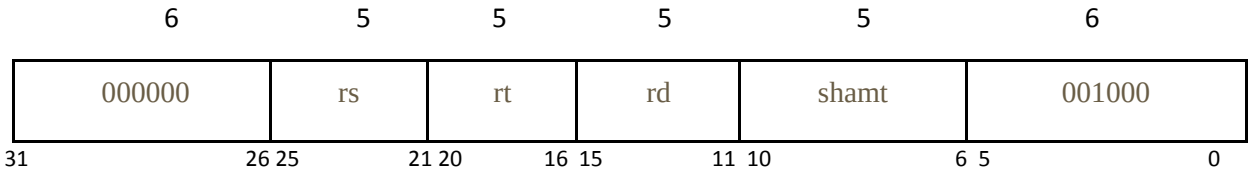
***Register contents HI and LO after operation:**

Flags affected after operation(if any)

- **Carry(C) : 1'b0**
- **Overflow(V): 1'b0**
- **Negative(N): 1'b0**
- **Zero(Z) : 1'b0**

II.E.1.f JR

JR	**Jump Register**



Format | jr rs

New Horizon

Purpose | To execute a branch to a location address held in register rs

Description | Jump to the effective address held in register rs

Operation | PC <----[RS]

Restrictions | The effective address must obey the word alignment rule

Exception | None.

Programmer's Notes | None.

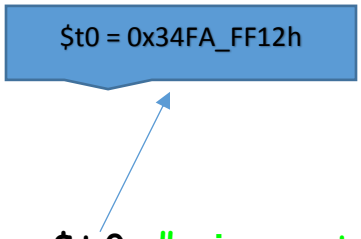
N,V,C,Z Flags | No flags are affected.

II.E.1.f.1 Example

Jump Register (jr) Instruction #1

Instruction Description: This instruction is an R-type instruction called jump register that executes a branch to a location address held in register $t0, also known as the effective address.

- **Initial register contents:**

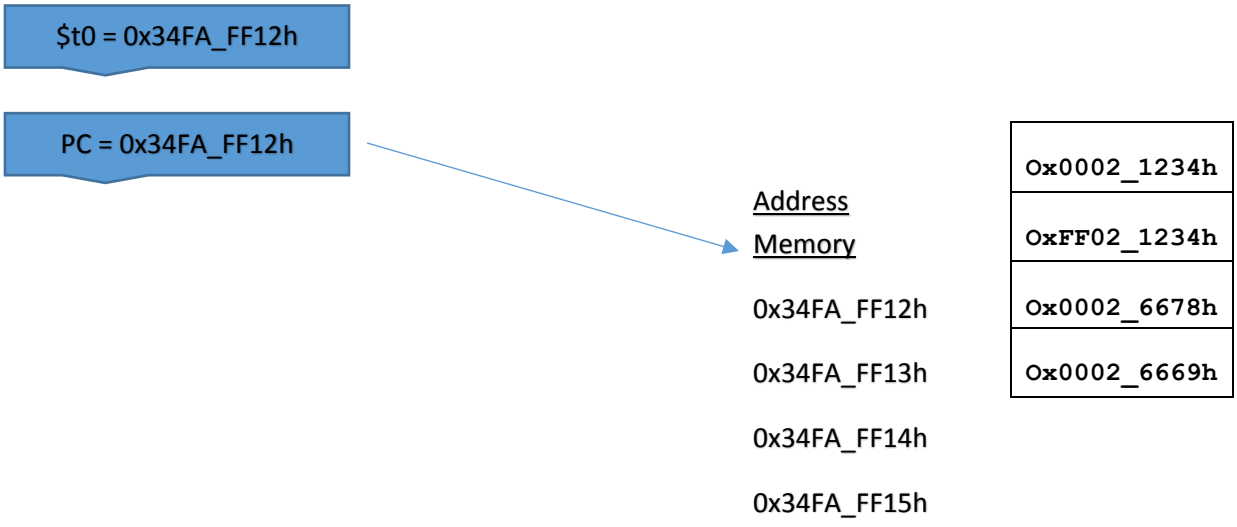



```
jr $t0 # jump to address stored in $t0
```

$t0 = 0x34FA_FF12h

PC = 0x5522_0000h #arbitrary value, not updated

***Register contents after operation:**

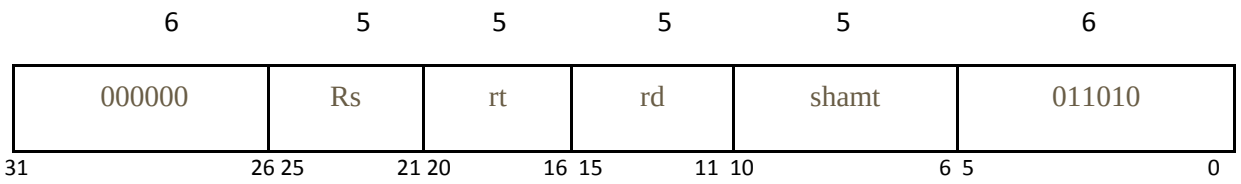


Flags affected after operation(if any)

- **Carry(C) : 1'b0**
- **Overflow(V): 1'b0**
- **Negative(N): 1'b0**
- **Zero(Z) : 1'b0**

II.E.1.g MFHI

MFHI	Move From HI



Format | mfhi rd **New Horizon**

Purpose | To copy the contents of HI register into a register specified by rd

Description | rd← HI
The 32 bit value of HI is placed into register specified by rd

Operation | rd← HI

Restrictions | None.

Exception | None.

Programmer's Notes | None.

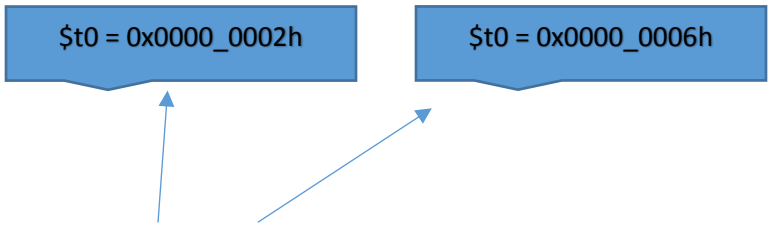
N,V,C,Z Flags | None.

II.E.1.g.1 Example

Move From Hi (mfhi) Instruction #1

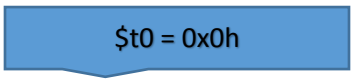
Instruction Description: This instruction is an R-type instruction called move from hi. Example shows a modulus division operation and has the remainder .

❖ **Initial register contents:**




```
div $t0, $t0, $t1,     # Hi = $t0 % $t1
mfhi $t0               # $rd = Hi
$t0 = 0x22FA_FA11h     #arbitrary value, not updated
```

***Register contents after operation:**

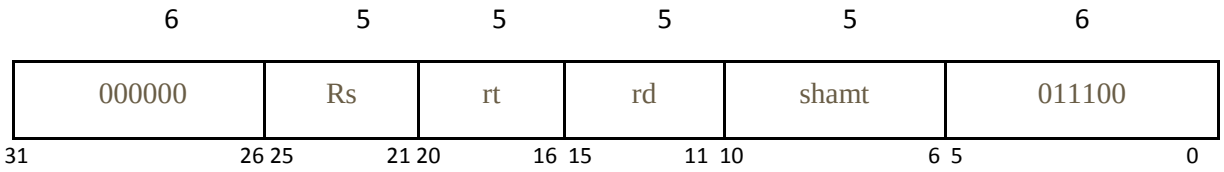


Flags affected after operation(if any)

- **Carry(C) : 1'b0**
- **Overflow(V): 1'b0**
- **Negative(N): 1'b0**
- **Zero(Z) : 1'b0**

II.E.1.h MFLO

MFLO	**Move From LO**



Format | mflo rd **New Horizon**

Purpose | To copy the contents of LO register into a register specified by rd

Description | Rd← LO

The 32 bit value of LO is placed into register specified by rd

Operation | Rd←LO

Restrictions | None.

Exception | None.

Programmer's Notes | None.

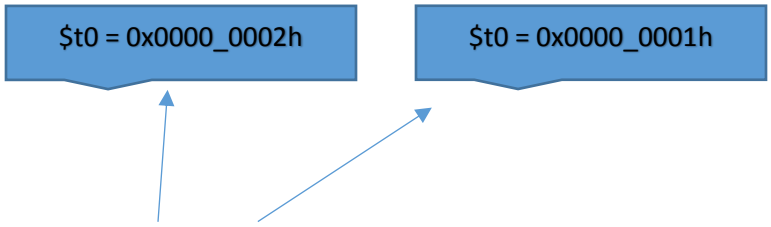
N,V,C,Z Flags | None.

II.E.1.h.1 Example

Move From Lo (mflo) Instruction #1

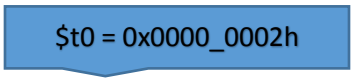
Instruction Description: This instruction is an R-type instruction called move from lo. Example shows a multiplication operation and has the resultant copy the contents of LO register into the specified destination register.

❖ **Initial register contents:**




```
mult $t0, $t0, $t1,    # LO = $t0 * $t1
mfhi $t0               # $rd = LO
$t0 = 0x22FA_FA11h     #arbitrary value, not updated
```

***Register contents after operation:**

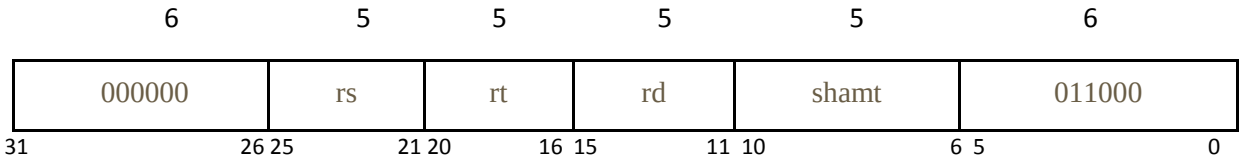


Flags affected after operation(if any)

- **Carry(C) : 1'b0**
- **Overflow(V): 1'b0**
- **Negative(N): 1'b0**
- **Zero(Z) : 1'b0**

II.E.1.i MUL

MUL	Multiply



Format | mul rs,rt **New Horizon**

Purpose | To multiply the values of rs and rt

Description | {LO,HI}<----- rs * rt
The 32 bit values of rs and rt are multiplied to produce a 64 bit value. This 64 bit value has its upper 32 bits placed into register HI and the lower order 32 bits into register LO.

Operation | Product ← rs * rt
LO ← Product[32:16]
HI ← Product[15:0]

Restrictions | None.

Exception | None.

Programmer's Notes | None.

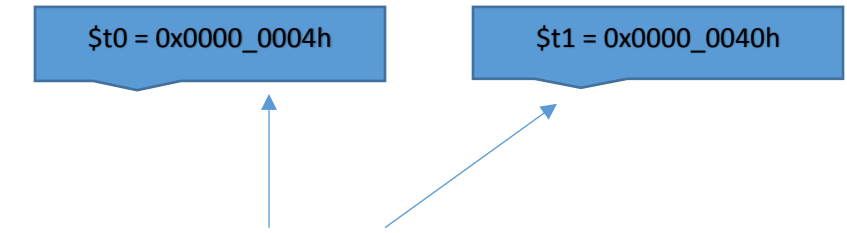
N,V,C,Z Flags | Negative flag and Zero flag are effected. Can never overflow or carry.

II.E.1.i.1 Example

Multiply(mult) Signed Instruction #1

Instruction Description: This instruction is an R-type instruction where temporary registers $t0 & $t1 are multiplied and destination register (in our example $t0) receives the resultant value.

***Initial contents:**

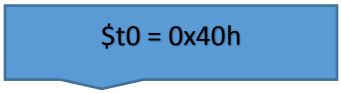



```
mult $t0, $t0, $t1 #$t0 = $t0 * $t1
```

$t0 = 0xABCD_EF00h

$t1 = 0x000_00FEAh

***Contents after Operation Results:**

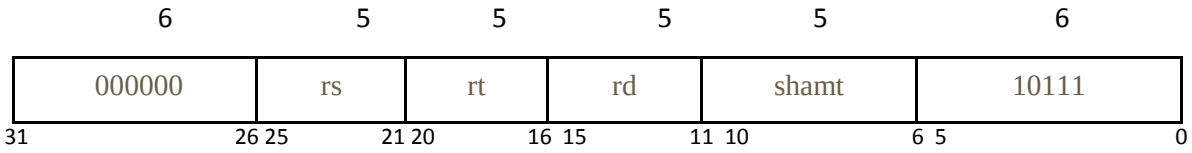


Flags affected after operation(if any)

- **Carry(C) : 1'b0**
- **Overflow(V): 1'b0**
- **Negative(N): 1'b1**
- **Zero(Z) : 1'b0**

II.E.1.j NOR

NOR	NOR



Format | nor rd, rs, rt **New Horizon**

Purpose | To perform a bitwise logical NOR

Description | To bitwise NOR the contents of rs with the contents of rt and place the product into rd.

Operation | temp ←rs NOR rt
rd ← temp

Restrictions | None

Exception | None.

Programmer's Notes | None.

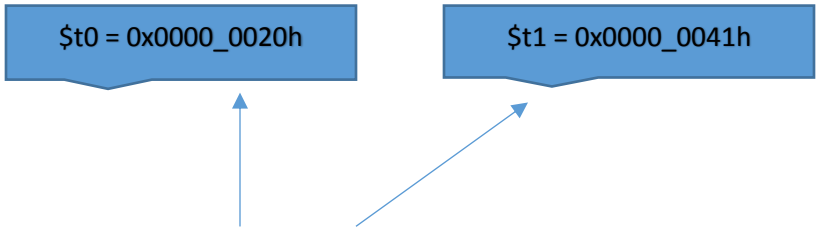
N,V,C,Z Flags | Only negative and zero flags effected.

II.E.1.j.1 Example

NOR(nor) Signed Instruction #1

Instruction Description: This instruction is an R-type instruction where temporary registers $t0 & $t1 are bitwise Nor'ed together. The destination register (in our example $t0) receives the resultant value.

***Initial contents:**

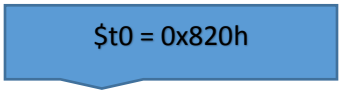



```
nor $t0, $t0, $t1 #$t0 = $t0 nor $t1
```

$t0 = 0x0000_0020h

$t1 = 0x0000_0041h

***Contents after Operation Results:**

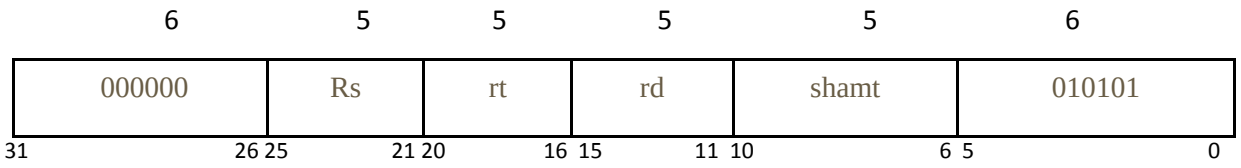


Flags affected after operation(if any)

- **Carry(C) : 1'b0**
- **Overflow(V): 1'b0**
- **Negative(N): 1'b0**
- **Zero(Z) : 1'b0**

II.E.1.k OR

OR	OR



Format | or rd, rs, rt **New Horizon**

Purpose | To perform a bitwise logical OR

Description | To bitwise OR the contents of rs with the contents of rt and place the product into rd.

Operation | temp ← rs OR rt
rd ← temp

Restrictions | None

Exception | None.

Programmer's Notes | None.

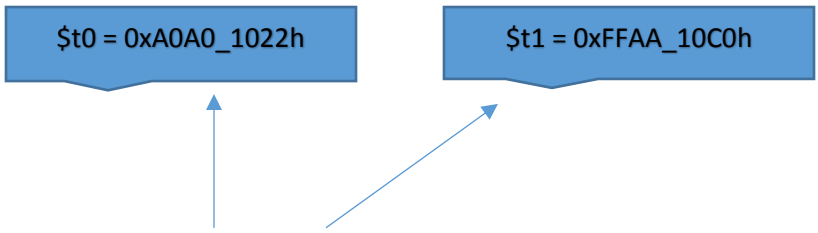
N,V,C,Z Flags | Negative flags and zero flags effected.

II.E.1.k.1 Example

OR(or) Signed Instruction #1

Instruction Description: This instruction is an R-type instruction where temporary registers $t0 & $t1 are bitwise OR'ed together. The destination register (in our example $t0) receives the resultant value.

***Initial contents:**




```
or $t0, $t0, $t1 #$t0 = $t0 or $t1
```

$t0 = 0xA0A0_1022h

$t1 = 0xFFAA_10C00h

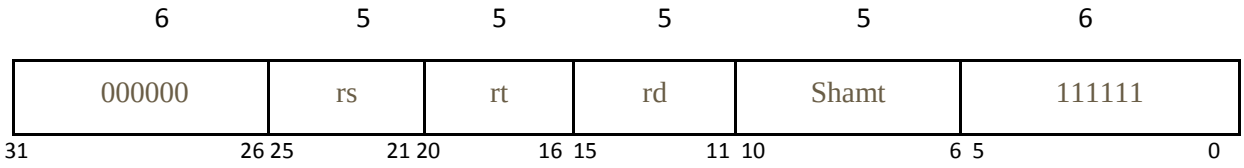
***Contents after Operation Results:**

Flags affected after operation(if any)

- **Carry(C) : 1'b0**
- **Overflow(V): 1'b0**
- **Negative(N): 1'b0**
- **Zero(Z) : 1'b0**

II.E.1.I SETIE

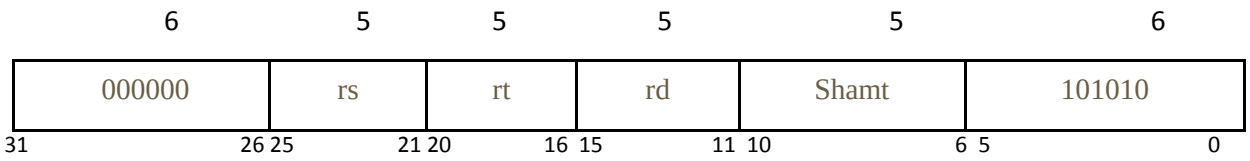
SETIE	Set Interrupt Enable



Format	Setie	**New Horizon**
Purpose	To allow interrupts to be acknowledged and received by the processor.	
. Description	Allows the program to receive interrupts during its execution of a program.	
Operation	Allows interrupts to occur.	
Restrictions	Once interrupts are allowed to occur, they cannot be changed later on to be ignored.	
Exception	None.	
Programmer's Notes	None.	
N,V,C,Z Flags	None.	

II.E.1.m SLT

SLT — Shift Less Than signed



Format | slt rd,rs,rt **New Horizon**

Purpose | To determine if rs is less than rt

Description | Rd←(rs < rt)
rt and rs hold signed integers.If rs is LESS THAN rt, then the comparison is true (1), if rs IS NOT LESS THAN rt, then the comparison is false (0).

Operation | if(rs<rt)
rd← 1
else
rd← 0

Restrictions | Valid values are from -2,147,483,648 to 2,147,483,648 since this is a signed data type operation.

Exception | None.

Programmer's Notes | None.

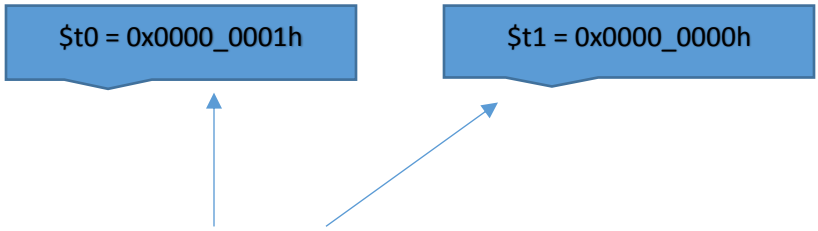
N,V,C,Z Flags | Negative and Zero flag are effected.

II.E.1.m.1 Example

Set Less Than Signed(slt) Instruction #1

Instruction Description: This instruction is an R-type instruction to determine if $t0 is less than $t1. The destination register (in our example $t0) receives either a '1' bit when $t0 is less than $t1, else destination register gets a '0' bit.

***Initial contents:**

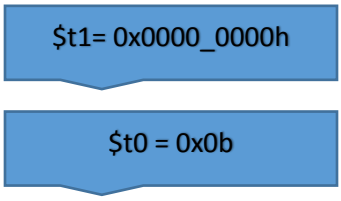



```
Slt $t0, $t0, $t1 #$t0 = $t0 < $t1
                  #if( $t0 < $t1)
                  #    $t0 = 1
                  #else
                  #    $t1 = 0
```

$t0 = 0x0000_0001h

$t1 = 0x0000_0000h

***Contents after Operation Results:**

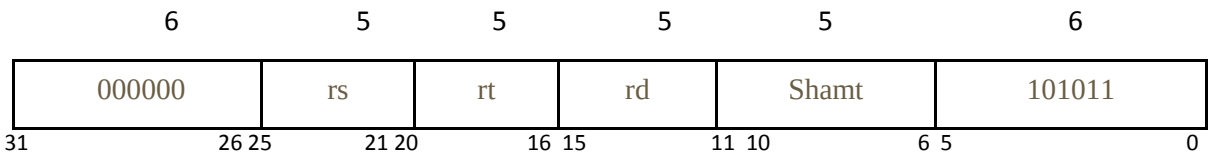


Flags affected after operation(if any)

- **Carry(C) : 1'b0**
- **Overflow(V): 1'b0**
- **Negative(N): 1'b0**
- **Zero(Z) : 1'b0**

II.E.1.n SLTU

SLTU	Shift Less Than Unsigned



Format | sltu rd,rs,rt **New Horizon**

Purpose | To determine if rs is less than rt

Description | rd<---(rs<rt)
rt and rs hold unsigned values.If rs is LESS THAN rt, then the comparison is true (1), if rs IS NOT LESS THAN rt, then the comparison is false (0).

Operation | if(rs<rt)
rd← 1
else
rd← 0

Restrictions | Valid values are from 0 to $2^{32} - 1$ since this is a unsigned data type operation.

Exception | None.

Programmer's Notes | None.

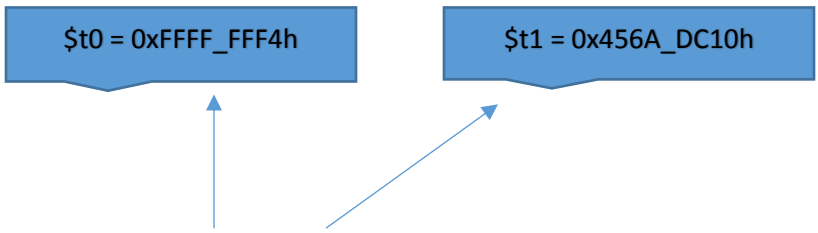
N,V,C,Z Flags | Negative and zero flag are effeted

II.E.1.n.1 Example

Set Less Than Unsigned(sltu) Instruction #1

Instruction Description: This instruction is an R-type instruction called set less than with unsigned values. The instruction determines if $t0 is less than $t1. The destination register (in our example $t0) receives either a '1' bit when $t0 is less than $t1, else destination register gets a '0' bit.

***Initial contents:**

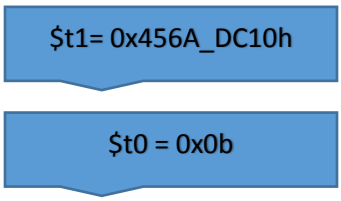



```
Sltu $t0, $t0, $t1 #$t0 = $t0 < $t1
                   #if( $t0 < $t1)
                   #    $t0 = 1
                   #else
                   #    $t1 = 0
```

$t0 = 0xFFFF_FFF4h

$t1 = 0x456A_DC10h

***Contents after Operation Results:**

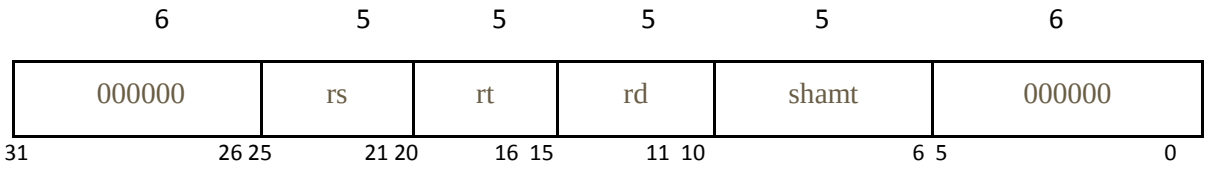


Flags affected after operation(if any)

- **Carry(C) : 1'b0**
- **Overflow(V): 1'b0**
- **Negative(N): 1'b0**
- **Zero(Z) : 1'b0**

II.E.1.o SLL

SLL	Shift Left Logical



Format | sll rd, rt, shamt

New Horizon

Purpose | To shift contents in rt to the left by the amount specified in shamt.

Description | The contents of rt is shifted to the left by inserting zeros into the shifted bit locations and moving the contents of rt to the left by the amount specified in shamt

Operation | temp ← rt << shamt
rd ← temp

Restrictions | Can only shift contents of the source register (rt) a maximum of 31 times in one sll call.

Exception | None.

Programmer's Notes | sll rd,rt,0 : Essentially swapping the contents of rd with rt.

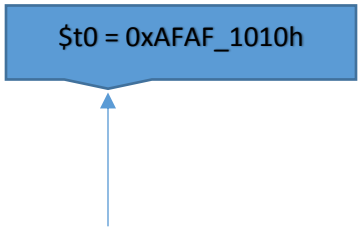
N,V,C,Z Flags | A carry occurs in situations where a shift left on rt pushes out a binary value of one

II.E.1.o.1 Example

Shift Left Logical(sll) Instruction #1

Instruction Description: This instruction is an R-type instruction called shift left logical. The instruction determines if $t0 is less than $t1. The destination register (in our example $t0) receives either a '1' bit when $t0 is less than $t1, else destination register gets a '0' bit.

***Initial contents:**

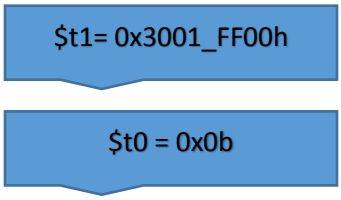



```
Slt $t0, $t0, 4 #$t0 = $t0 < $t1
                #if( $t0 < $t1)
                #    $t0 = 1
                #else
                #    $t0 = 0
```

$t0 = 0xAFAF_1010h

$t1 = 0x3001_FF00h

***Contents after Operation Results:**



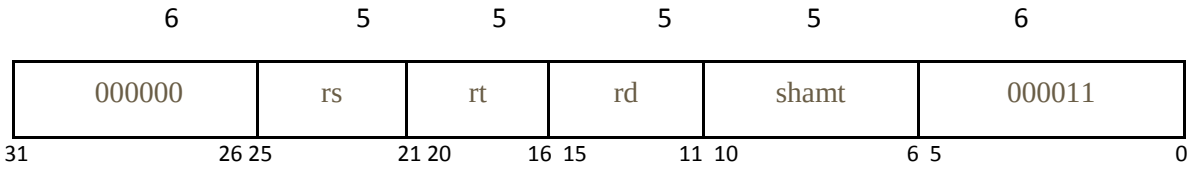
#$t0 was less than $t1, so it receives a '1'bit

Flags affected after operation(if any)

- Carry(C) : 1'b0
- Overflow(V): 1'b0
- Negative(N): 1'b0
- Zero(Z) : 1'b0

II.E.1.p SRA

SRA — Shift Right Arithmetic



Format | sra rd, rt, shamt **New Horizon**

Purpose | To shift the contents of rt to the right by the amount of times specified by shamt field.

Description | The contents of rt is shifted to the left by inserting the bit value of the msb of rs (before any shifting) into the shifted bit locations and moving the contents of rt to the right by the amount specified in shamt

Operation | temp ← rt >>> shamt
rd ← temp

Restrictions | None.

Exception | None.

Programmer's Notes | Retains the signed value of the data being shifted to the right.

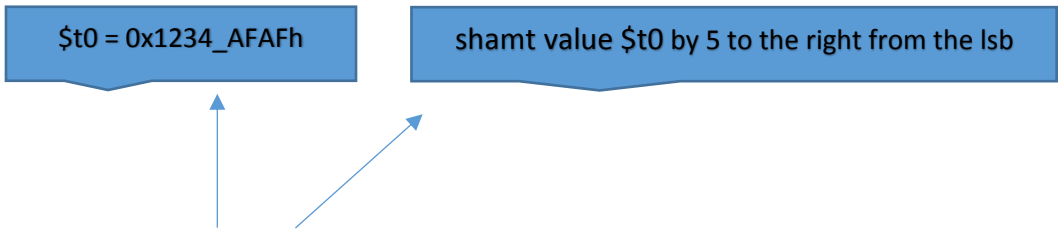
N,V,C,Z Flags | All flags are effect

II.E.1.p.1 Example

Shift right arithmetic(sra) Instruction #1

Instruction Description: This instruction is an R-type instruction called shift right arithmetic. The $t0 register is shifted by the amount specified by the shamt operand. The destination register (in our example $t0) then receives a new value which is the shifted value .

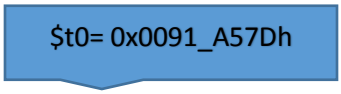
***Initial contents:**




```
Sra $t0, $t0, 5 #$t0 = $t0 → shamt 5
```

$t0 = 0x1234_AFAFh

***Contents after Operation Results:**

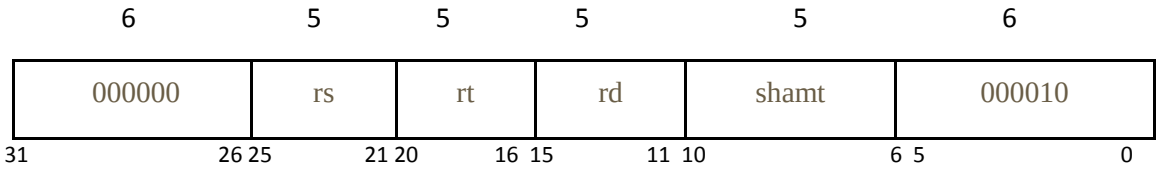


Flags affected after operation(if any)

- Carry(C) : 1'b0
- Overflow(V): 1'b0
- Negative(N): 1'b0
- Zero(Z) : 1'b0

II.E.1.q SRL

SRL — Shift Right Logical



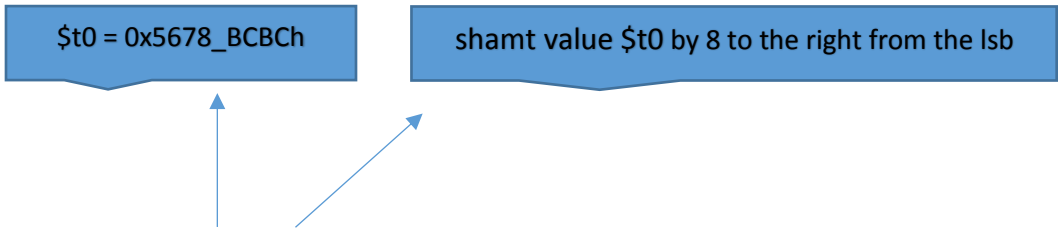
Format	srl rd, rt, shamt
Purpose	To shift the contents in rt to the right by the amount specified in shamt.
Description	The contents of rt is shifted to the right by inserting zeros into the shifted bit locations and moving the contents of rt to the right by the amount specified in shamt
Operation	temp ← rt >> shamt rd ← - temp
Restrictions	Can only shift contents of the source register (rt) a maximum of 31 times in one srl call.
Exception	None.
Programmer's Notes	srl rd,rt,0 : Essentially swapping the contents of rd with rt.
N,V,C,Z Flags	A carry occurs in situations where a shift right on rt pushes out a binary value of one

II.E.1.q.1 Example

Shift right logical (srl) Instruction #1

Instruction Description: This instruction is an R-type instruction called shift right logical. The $t0 register is shifted by the amount specified by the shamt operand. The destination register (in our example $t0) then receives a new value which is the shifted value.

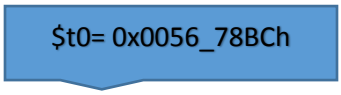
***Initial contents:**




```
Sra $t0, $t0, 8 #$t0 = $t0 → shamt 5
```

$t0 = 0x5678_BCBCh

***Contents after Operation Results:**

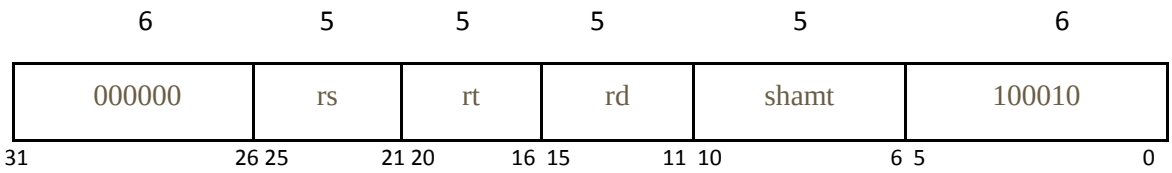


Flags affected after operation (if any)

- **Carry(C) : 1'b0**
- **Overflow(V): 1'b0**
- **Negative(N): 1'b0**
- **Zero(Z) : 1'b0**

II.E.1.r SUB

Sub	**Subtract signed**



Format | sub rd, rs,rt, **New Horizon**

Purpose | To subtract contents of rs and rt and place the product into register rd.

Description | The contents of rt is subtracted from the contents of rs and placed into rd. If the contents of rt is greater than the contents of rs, overflow and carry may occur.The end result of this subtraction will be placed into register specified in rd.

Operation | temp ← rs - rt
rd ← temp

Restrictions | Registers can only hold a value of : $-2^{32}\ to\ 2^{32}\ -1$

Exception | None.

Programmer's Notes | sub rd,rs,rt : Can use to check for equality , making the contents of rd equal to zero.
sub rd,rs,$zero : Can use to swap contents of rd with rs.
sub rd,$zero,rt : Can be used to 2's complement rt and place that value into rd.

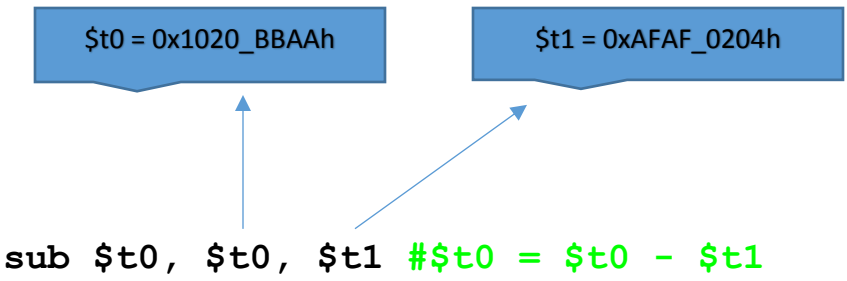
N,V,C,Z Flags | A carry occurs in situations where rs is less than rt. Overflow can occur in situations where rs and rt are both positive or both negative.

II.E.1.r.1 Example

Subtract(sub) Signed Instruction #1

Instruction Description: This instruction is an R-type instruction where temporary registers $t0 & $t1 are bitwise OR'ed together. The destination register (in our example $t0) receives the resultant value. Since the final result stored in register $t0 is a negative value, the negative flag gets asserted and the carry, overflow, and zero flags are deasserted.

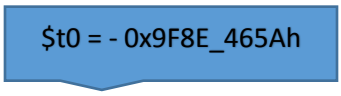
***Initial contents:**



$t0 = 0x1020_BBAAh

$t1 = 0xAFAF_0204h

***Contents after Operation Results:**

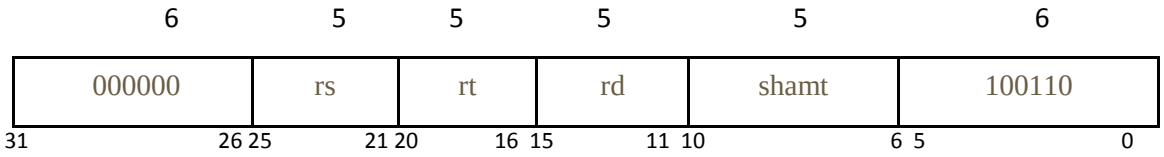


Flags affected after operation(if any)

- **Carry(C) : 1'b0**
- **Overflow(V): 1'b0**
- **Negative(N): 1'b1**
- **Zero(Z) : 1'b0**

II.E.1.s XOR

XOR	XOR



Format | xor rd, rs, rt **New Horizon**

Purpose | To perform a bitwise logical XOR

Description | To bitwise XOR the contents of rs with the contents of rt and place the product into rd.

Operation | temp ← rs XOR rt
rd ← temp

Restrictions | None

Exception | None.

Programmer's Notes | None.

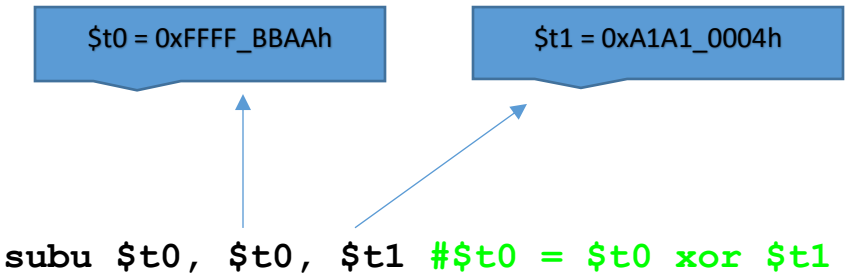
N,V,C,Z Flags | Negative and Zero flags are effected.

II.E.1.s.1 Example

XOR(xor) Instruction #1

Instruction Description: This instruction is an xor R-type instruction where temporary registers $t0 & $t1 contents are bit wise xor'ed. The destination register (in our example $t0) receives the resultant value.

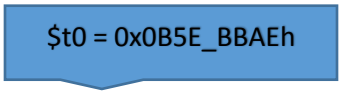
***Initial contents:**



$t0 = 0xAAFF_BBAAh

$t1 = 0xA1A1_0004h

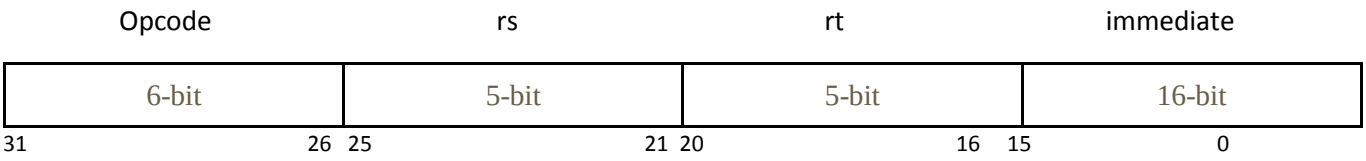
***Contents after Operation Results:**



Flags affected after operation(if any)

- Carry(C) : 1'b0
- Overflow(V): 1'b0
- Negative(N): 1'b0
- Zero(Z) : 1'b0

II.E.2 I-Type Instructions

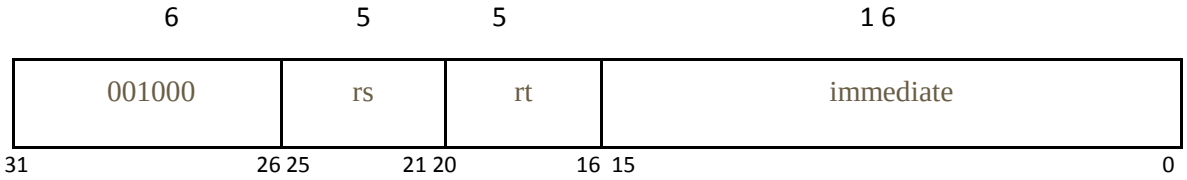


Description of each bit field summarization is as follows:

- **Opcode** : 6 bit value that specifies the I type instruction to be performed. These are stored at bit locations 31 to 26.
- **rs** : 5 bit value that specifies the source register address. These are stored at bit locations 25 to 21.
- **rt** : 5 bit value that specifies the destination register address. These are stored at bit locations 20 to 16.
- Immediate : 16 bit value that is used for both arithmetic and logic operations. Used as a signed offset in load/store

II.E.2.a ADDI

ADDI — Add Immediate signed



Format | addi rt, rs, immediate **New Horizon**

Purpose | To add a 32 bit value held in rs with a 16 bit immediate signed value and placed into register rt.

Description | The 16 bit immediate value is added to the contents of rs. The immediate value is sign extended to 32 bit length before being added to rs through the following operation : {16{immediate [15],immediate}.

Operation | temp ← rs + {16{immediate[15], immediate}
rt ← temp

Restrictions | Immediate value can only hold a value of : $-2^{15} to\ 2^{15}$- 1

Exception | None.

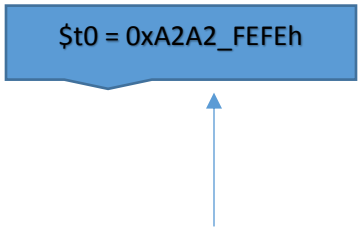
Programmer's Notes | addi rt,rs,0 : Can use to swap contents of rt with rs.

Since addi is a signed operation, choosing for the immediate value to be negative is equivalent to having a subi rt,rs,immediate pseudo-instruction.

II.E.2.a.1 Example

Addi signed(addi) Instruction #1

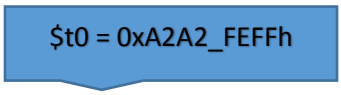
***Initial contents:**




```
addi $t0, $t0, 0x1 #$t0 = $t0 + $t1
```

$t0 = 0xA2A2_FEFFh #32 bit register contents

***Contents after Operation Results:**

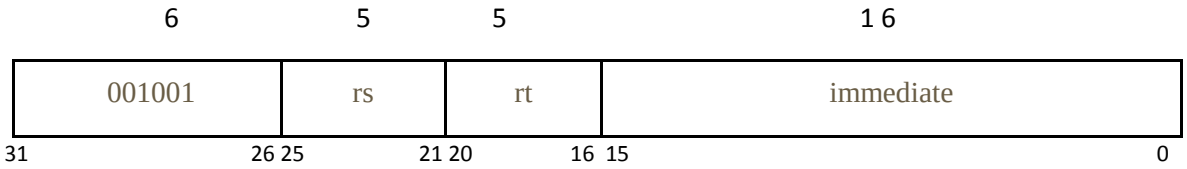


Flags affected after operation(if any)

- Carry(C) : 1'b0
- Overflow(V): 1'b0
- Negative(N): 1'b0
- Zero(Z) : 1'b0

II.E.2.b ADDIU

ADDIU	**Add Immediate Unsigned**

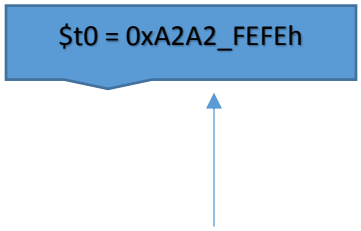


Format	addiu rt, rs,immediate, **New Horizon**
Purpose	To add a 32 bit value held in rs with a 16 bit immediate unsigned value and placed into register rt.
Description	The 16 bit immediate value is added to the contents of rs. The immediate value is sign extended to 32 bit length before being added to rs through the following operation: {16{immediate [15],immediate}.
Operation	temp ← rs + {16{immediate[15], immediate} rt ← temp
Restrictions	Immediate value can only have a value of : $0\ to\ 2^{16} - 1$
Exception	None
Programmer's Notes	addiu rd,rs,$zero : Can use to swap contents of rt with rs.
N,V,C,Z Flags	All flags can be effected.

II.E.2.b.1 Example

Addi unsigned(addiu) Instruction #1

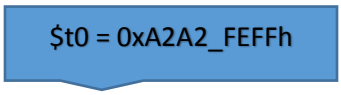
***Initial contents:**




```
addiu $t0, $t0, 0x1 #$t0 = $t0 + $t1
```

$t0 = 0xA2A2_FEFFh #32 bit register contents

***Contents after Operation Results:**

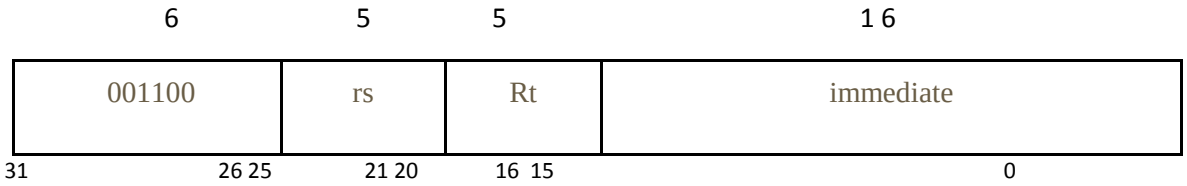


Flags affected after operation(if any)

- Carry(C) : 1'b0
- Overflow(V): 1'b0
- Negative(N): 1'b0
- Zero(Z) : 1'b0

II.E.2.c ANDI

Andi	**And Immediate**



Format | andi rt, rs, immediate **New Horizon**

Purpose | To perform a bitwise logical AND with an immediate value.

Description | To bitwise AND the contents of rs with a 16 bit immediate value. The immediate value is zero extended to 32 bit length before being added to rs through the following operation : {16{0},immediate}.

Operation | temp <--- rs + {16{0}, immediate}
rt <---- temp

Restrictions | None.

Exception | None.

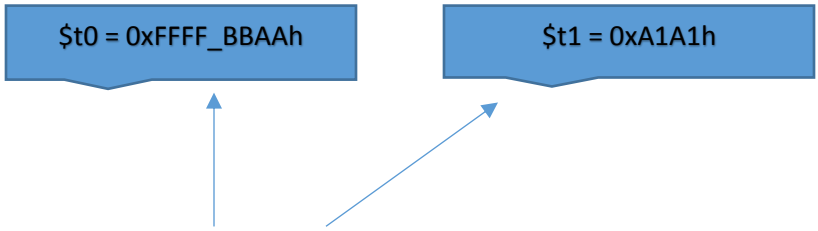
Programmer's Notes | andi rt,rs,$zero : Can use to clear the contents of rt.

N,V,C,Z Flags | Negative and Zero flags are effected

II.E.2.c.1 Example

ANDi(andi) Instruction #1

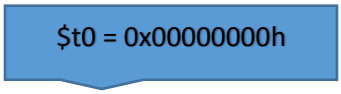
***Initial contents:**




```
andi $t0, $t0, 0x1 #$t0 = $t0 andi $t1
```

$t0 = 0x0000_0000h **#32 bit register contents**

***Contents after Operation Results:**

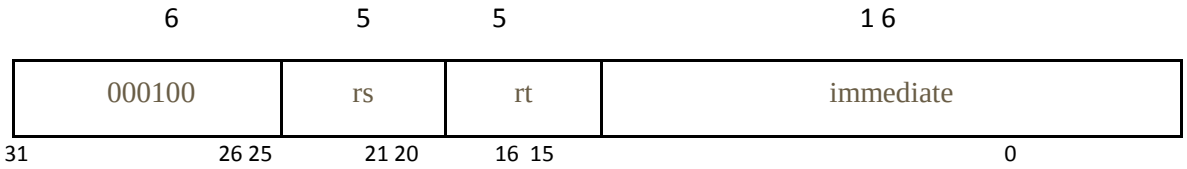


Flags affected after operation(if any)

- Carry(C) : 1'b0
- Overflow(V): 1'b0
- Negative(N): 1'b0
- Zero(Z) : 1'b0

II.E.2.d BEQ

BEQ	**Branch On Equal signed**



Format | beq rt, rs, offset **New Horizon**

Purpose | To compare the values in rs and rt and perform a PC-relative conditional branch

Description | If (rs == rt)

PC <----- (PC +4)+ {14{offset[15]},offset,2'b0}

else

PC<------- PC +4

A 16 bit signed offset is concatenated with two lower order bits making it 18 bits in length and then sign extended into a 32 bit length value.If rs and rt ARE EQUAL to each other,the PC will have the effective address of :

(PC +4)+ {14{offset[15]},offset,2'b0} However, if they ARE NOT EQUAL to each other then PC will have the effective address of: PC +4 .

Operation | Effective Address <--- (PC +4)+ {14{offset[15]},offset,2'b0}

if(rs == rt)

PC<-----Effective Address

else

PC<----- PC + 4

Restrictions | Distance to branch to is limited to -2^{15} to $2^{15} - 1$ from the instruction after the branch

Exception | None.

Programmer's Notes | jump (j) or jump(jr) instructions can be used to jump to addresses farther away.

N,V,C,Z Flags | No flags are affected.

II.E.2.d.1 Example

Branch on Equal (beq) Instruction #1

***Initial contents:**

PC = 0x0

$4 = 0x5A5A_5A5A

```
Add $3, $0, $4          #$t3 = $t3 + $t4
beq $t4, $t3, BRANCH    #$t0 = $t0 beq $t1
                        #if ($t0 == $t1)
                        #     jump to branch address
                        #     (pc = pc + 4 +# branchAddr)
                        #else
                        #     fall through next instr.
mul $t2, $t1, $t3       #$t2 = $t1 * $t3
BRANCH: end   #$t0 = $t0 beq $t1
                        #$t0 = $t0 beq $t1
```

$3 = 0x357A_5C4Ah

***Contents after Operation Results:**

$t3 = 0x5A5A_5A5Ah

$t4 = 0x5A5A_5A5Ah

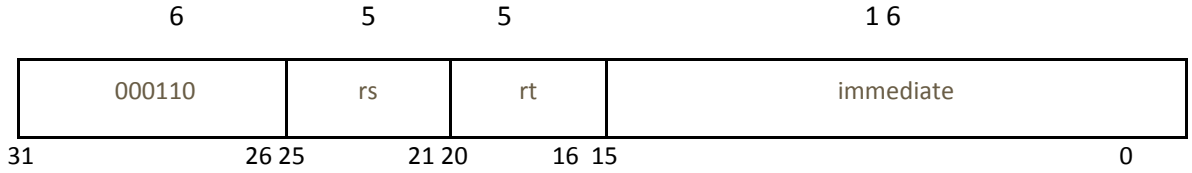
PC = 0xCh

Flags affected after operation(if any)

- Carry(C) : 1'b0
- Overflow(V): 1'b0
- Negative(N): 1'b0
- Zero(Z) : 1'b0

II.E.2.e BLEZ

BLEZ	Branch If Less Than or Equal to Zero



Format | blez rs, offset **New Horizon**

Purpose | To compare the values in rs and rt and perform a PC-relative conditional branch

Description | If (rs <= 0)

PC <----- (PC +4)+ {14{offset[15]},offset,2'b0}

else

PC<------- PC +4

A 16 bit signed offset is concatenated with two lower order bits making it 18 bits in length and then sign extended into a 32 bit length value.If rs is LESS THAN OR EQUAL to zero,,the PC will have the effective address of :

(PC +4)+ {14{offset[15]},offset,2'b0} However, if it is NOT LESS THAN OR EQUAL to zero , then PC will have the effective address of: PC +4 .

Operation | Effective Address <--- (PC +4)+ {14{offset[15]},offset,2'b0}

if(rs <= 0)

PC<-----Effective Address

else

PC<----- PC + 4

Restrictions | Distance to branch to is limited to -2^{15} to $2^{15} - 1$ from the instruction after the branch

Exception | None.

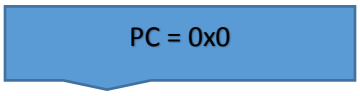
Programmer's Notes | jump (j) or jump(jr) instructions can be used to jump to addresses farther away.

N,V,C,Z Flags | No flags are affected.

II.E.2.e.1 Example

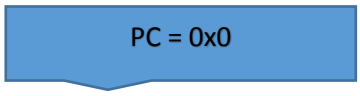
Branch If Less Than or Equal to Zero(BLEZ) Instruction #1

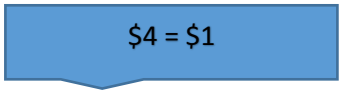
***Initial contents:**

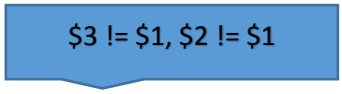



```
        #Effective Address = (pc + 4) + {14{offset[15],offset, 2'b0}}
    blez $0, ELSE #if($t0 <= 0)
                  #  pc = Effective Address
                  #else
                  #  pc = pc + 4
    add $2, $0, $1
    add $3, $0, $1
ELSE: add $t4, $0, $1
END
```

***Contents after Operation Results:**







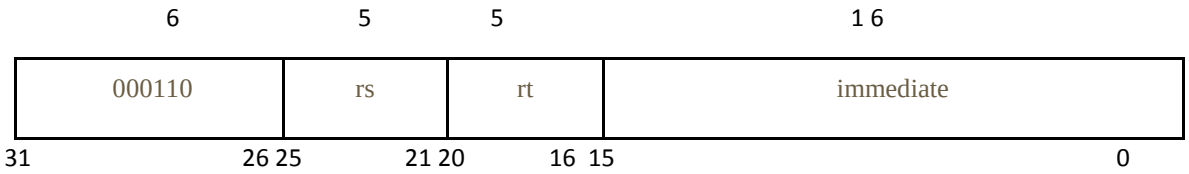
Since $0 is always zero it is at least equal to zero so it will branch.

Flags affected after operation(if any)

- Carry(C) : 1'b0
- Overflow(V): 1'b0
- Negative(N): 1'b0
- Zero(Z) : 1'b0

II.E.2.f BGTZ

BGTZ	**Branch If Greater Than Zero**



Format | bgtz rs, offset **New Horizon**

Purpose | To compare the values in rs and rt and perform a PC-relative conditional branch

Description | If (rs > 0)

PC <----- (PC +4)+ {14{offset[15]},offset,2'b0}

else

PC<------- PC +4

A 16 bit signed offset is concatenated with two lower order bits making it 18 bits in length and then sign extended into a 32 bit length value.If rs is GREATER THAN zero,the PC will have the effective address of :

(PC +4)+ {14{offset[15]},offset,2'b0} However, if it is NOT GREATER THAN zero , then PC will have the effective address of: PC +4 .

Operation | Effective Address <--- (PC +4)+ {14{offset[15]},offset,2'b0}

if(rs => 0)

PC<-----Effective Address

else

PC<----- PC + 4

Restrictions | Distance to branch to is limited to -2^{15} to $2^{15} - 1$ from the instruction after the branch

Exception | None.

Programmer's Notes | jump (j) or jump(jr) instructions can be used to jump to addresses farther away.

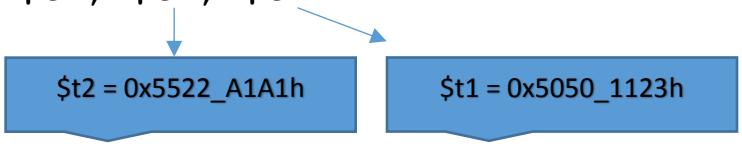
N,V,C,Z Flags | No flags are affected.

II.E.2.f.1 Example

Branch If greater Than Zero(Bgez) Instruction #1

***Initial contents:**

```
#Effective Address = (pc + 4) + {14{offset[15],offset, 2'b0}}
bgtz $0, ELSE #if($t0 >= 0)
                #  pc = Effective Address
                #else
                #  pc = pc + 4
END
ELSE: add $t2, $t2, $t1
```



$t0 = 0x1234_A0A0h **#32 bit register contents**

$t1 = 0x5050_1123h **#32 bit register contents**

$t2 = 0x5522_A1A1h **#32 bit register contents**

***Contents after Operation Results:**

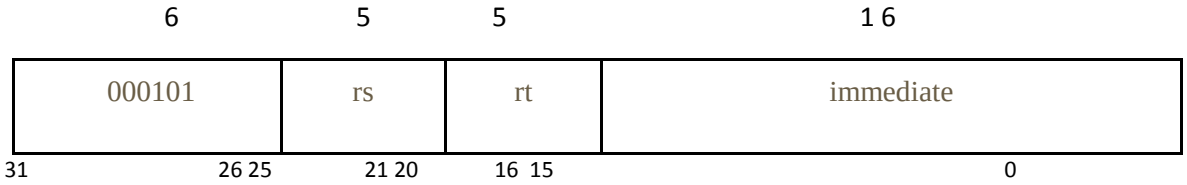
Since in this case the register we are comparing to zero is not greater than zero, the PC does not get loaded with an effective address. Therefore, the add instruction is never executed.

Flags affected after operation(if any)

- Carry(C) : 1'b0
- Overflow(V): 1'b0
- Negative(N): 1'b0
- Zero(Z) : 1'b0

II.E.2.g BNE

BNE	Branch On Not Equal



Format | bne rt, rs, offset **New Horizon**

Purpose | To compare the values in rs and rt and perform a PC-relative conditional branch

Description | If (rs != rt)

PC <----- (PC +4)+ {14{offset[15]},offset,2'b0}

else

PC<------- PC +4

A 16 bit signed offset is concatenated with two lower order bits making it 18 bits in length and then sign extended into a 32 bit length value.If rs and rt ARE NOT EQUAL to each other,the PC will have the effective address of :

(PC +4)+ {14{offset[15]},offset,2'b0} However, if they ARE EQUAL to each other then PC will have the effective address of: PC +4 .

Operation | Effective Address <--- (PC +4)+ {14{offset[15]},offset,2'b0}

if(rs != rt)

PC<-----Effective Address

else

PC<----- PC + 4

Restrictions | Distance to branch to is limited to $-2^{15}\ to\ 2^{15}-1$ from the instruction after the branch

Exception | None.

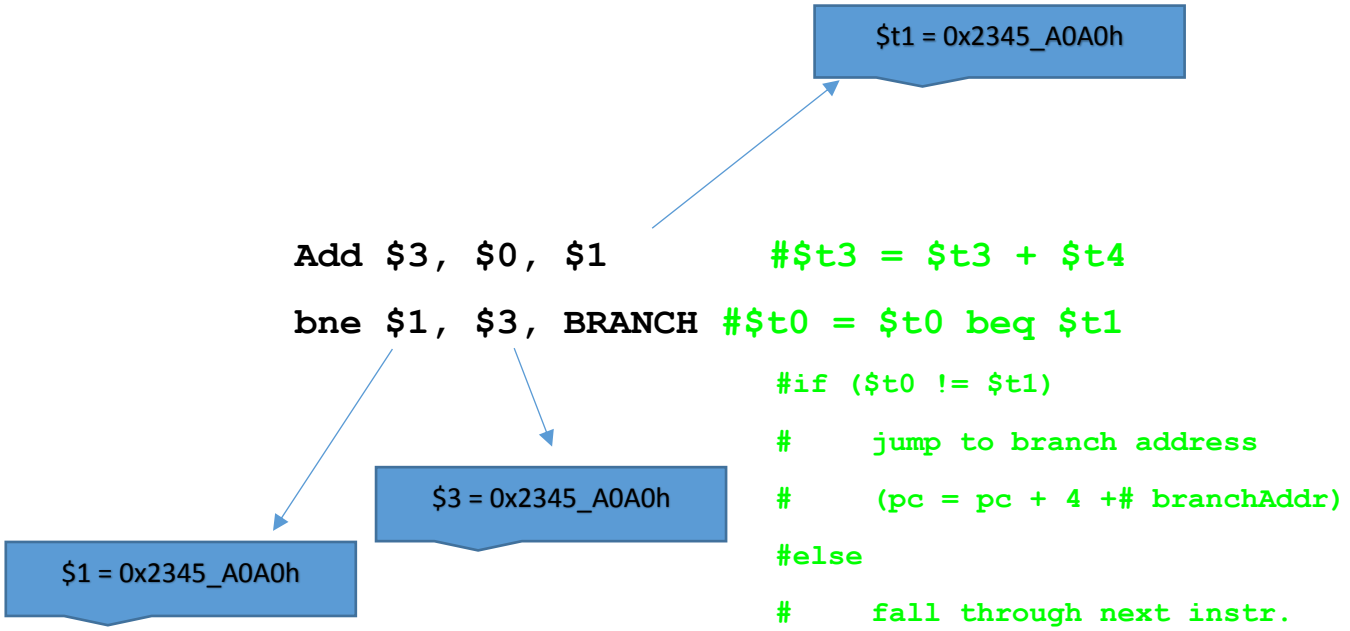
Programmer's Notes | jump (j) or jump(jr) instructions can be used to jump to addresses farther away.

N,V,C,Z Flags | No flags are affected.

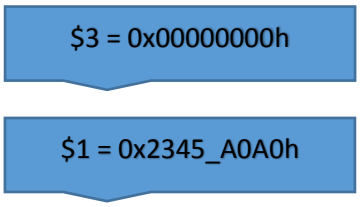
II.E.2.g.1 Example

Branch not Equal (bne) Instruction #1

***Initial contents:**



***Contents after Operation Results:**

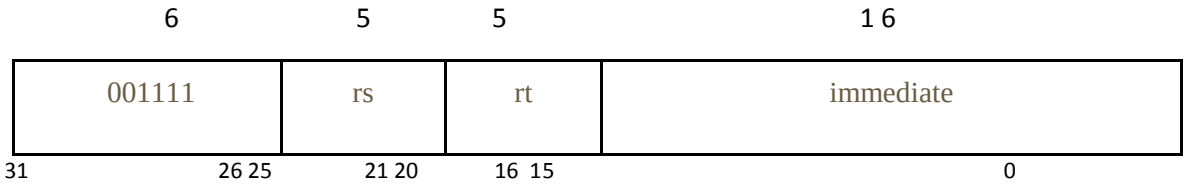


Flags affected after operation(if any)

- Carry(C) : 1'b0
- Overflow(V): 1'b0
- Negative(N): 1'b0
- Zero(Z) : 1'b0

II.E.2.h LUI

LUI	Load Upper Immediate



Format | lui rt, immediate

Purpose | To load the upper 16 bits of rt with the 16 bit immediate value

Description | The 16 bit immediate value is extended to a 32 bit value through the following operation: {immediate,16{0}}.After having extended the value of immediate to 32 bits , the now 32 bit immediate is placed into rt.

Operation | Immediate_32bit ← {immediate,16{0}}
rt ← immediate_32 bit

Restrictions | None

Exception | None.

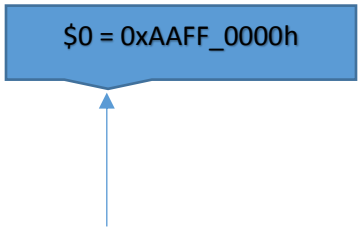
Programmer's Notes | None.

N,V,C,Z Flags | Negative and Zero flags are effected.

II.E.2.h.1 Example

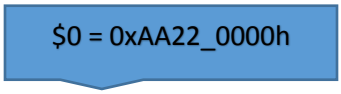
Load Upper Immediate (LUI) Instruction #1

***Initial contents:**




```
lui $0, oxAA22h          #$t0 <- {{0xAA22h}, 16{0}}
```

***Contents after Operation Results:**

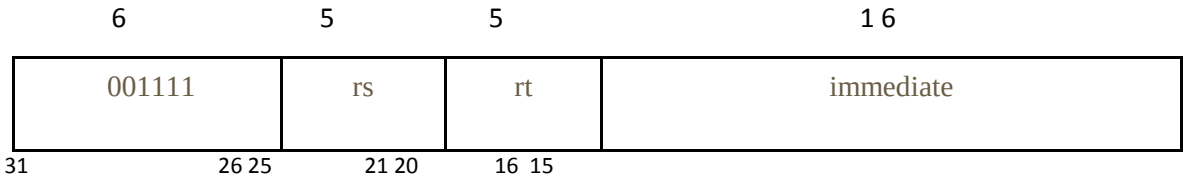


Flags affected after operation (if any)

- Carry(C) : 1'b0
- Overflow(V): 1'b0
- Negative(N): 1'b0
- Zero(Z) : 1'b0

II.E.2.i LW

LW	Load Word



Format | lw rt, immediate (rs)

Purpose | To load word from memory with an address offset

Description | rt<-----Memory[rs +immediate]
The effective memory address location is the 16 bit immediate value (the offset) added with the contents of rs. Before adding the 16 bit immediate and the contents of rs, the immediate value is extended into a 32 bit value through the following operation : {16{immediate[15]},immediate}.

Operation | Immediate_32bit = {16{immediate[15]},immediate}
Effective Address = rs + immediate_ 32bit
rt<--------Memory[Effective Address]

Restrictions | The effective address must obey the word alignment rule

Exception | None.

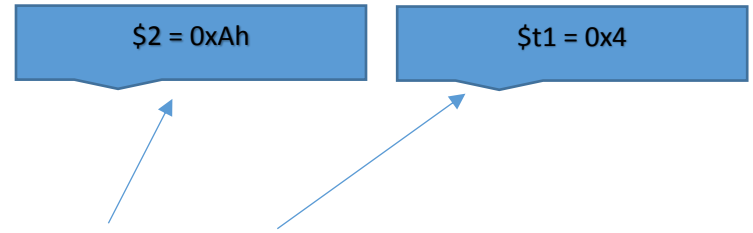
Programmer's Notes | None.

N,V,C,Z Flags | No flags are affected.

II.E.2.i.1 Example

Load Word (lw) Instruction #1

***Initial contents:**




```
lw $2, 0($1)          #$t0 <- {{0xAA22h}, 16{0}}
```

M[0x4] = 0xABCDEF01

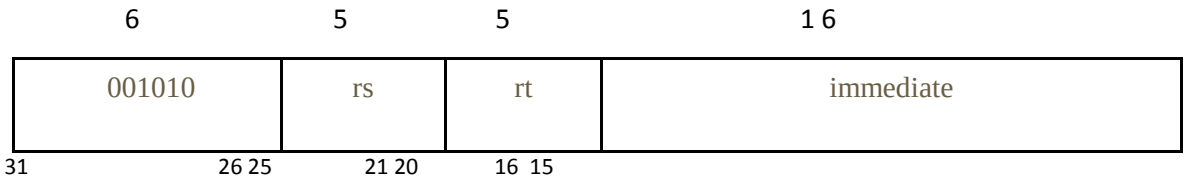
***Contents after Operation Results:**

Flags affected after operation (if any)

- **Carry(C) : 1'b0**
- **Overflow(V): 1'b0**
- **Negative(N): 1'b0**
- **Zero(Z) : 1'b0**

II.E.2.j SLTI

SLTI	Shift Less Than Immediate



Format | slti rt,rs,immediate

Purpose | To determine if rs less than the 16 bit immediate

Description | Immediate_32bit = {16{immediate[15]},immediate}

rt<---(rs<immediate_32bit)

Immediate and rs are signed values.Immediate value is extended to a 32 bit value before being compared to rs through the following operation : {16{immediate[15],immediate}.If rs is LESS THAN sign extended immediate value, then the comparison is true (1), if rs IS NOT LESS THAN rt, then the comparison is false (0).

Operation | Immediate_32bit = {16{immediate[15]},immediate}

```
if( rs<immediate_32bit)
    rt<---1
 else
    rt<---0
```

Restrictions | None.

Exception | None.

Programmer's Notes | None.

N,V,C,Z Flags | All flags can effected.

II.E.2.j.1 Example

Set less than immediate (slti) Instruction #1

***Initial contents:**

```
slti $t1, $t0, 0xF234h      #if($t1 < 0x1234h)
                            #   $t0 = 1
                            #else
                            #   $t0 = 0
```

***Contents after Operation Results:**

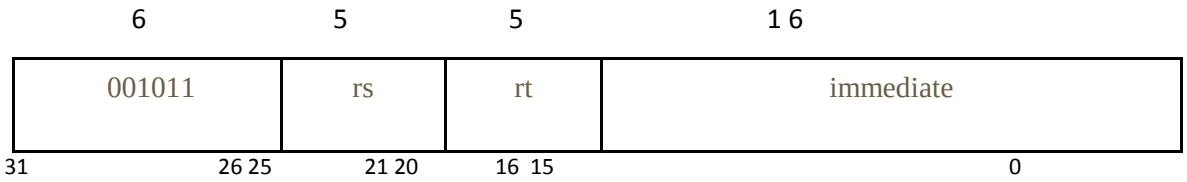
$t1 = 0x1

Flags affected after operation (if any)

- Carry(C) : 1'b0
- Overflow(V): 1'b0
- Negative(N): 1'b0
- Zero(Z) : 1'b0

II.E.2.k SLTIU

SLTIU	Shift Less Than Immediate Unsigned



Format | sltiu rt,rs,immediate

Purpose | To determine if rs less than the 16 bit immediate

Description | Immediate_32bit = {16{immediate[15]},immediate}

rt<---(rs<immediate_32bit)

Immediate and rs are signed values.Immediate value is extended to a 32 bit value before being compared to rs through the following operation : {16{immediate[15],immediate}.If rs is LESS THAN sign extended immediate value, then the comparison is true (1), if rs IS NOT LESS THAN rt, then the comparison is false (0).

Operation |
```
Immediate_32bit = {16{immediate[15]},immediate}
 if( rs<immediate_32bit)
      rt<---1
  else
      rt<---0
```

Restrictions | None.

Exception | None.

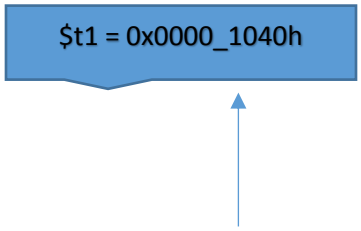
Programmer's Notes | None.

N,V,C,Z Flags | All flags can be effected.

II.E.2.k.1 Example

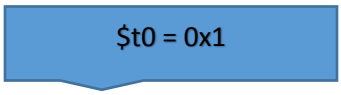
Set less than immediate usigned (sltiu) Instruction #1

***Initial contents:**




```
sltiu $t0, $t1, ox1122h      #if($t1 < 0x1122h)
                             #   $t0 = 1
                             #else
                             #   $t0 = 0
```

***Contents after Operation Results:**

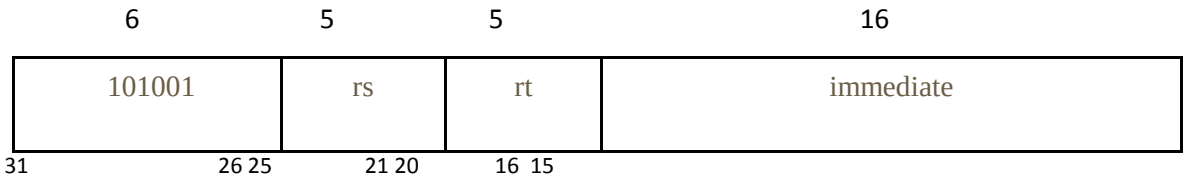


Flags affected after operation (if any)

- Carry(C) : 1'b0
- Overflow(V): 1'b0
- Negative(N): 1'b0
- Zero(Z) : 1'b0

II.E.2.I SW

SW	Store Word



Format | sw rt, immediate (rs)

Purpose | To store the contents of a register to a memory address with an address offset

Description | Memory[rs+immediate]<---- rt
The effective memory address location is the 16 bit immediate value (the offset) added with the contents of rs. Before adding the 16 bit immediate and the contents of rs, the immediate value is extended into a 32 bit value through the following operation : {16{immediate[15]},immediate}. The contents of rt are placed into the effective memory address location

Operation | Immediate_32bit = {16{immediate[15]},immediate}
Effective Address = rs + immediate_ 32bit
Memory[Effective Address]<---------rt

Restrictions | The effective address must obey the word alignment rule

Exception | None.

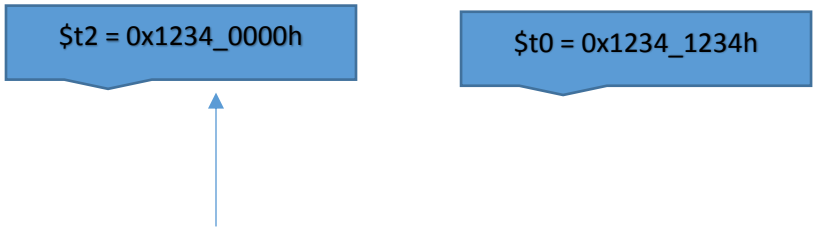
Programmer's Notes | None.

N,V,C,Z Flags | No flags are affected.

II.E.2.l.1 Example

Store Word (sw) Instruction #1

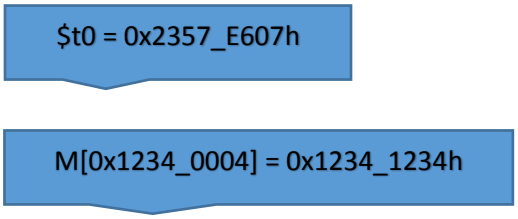
***Initial contents:**




```
sw $t0, 1($t2)      #store $t0 results in memory location 4($t2)
                     #M[4'32b + $t2] = $t0
```

M[0x1234_0004] = 0xABABABAB

***Contents after Operation Results:**

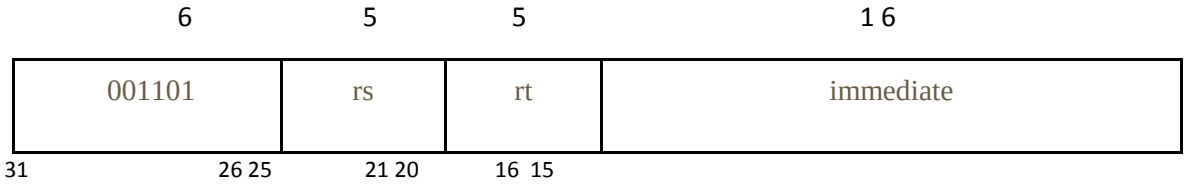


Flags affected after operation (if any)

- Carry(C) : 1'b0
- Overflow(V): 1'b0
- Negative(N): 1'b0
- Zero(Z) : 1'b0

II.E.2.m ORI

ORI **Or Immediate**



Format | ori rt,rs, immediate

Purpose | To bitwise OR the bits of rs with the immediate value

Description | The 16 bit immediate value is bitwise OR'd with the contents of rs and placed into rs. The immediate value is sign extended to 32 bit length before being bitwise OR'd with rs through the following operation : {16{0},immediate}.

Operation | Immidiate_32bit = {16{0},immediate}
rt<----rs OR Immediate_32bit

Restrictions | None.

Exception | None.

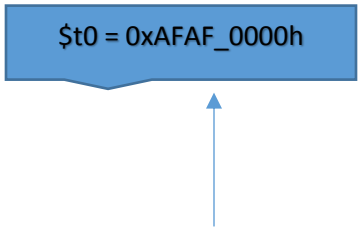
Programmer's Notes | None.

N,V,C,Z Flags | All flags can be effected.

II.E.2.m.1 Example

ORI(ori) Instruction #1

***Initial contents:**




```
ori $t0, $t0, 0x1234h #$t0 = $t0 ori 0x1234h
```

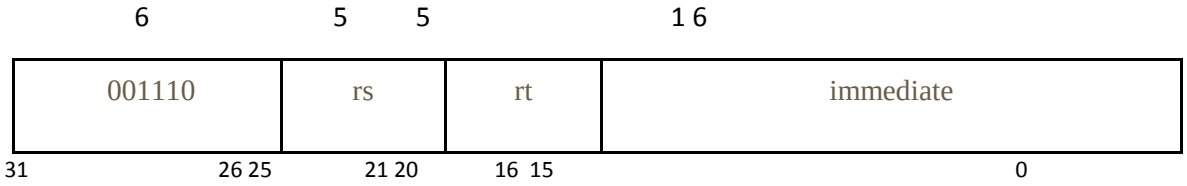
***Contents after Operation Results:**

Flags affected after operation(if any)

- Carry(C) : 1'b0
- Overflow(V): 1'b0
- Negative(N): 1'b0
- Zero(Z) : 1'b0

II.E.2.n XORI

XORI	XOR Immediate



Format | xori rt, rs,immediate

Purpose | To perform a bitwise logical XOR on rs with the immediate value

Description | The 16 bit immediate value is bitwise XOR'd with the contents of rs and placed into rs. The immediate value is sign extended to 32 bit length before being bitwise OR'd with rs through the following operation : {16{0},immediate}.

Operation | Immidiate_32bit = {16{0},immediate}
rt<----rs XOR Immediate_32bit

Restrictions | None

Exception | None.

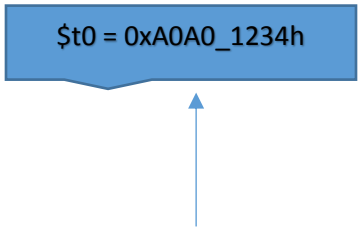
Programmer's Notes | None.

N,V,C,Z Flags | All flags can be effected

II.E.2.n.1 Example

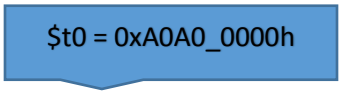
XORI(xori) Instruction #1

***Initial contents:**




```
xori $t0, $t0, 0x1234 #$t0 = $t0 xori 0x1234
```

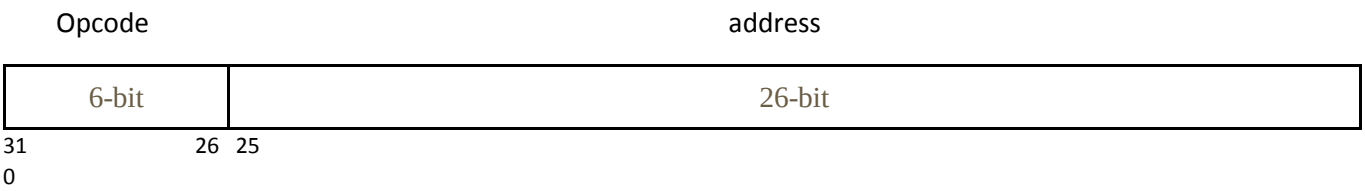
***Contents after Operation Results:**



Flags affected after operation(if any)

- Carry(C) : 1'b0
- Overflow(V): 1'b0
- Negative(N): 1'b0
- Zero(Z) : 1'b0

II.E.3 J-Type Instructions

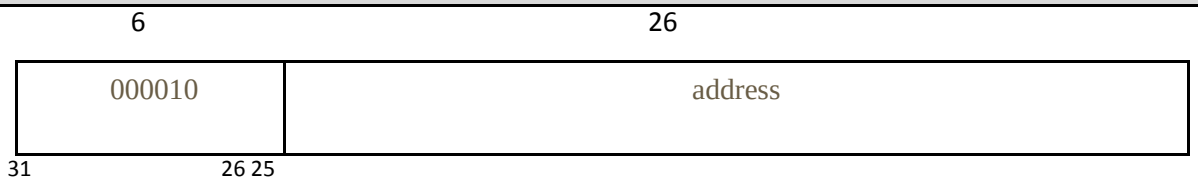


Description of each bit field summarization is as follows:

- **Opcode** : 6 bit value that specifies the J-type instruction to be performed. These are stored at bit locations 31 to 26.
- **Address** : 26 bit value that specifies the destinations absolute address for PC to jump to. This is stored at bit locations 25 to 0.

II.E.3.a Jump

J	Jump



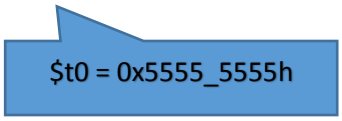
Format	J destination_address
Purpose	To perform an unconditional branch
Description	PC<------- {PC[31:28],address,2'b0} A 26 bit address is concatenated with two lower order bits , making it into 28 bits in length and then concatenated with the upper 4 bits of PC (now equal to PC +4).
Operation	Effective Address : {PC[31:28],address,2'b0} PC← Effective Address
Restrictions	None.
Exception	None.
Programmer's Notes	None.
N,V,C,Z Flags	None.

II.E.3.a.1 Example

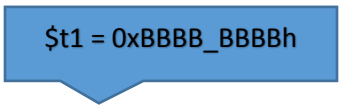
Jump(j) Instruction #1

***Initial contents:**

```
        j  IF                    #jump to IF label
EXIT:.....
ELSE: add $t1, $t1, $t0    #$t1 = $t1 + $t1
IF:    sub $t1, $0, $t0     #$t1 = $t1 - $t0
```



***Contents after Operation Results:**



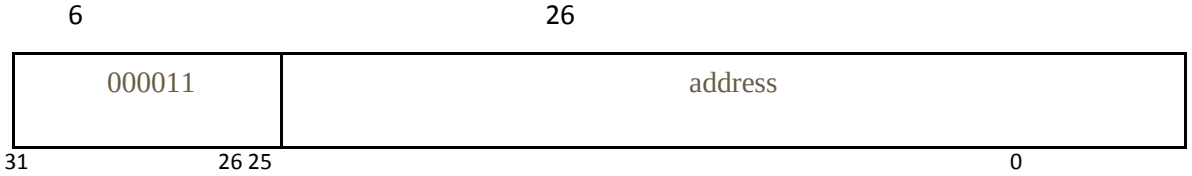
#Result after jumping to IF label and SUB #instruction

Flags affected after operation(if any)

- **Carry(C) : 1'b0**
- **Overflow(V): 1'b0**
- **Negative(N): 1'b0**
- **Zero(Z) : 1'b0**

II.E.3.b JAL

JAL	Jump and Link



Format | Jal destination_address

Purpose | To perform an unconditional branch and save the current value of the PC(now pointing at the next instruction, so PC+4) into register $ra (link register).

Description | $ra<-------PC (value of PC pointing at next instruction)
PC<------- {PC[31:28],destination_address,2'b0}
A 26 bit address is concatenated with two lower order bits , making it into 28 bits in length and then concatenated with the upper 4 bits of PC (now equal to PC +4). The current value of PC is saved to $ra, a return address register.

Operation | $ra <--- PC + 4
Effective Address <------- {PC[31:28],destination_address,2'b0}
PC <----- Effective Address

Restrictions | None.

Exception | None.

Programmer's Notes | If wanting to return to position in program after having jumped to the effective address, perform a jr $ra. The value of PC will be loaded the address of the next instruction below the jal destination_address instruction.

N,V,C,Z Flags | No flags are affected..

II.E.3.b.1 Example

Jump and Link(jal) Instruction #1

***Initial contents:**

PC before = 0x0000_0000h

```
jal  0x0BCD_EF12
```

***Contents after Operation Results:**

PC = 0x0BCD_EF12h

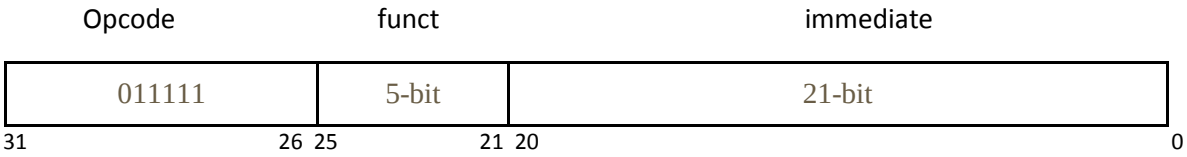
Flags affected after operation(if any)

- Carry(C)   : 1'b0
- Overflow(V): 1'b0
- Negative(N): 1'b0
- Zero(Z)   : 1'b0

II.E.4 Enhanced Instructions

The following are the three enhanced instruction formats:

II.E.4.a Jump Enhanced Instruction format



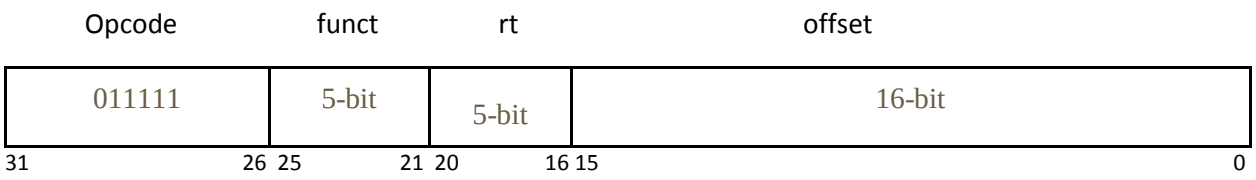
Description of each bit field summarization is as follows:

- **Opcode** : 6 bit value that is set constant to the value 0x1F, bit field is bits [31:26]
- **funct** : 5 bit value indication what operation will be performed, bit field is bits [25:21]
- **immediate** : 16 bit value that will be used only if the selected enhanced instruction is a jump by flag value operation.

Enhanced instructions that use this format

Name	Description	funct [25:21]
JC	Jump if Carry	5'H09
JN	Jump if Negative	5'H0B
JZ	Jump if Zero	5'h0A
JV	Jump if Overflow	5'h0C

II.E.4.b Branch Enhanced Instruction format



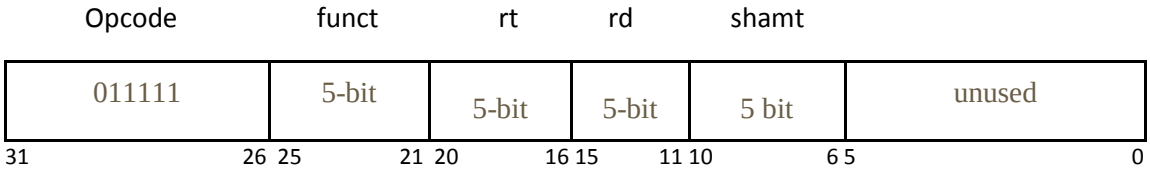
Description of each bit field summarization is as follows:

- **Opcode** : 6 bit value that is set constant to the value 0x1F, bit field is bits [31:26]
- **funct** : 5 bit value indication what operation will be performed, bit field is bits [25:21]
- **rt** : 5 bit value pointing at a register to be used in the instruction as a operand [20:16]
- **offset** : 16 bit value that will be used to branch to the selected address. [15:0]

Enhanced instructions that use this format:

Name	Description	funct [25:21]
BLTZ	Branch if less than zero	5'h00
BGEZ	Branch if greater than or equal to zero	5'h01

II.E.4.c Barrel Shift with rotate Instruction format



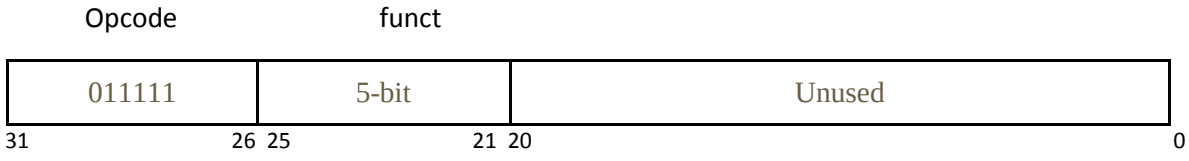
Description of each bit field summarization is as follows:

- **Opcode** : 6 bit value that is set constant to the value 0x1F, bit field is bits [31:26]
- **funct** : 5 bit value indication what operation will be performed, bit field is bits [25:21]
- **rt** : 5 bit value indicating which register to be used in the instruction as a operand [20:16]
- **rd** : 5 bit value indicating which register will have the values produced in the operation stored into it.
- **Shamt** : 5 bit value that will be used as the amount of times rt will be shifted

Enhanced instructions that use this format:

Name	Description	funct [25:21]
BSRR	Barrel shift right with rotate	5'h06
BSLR	Barrel shift left with rotate	5'h04

II.E.4.d No Operation enhanced instruction format

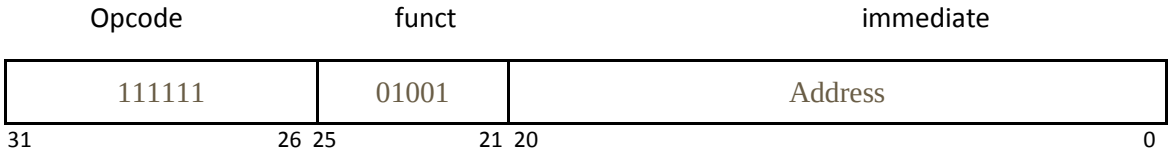


- **Opcode** : 6 bit value that is set constant to the value 0x1F, bit field is bits [31:26]
- **funct** : 5 bit value indication what operation will be performed, bit field is bits [25:21]

Only one instruction uses this format : **NOOP,** the no operation instruction

II.E.4.e JC

JC	Jump if carry



Format | J destination **New Horizon**

Purpose | To perform a conditional jump dependent on the value of the carry flag.

Description | PC<------- {PC[31:24],address,2'b00}

A 21 bit address is concatenated with two lower order bits , making it into 23 bits in length and then concatenated with the upper 8 bits of PC (now equal to PC +4).

Operation | If (carry flag is high)

Effective Address ← {PC[31:28],address,2'b0}

PC← Effective Address

Else

PC ← PC + 4

Restrictions | None.

Exception | None.

Programmer's Notes | Has a smaller jump range than jump if register (jr) and jump (j)

N,V,C,Z Flags | None.

II.E.4.e.1 Example

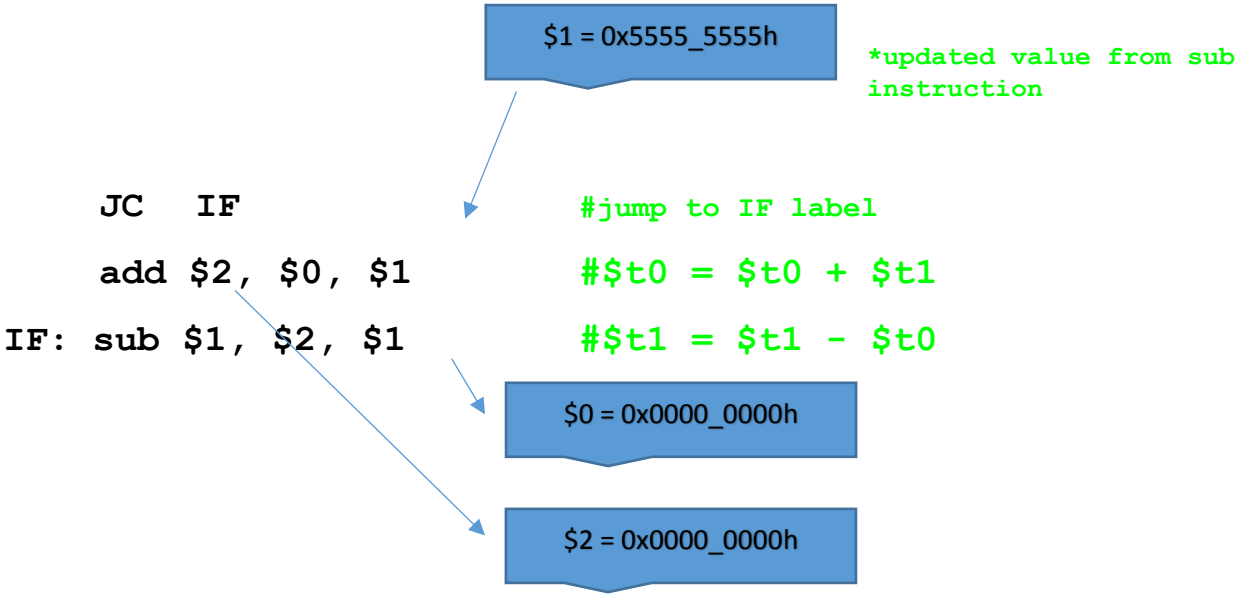
Jump if Carry Instruction #1

***Initial contents:**

PC = 0x0000_0000h

Flag Register:

- Carry(C) : 1'b1
- Overflow(V): 1'b0
- Negative(N): 1'b0
- Zero(Z) : 1'b0



***Contents after Operation Results:**

PC = 0x0000_0008h

#Since the carry flag was set the PC is incremented to the label address

$0 = 0x0000_0000h

#Result for zero register shall always be #zero

$1 = 0xBBBB_BBBBh

#Since at the time of the sub $2 was still #zero because of the jump over the #instruction, $1 = twos complement of #itself.

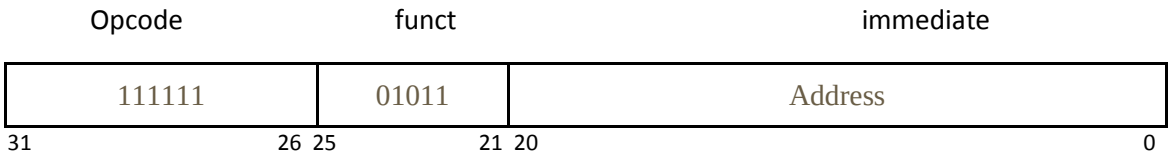
$2 = 0x0000_0000h

Flags affected after operation(if any)

- **Carry(C) : 1'b1**
- **Overflow(V): 1'b0**
- **Negative(N): 1'b0**
- **Zero(Z) : 1'b0**

II.E.4.f JN

JN	Jump if negative



Format | J destination **New Horizon**

Purpose | To perform a conditional jump dependent on the value of the negative flag.

Description | PC<------- {PC[31:24],address,2'b00}

A 21 bit address is concatenated with two lower order bits , making it into 23 bits in length and then concatenated with the upper 8 bits of PC (now equal to PC +4).

Operation | If (negative flag is high)

Effective Address ← {PC[31:28],address,2'b0}

PC← Effective Address

Else

PC ← PC + 4

Restrictions | None.

Exception | None.

Programmer's Notes | Has a smaller jump range than jump if register (jr) and jump (j)

N,V,C,Z Flags | None.

II.E.4.f.1 Example

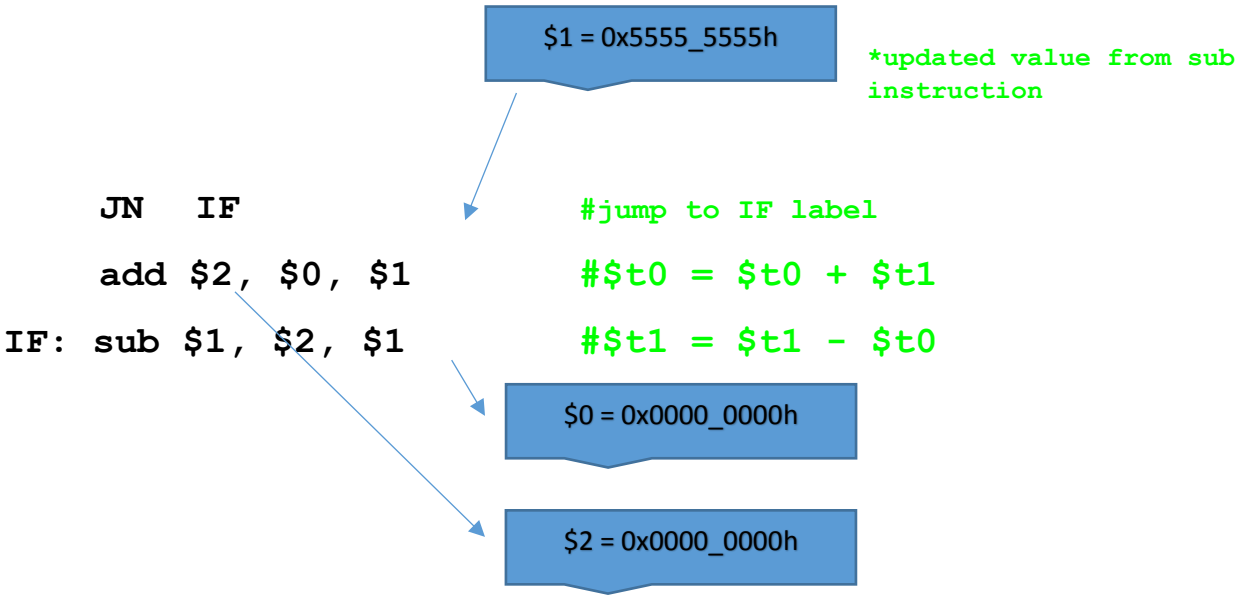
Jump if Negative Instruction #1

***Initial contents:**

PC = 0x0000_0000h

Flag Register:

- Carry(C) : 1'b0
- Overflow(V): 1'b0
- Negative(N): 1'b1
- Zero(Z) : 1'b0



***Contents after Operation Results:**

PC = 0x0000_0008h

#Since the carry flag was set the PC is incremented to the label address

$0 = 0x0000_0000h

#Result for zero register shall always be #zero

$1 = 0xBBBB_BBBBh

#Since at the time of the sub $2 was still #zero because of the jump over the #instruction, $1 = twos complement of #itself.

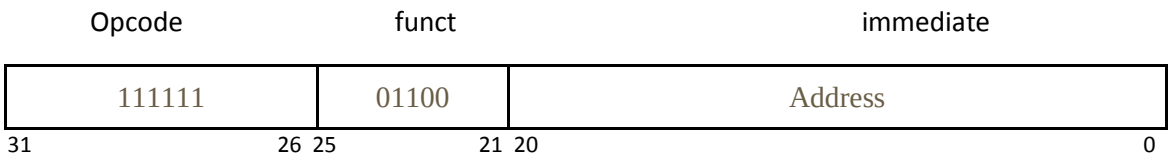
$2 = 0x0000_0000h

Flags affected after operation(if any)

- **Carry(C) : 1'b0**
- **Overflow(V): 1'b0**
- **Negative(N): 1'b1**
- **Zero(Z) : 1'b0**

II.E.4.g JV

JV	Jump if overflow



Format | J destination **New Horizon**

Purpose | To perform a conditional jump dependent on the value of the overflow flag.

Description | PC<------- {PC[31:24],address,2'b00}

A 21 bit address is concatenated with two lower order bits , making it into 23 bits in length and then concatenated with the upper 8 bits of PC (now equal to PC +4).

Operation | If (overflow flag is high)

Effective Address ← {PC[31:28],address,2'b0}

PC← Effective Address

Else

PC ← PC + 4

Restrictions | None.

Exception | None.

Programmer's Notes | Has a smaller jump range than jump if register (j) and jump (j)

N,V,C,Z Flags | None.

II.E.4.g.1 Example

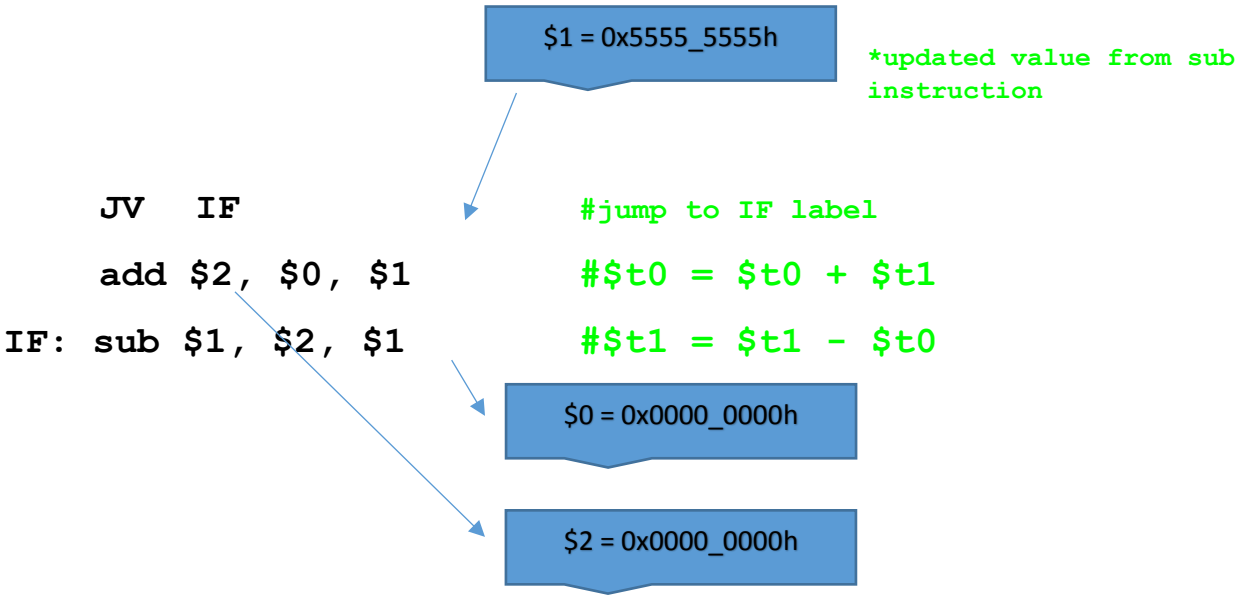
Jump if Overflow Instruction #1

***Initial contents:**

PC = 0x0000_0000h

Flag Register:

- Carry(C) : 1'b0
- Overflow(V): 1'b1
- Negative(N): 1'b0
- Zero(Z) : 1'b0



***Contents after Operation Results:**

PC = 0x0000_0008h

#Since the carry flag was set the PC is incremented to the label address

$0 = 0x0000_0000h

#Result for zero register shall always be #zero

$1 = 0xBBBB_BBBBh

#Since at the time of the sub $2 was still #zero because of the jump over the #instruction, $1 = twos complement of #itself.

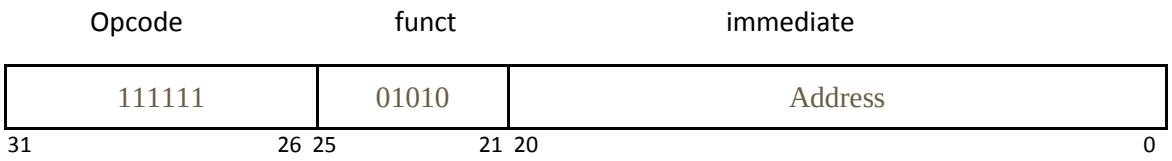
$2 = 0x0000_0000h

Flags affected after operation(if any)

- **Carry(C) : 1'b0**
- **Overflow(V): 1'b1**
- **Negative(N): 1'b0**
- **Zero(Z) : 1'b0**

II.E.4.h JZ

JZ	Jump if zero



Format | J destination **New Horizon**

Purpose | To perform a conditional branch dependent on the value of the zero flag.

Description | PC<------- {PC[31:24],address,2'b00}

A 21 bit address is concatenated with two lower order bits , making it into 23 bits in length and then concatenated with the upper 8 bits of PC (now equal to PC +4).

Operation | If (zero flag is high)

Effective Address ← {PC[31:28],address,2'b0}

PC← Effective Address

Else

PC ← PC + 4

Restrictions | None.

Exception | None.

Programmer's Notes | Has a smaller jump range than jump if register (jr) and jump (j)

N,V,C,Z Flags | None.

II.E.4.h.1 Example

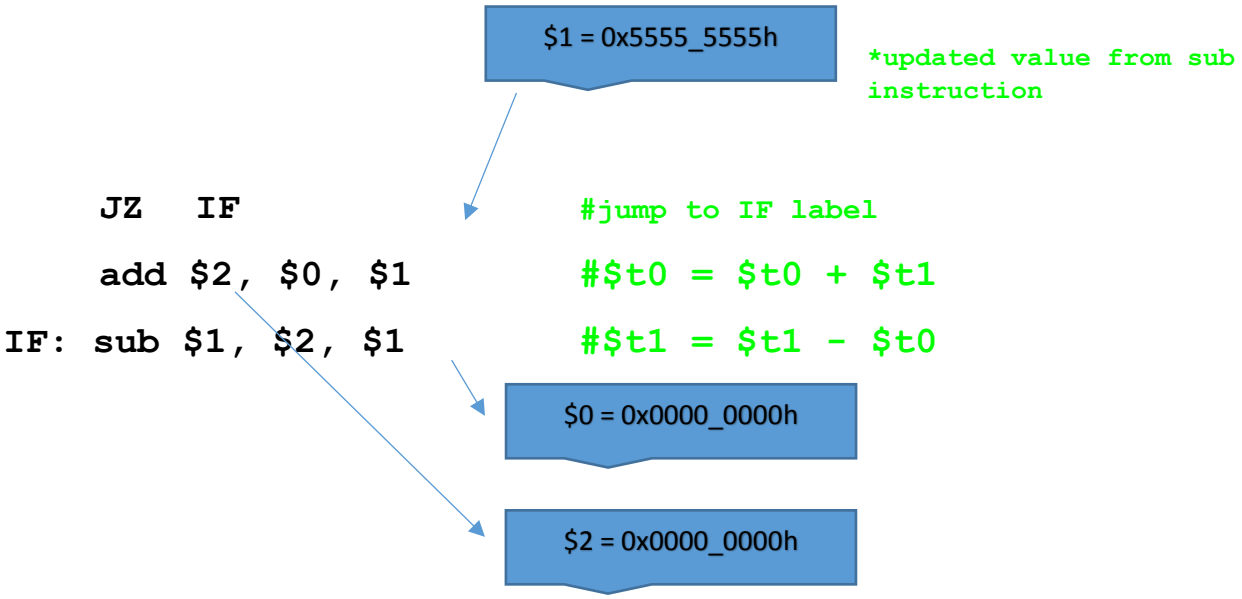
Jump if Zero Instruction #1

***Initial contents:**

PC = 0x0000_0000h

Flag Register:

- Carry(C) : 1'b0
- Overflow(V): 1'b0
- Negative(N): 1'b0
- Zero(Z) : 1'b1



***Contents after Operation Results:**

PC = 0x0000_0008h

#Since the carry flag was set the PC is incremented to the label address

$0 = 0x0000_0000h

#Result for zero register shall always be #zero

$1 = 0xBBBB_BBBBh

#Since at the time of the sub $2 was still #zero because of the jump over the #instruction, $1 = twos complement of #itself.

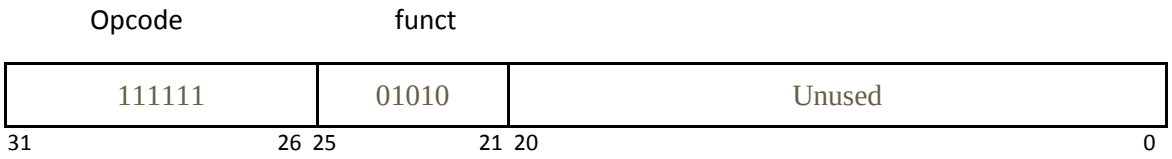
$2 = 0x0000_0000h

Flags affected after operation(if any)

- **Carry(C) : 1'b0**
- **Overflow(V): 1'b0**
- **Negative(N): 1'b0**
- **Zero(Z) : 1'b1**

II.E.4.i NOOP

NOOP	**No Operation**



Format	NOOP	**New Horizon**
Purpose	To provide a 3 clock cycle time of no operations taking place	
Description	No operation is taking place for 3 clock cycles.	
Operation	To perform no operation for 3 clock cycles.	
Restrictions	None.	
Exception	None.	
Programmer's Notes	Has a smaller jump range than jump if register (jr) and jump (jmp)	
N,V,C,Z Flags	None.	

II.E.4.i.1 Example

NO OPERATION Instruction #1

***Initial contents:**

Flag Register:

- **Carry(C) : 1'b0**
- **Overflow(V): 1'b0**
- **Negative(N): 1'b0**
- **Zero(Z) : 1'b1**

```
add $2, $0, $1          #$t0 = $t0 + $t1
NOOP                    #does nothing for 3 clock cycles
```

***Contents after Operation Results:**

$0 = 0x0000_0000h

#Result for zero register shall always be #zero

$1 = 0xBBBB_BBBBh

#Since at the time of the sub $2 was still #zero because of the jump over the #instruction, $1 = twos complement of #itself.

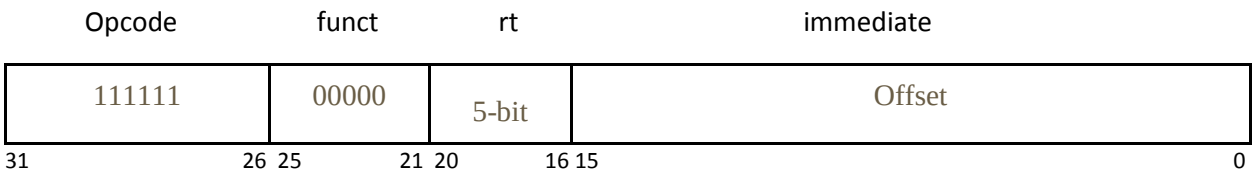
$2 = 0xBBBB_BBBBh

Flags affected after operation(if any)

- **Carry(C) : 1'b0**
- **Overflow(V): 1'b0**
- **Negative(N): 1'b0**
- **Zero(Z) : 1'b1**

II.E.4.j BLTZ

BLTZ	Branch if less than zero



Format | bltz rt, offset **New Horizon**

Purpose | To determine if rt is less than zero and perform a PC-relative conditional branch

Description | If (rt<0)

PC ←(PC +4)+ {14{offset[15]},offset,2'b0}

else

PC← PC +4

A 16 bit signed offset is concatenated with two lower order bits making it 18 bits in length and then sign extended into a 32 bit length value. If rt is less than 0 ,the PC will have the effective address of :(PC +4)+ {14{offset[15]},offset,2'b0}

However, if rt is not less than zero, then PC will have the effective address of: PC +4 .

Operation | Effective Address ← (PC +4)+ {14{offset[15]},offset,2'b0}

if(rt<0)

PC←Effective Address

else

PC← PC + 4

Restrictions | Distance to branch to is limited to -2^{15} to $2^{15} - 1$ from the instruction after the branch

Exception | None.

Programmer's Notes | jump (j) or jump(jr) instructions can be used to jump to addresses farther away.

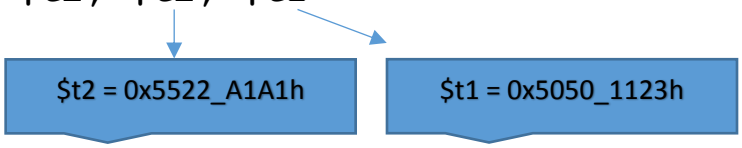
N,V,C,Z Flags | No flags are affected.

II.E.4.j.1 Example Need

Branch If Less Than Zero(BLTZ) Instruction #1

***Initial contents:**

```
            #Effective Address = (pc + 4) + {14{offset[15],offset, 2'b0}}
     bltz $0, ELSE #if($t0 >= 0)
                          #  pc = Effective Address
                          #else
                          #  pc = pc + 4
END
*
*
*
ELSE: add $t2, $t2, $t1
```



$t0 = 0x1234_A0A0h **#32 bit register contents**

$t1 = 0x5050_1123h **#32 bit register contents**

$t2 = 0x5522_A1A1h **#32 bit register contents**

***Contents after Operation Results:**

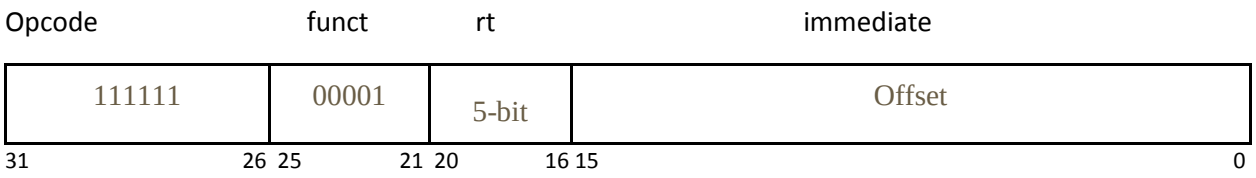
Since in this case the register we are comparing to zero is not less than zero, the PC does not get loaded with an effective address. Therefore, the add instruction is never executed.

Flags affected after operation(if any)

- Carry(C) : 1'b0
- Overflow(V): 1'b0
- Negative(N): 1'b0
- Zero(Z) : 1'b0

II.E.4.k BGEZ

BGEZ	Branch if greater than or equal to zero



Format | bltz rt, offset **New Horizon**

Purpose | To determine if rt is less than zero and perform a PC-relative conditional branch

Description | If (rt => 0)

PC ← (PC +4)+ {14{offset[15]},offset,2'b0}

else

PC← PC +4

A 16 bit signed offset is concatenated with two lower order bits making it 18 bits in length and then sign extended into a 32 bit length value. If rt is greater than or equal to 0 ,the PC will have the effective address of :(PC +4)+ {14{offset[15]},offset,2'b0}

However, if rt is not greater than or equal to zero, then PC will have the effective address of: PC +4 .

Operation | Effective Address <--- (PC +4)+ {14{offset[15]},offset,2'b0}

if(rt => 0)

PC←Effective Address

else

PC← PC + 4

Restrictions | Distance to branch to is limited to -2^{15} to $2^{15} - 1$ from the instruction after the branch

Exception | None.

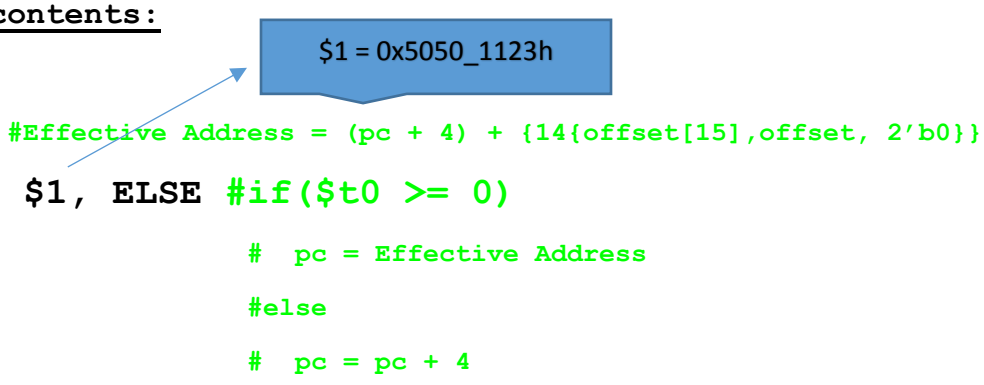
Programmer's Notes | jump (j) or jump(jr) instructions can be used to jump to addresses farther away.

N,V,C,Z Flags | No flags are affected.

II.E.4.k.1 Example NEED

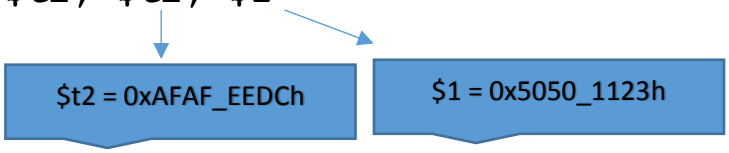
Branch If Greater Than or Equal to Zero(BGEZ) Instruction #1

***Initial contents:**



END

ELSE: add $t2, $t2, $1



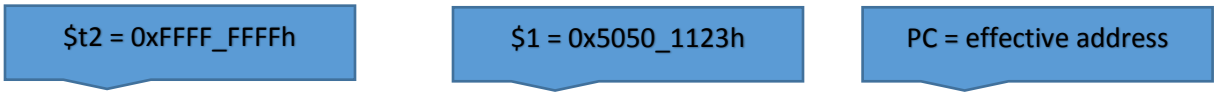
$t0 = 0x1234_A0A0h **#32 bit register contents**

$t1 = 0x5050_1123h **#32 bit register contents**

$t2 = 0x5522_A1A1h **#32 bit register contents**

***Contents after Operation Results:**

Since in this case the register we are comparing to zero is greater than zero, the PC does will get loaded with an effective address. Thus the resultant of this snippet is:

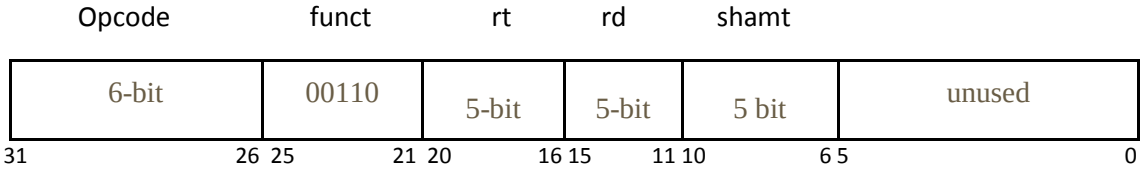


Flags affected after operation(if any)

- **Carry(C) : 1'b0**
- **Overflow(V): 1'b0**
- **Negative(N): 1'b0**
- **Zero(Z) : 1'b0**

II.E.4.l Barrel Shift Right Rotate BSRR

BSRR	**Barrel shift right with rotate**



Format | bsrr rd, rt, shamt **New Horizon**

Purpose | To shift and rotate the contents in rt to the left by the amount specified in shamt..

Description | The contents of rt is rotated by inserting the lsb into the msb location and shifting the contents of the registers to the right. The amount of times the register will be shifted to the right and rotated is by the value specified in the shamt field.

Operation | temp ← rt >> shamt
rd ← temp

Restrictions | Can only shift contents of the source register (rt) a maximum of 31 times in one bsrl call.

Exception | None.

Programmer's Notes | None.

N,V,C,Z Flags | Overflow and carry flags are unaffected by this operations

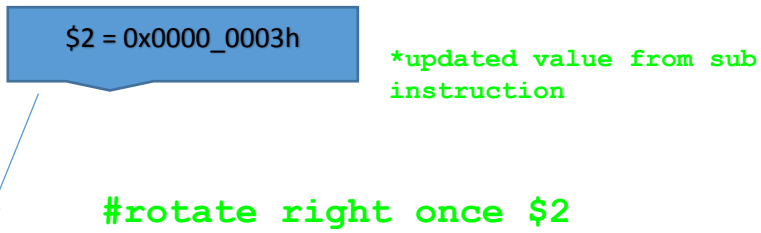
II.E.4.l.1 Example

Barrel Shift Right Rotate Instruction #1

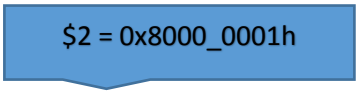
***Initial contents:**

Flag Register:

- **Carry(C) : 1'b0**
- **Overflow(V): 1'b0**
- **Negative(N): 1'b0**
- **Zero(Z) : 1'b1**



***Contents after Operation Results:**

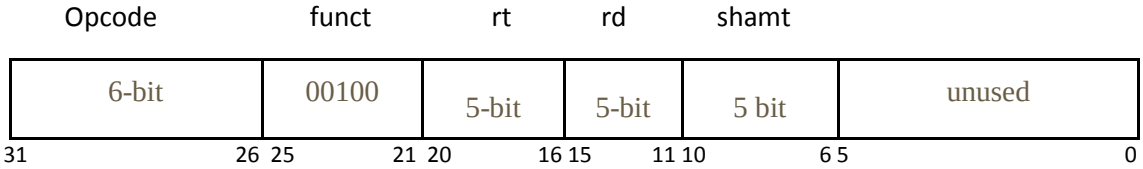


Flags affected after operation(if any)

- **Carry(C) : 1'b0**
- **Overflow(V): 1'b0**
- **Negative(N): 1'b0**
- **Zero(Z) : 1'b1**

II.E.4.m Barrel Shift Left Rotate BSLR

BSLR	**Barrel shift left with rotate**



Format | bslr rd, rt, shamt **New Horizon**

Purpose | To shift and rotate the contents in rt to the left by the amount specified in shamt..

Description | The contents of rt is rotated by inserting the msb into the lsb location and shifting the contents of the registers to the left. The amount of times the register will be shifted to the left and rotated is by the value specified in the shamt field.

Operation | temp ← rt << shamt
rd ← temp

Restrictions | Can only shift contents of the source register (rt) a maximum of 31 times in one bsrl call.

Exception | None.

Programmer's Notes | None.

N,V,C,Z Flags | Overflow and carry flags are unaffected by this operation.

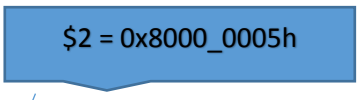
II.E.4.m.1 Example

Barrel Shift Left Rotate Instruction #1

***Initial contents:**

Flag Register:

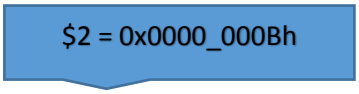
- **Carry(C) : 1'b0**
- **Overflow(V): 1'b0**
- **Negative(N): 1'b0**
- **Zero(Z) : 1'b1**



***updated value from sub instruction**

```
BSLR $2, $2, #1      #rotate right once $2
```

***Contents after Operation Results:**



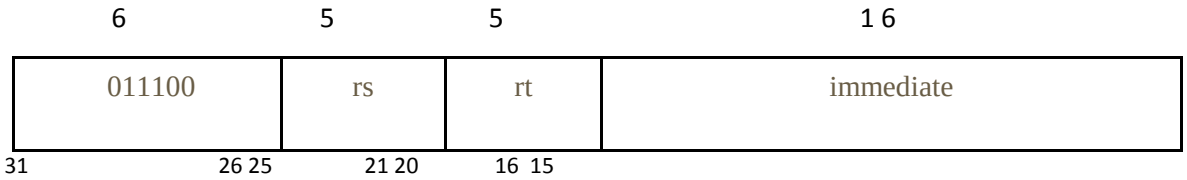
#the MSB of $2 becomes the lsb while the register is being is shifted to the left.

Flags affected after operation(if any)

- **Carry(C) : 1'b0**
- **Overflow(V): 1'b0**
- **Negative(N): 1'b0**
- **Zero(Z) : 1'b1**

II.E.4.n Input

INPUT	Input



Format | Input rt, immediate (rs)

Purpose | To load word from I/O memory with an address offset

Description | rt<-----Memory[rs +immediate]
The effective I/O memory address location is the 16 bit immediate value (the offset) added with the contents of rs. Before adding the 16 bit immediate and the contents of rs, the immediate value is extended into a 32 bit value through the following operation : {16{immediate[15]},immediate}.

Operation | Immediate_32bit = {16{immediate[15]},immediate}
Effective Address = rs + immediate_ 32bit
rt<--------I/O Memory[Effective Address]

Restrictions | The effective address must obey the word alignment rule

Exception | None.

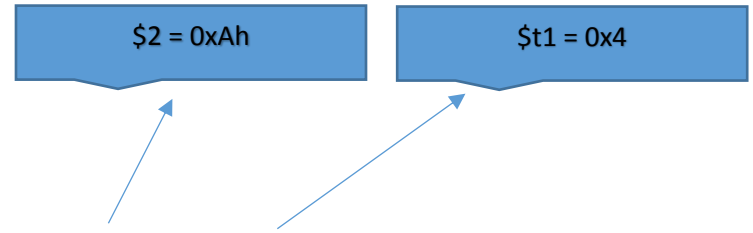
Programmer's Notes | None.

N,V,C,Z Flags | No flags are affected.

II.E.4.n.1 Example

INPUT Instruction #1

***Initial contents:**




```
lw $2, 0($1)          #$t0 <- {{0xAA22h}, 16{0}}
```

IOMEM[0x4] = 0xABCDEF01

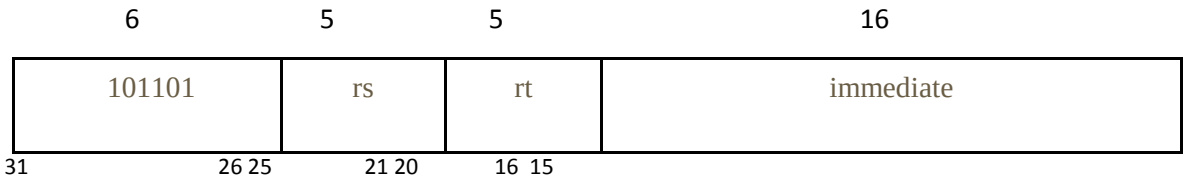
***Contents after Operation Results:**

Flags affected after operation (if any)

- **Carry(C) : 1'b0**
- **Overflow(V): 1'b0**
- **Negative(N): 1'b0**
- **Zero(Z) : 1'b0**

II.E.4.o OUTPUT

OUTPUT	Output



Format | input rt, immediate (rs)

Purpose | To store the contents of a register to a memory address with an address offset

Description | I/O Memory[rs+immediate]<---- rt
The effective I/o memory address location is the 16 bit immediate value (the offset) added with the contents of rs. Before adding the 16 bit immediate and the contents of rs, the immediate value is extended into a 32 bit value through the following operation :
{16{immediate[15]},immediate}. The contents of rt are outputted into the effective memory address location

Operation | Immediate_32bit = {16{immediate[15]},immediate}
Effective Address = rs + immediate_ 32bit
I/O Memory[Effective Address]<---------rt

Restrictions | The effective address must obey the word alignment rule

Exception | None.

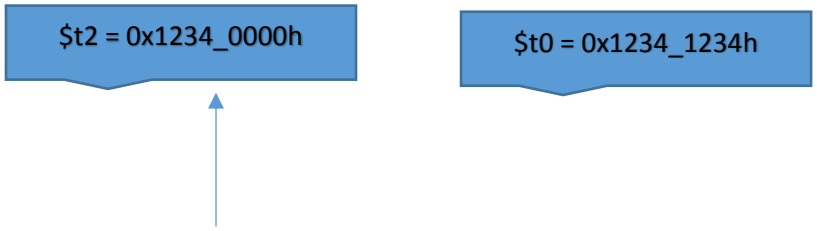
Programmer's Notes | None.

N,V,C,Z Flags | No flags are affected.

II.E.4.o.1 Example

OUTPUT Instruction #1

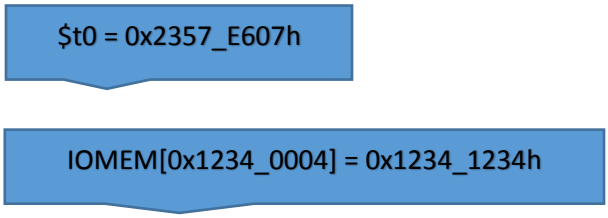
***Initial contents:**




```
sw $t0, 1($t2)      #store $t0 results in memory location 4($t2)
                    #M[4'32b + $t2] = $t0
```

IOMEM[0x1234_0004] = 0xABABABAB

***Contents after Operation Results:**



Flags affected after operation (if any)

- Carry(C) : 1'b0
- Overflow(V): 1'b0
- Negative(N): 1'b0
- Zero(Z) : 1'b0

III Verilog Implementation/Design/Verification

III.A Top Level MIPS

```
`timescale 1ns / 1ps
/****************************** C E C S  4 4 0 ******************************
 *
 * File Name:  Top lvlTB.v
 * Designer:   Kassandra Flores, Adan Hernandez
 * Email:      kassandra.flores@student.csulb.edu,
 *                  adan.hernandez@student.csulb.edu
 * Rev. No.:   Version 1.0
 * Rev. Date:  10/18/2016
 *
 * Purpose: To implement the Instruction Unit, Integer datapath , Data
 * memory and Control unit and create an operating MIPS baeline architecture.
 * Interrupt sources into the CPU will come from a defined I/O space and
 * the cpu is tasked to identify and acknowledge that a interrupt has been
 * requested.
 ***************************************************************************/

module Top_lvlTB;

    // Inputs
    reg sys_clk;
    reg reset;
    reg intr;
    wire [31:0] D_in_Mem;
    wire [31:0] D_in_IO;

    // Outputs
    wire [31:0] ALU_OUT;
    wire dm_wr;
    wire dm_cs;
    wire dm_rd;
    wire IO_wr;
    wire IO_rd;
    wire intr_ack;
    wire intr_req;
    wire [31:0] D_OUT;

    // Instantiate the Unit Under Test (UUT)
    CPU uut (
        .sys_clk(sys_clk),
        .reset(reset),
        .intr_req(intr_req),
        .ALU_OUT(ALU_OUT),
        .dm_wr(dm_wr),
        .dm_cs(dm_cs),
        .dm_rd(dm_rd),
        .IO_wr(IO_wr),
        .IO_rd(IO_rd),
        .D_in_Mem(D_in_Mem),
        .D_in_IO(D_in_IO),
        .intr_ack(intr_ack),
        .D_OUT(D_OUT)
    );

    Data_Memory dm_uut (
    .clk(sys_clk),
    .dm_cs(dm_cs),
    .dm_wr(dm_wr),
    .dm_rd(dm_rd),
    .D_Out(D_in_Mem),
    .Addr(ALU_OUT[11:0]),
    .D_In(D_OUT)
    );


I_O
```

```
    IOMEM (
    .sys_clk(sys_clk),
    .intr(intr),
    .Addr(ALU_OUT[11:0]),
    .D_In(D_OUT),
    .intr_req(intr_req),
    .intr_ack(intr_ack),
    .IO_wr(IO_wr),
    .IO_rd(IO_rd),
    .IO_D_Out(D_in_IO)
    );


// Create a 10 ns clock
   always
   #5 sys_clk=~sys_clk;

    initial begin
    // Initialize Inputs
        $timeformat(-9, 1, " ns", 9);
        $readmemh("iMem14_Fa16_w_isr_commented.list",uut.IU_uut.Instruction_Memory.Mem);
        $readmemh("dMem14_Fa16.list",dm_uut.Mem);
        sys_clk = 0;
        reset = 1;
        intr = 0;
        $display(" WITH ISR, Interrupt after writing to register 1 to 15 ");
        // Wait 100 ns for global reset to finish
        #100;
      reset = 1'b0;
        // Add stimulus here
         #910 intr = 1;
         @(posedge intr_ack)
         intr = 0;

    end

endmodule
```

III.B Source Code of Supporting Modules

III.B.1 Control Unit

```
`timescale 1ns / 1ps
/***************************** C E C S  4 4 0 ******************************
 *
 * File Name:  MCU.v
 * Designer:   Kassandra Flores, Adan Hernandez
 * Email:      kassandra.flores@student.csulb.edu,
 *                  adan.hernandez@student.csulb.edu
 * Rev. No.:   Version 1.0
 * Rev. Date:  10/11/2016
 *
 * Purpose: A state machine implementing the MIPS Control Unit (MCU) for the
 * major cycles of fetch, decode, execute. All control signals for the Integer
 * datapath and instruction unit will be outputted from the control unit.
 * Inputs into the control unit are the Instruction Register , the
 * status flags and an interrupt. At reset the stack pointer will be set to
 * 32 bit value of 32'h3FC.
 ***************************************************************************/
module MCU(sys_clk,reset,intr, // system inputs
           C,N,Z,V,              // ALU statis inputs
          IR,                       // Instruction Register input
              int_ack,             // Ouput to I/O subsystem
              pc_sel,               // Selects input value into Program Counter Reg
              pc_ld,                    // Load Control into PC register
              pc_inc,                // Increment PC register contents by 4
              ir_ld,                    // Load control into Instrution Register (IR)
              im_cs,                    // Instruction Memory Chip Select
              im_rd,                 // Instruction Memory read control
              im_wr,                    // Instruction Memory write control
              D_En,                  // Register File Data Write Enable
              DA_Sel,                // Register File Destination Address Select
              T_Sel,                    // RT register Content Select
              HILO_ld,               // HI and LO register load control
              Y_Sel,                    // Integer Datapath Output Select
              dm_cs,                    // Data Memory Chip select
              dm_rd,                    // Data Memory Read select
              dm_wr,                    // Data Memory Write select
              D_In_Sel,           // Data In select
              IO_wr,                   // Input Ouput Module write control
              IO_rd,                    // Input Output Module read control
              FS,                       // Function Select
              S_Sel,                    // S_Addr Register File Select
              D_OUT_Sel,          // D_OUT from Integer Datapath Select
              ps_c,                  // Present state carry flag
              ps_n,                // Present state negative flag
              ps_z,                // Present state zero flag
              ps_v,                // present state overflow flag
              ld Flags             // Load Flag control
            );
input sys_clk,reset,intr;   // System clock , reset, and interrupt request
input C,N,Z,V;                  // Integer ALU status flags
input [31:0] IR ;               // Instruction Register Input from IU
output reg ld_Flags;            // Flag Values from return from interrupt
output reg pc_ld,pc_inc;        // Program Counter Load Control, Inc by 4 control
output reg [1:0] pc_sel;       // PC input select
output reg ir_ld;                   // IR register load control
output reg im_cs,im_rd,im_wr;    // Instruction Mem read,chip select,write control
output reg D_En;                    // RF data write enable
output reg [1:0] DA_Sel;        // RF Destination address select
output reg T_Sel;                   // RT register content select
output reg HILO_ld;             // HI and LO reg control
output reg [2:0] Y_Sel;         // Integer Datapath output select
output reg dm_cs,dm_wr,dm_rd;    // Data Mem chip select, write , read control
output reg int_ack;             // Interrupt Acknowledge
output reg D_In_Sel;           // Data In select
output reg [4:0] FS;                // Function Select
output reg IO_wr,IO_rd;         // Input output Mem module read and write control
```

```
output reg S_Sel;                    // S_Addr Register File Select
output reg [1:0] D_OUT_Sel;          // D_OUT form Integer datapath select
integer i ;                          // for loop control variable in task

//////////////////////////////
//      PARAMETERS          //
//////////////////////////////
parameter
RESET  = 00 , FETCH  = 01 , DECODE = 02,
ADD    = 10 , ADDU   = 11 , AND    = 12, OR    = 13, NOR   = 14, SRA = 15,
SLL    = 16 , SLT    = 17 , MULT   = 18, DIV   = 19, XOR   = 9, SLTU = 8,
SUB    = 7  , SUBU   =  6 ,
ORI    = 20 , LUI    = 21 , LW_CALC    = 22, SW    = 23, SRL = 24, ADDI = 25,
SLTI   = 26 ,SLTIU = 27,  XORI = 28, ANDI = 29,
WB_alu = 30 , WB_imm = 31 , WB_Din = 32, WB_hi = 33, WB_lo = 34, WB_mem = 35,
WB_LW  = 36 , LW = 37 , MFLO = 38 , MFHI = 39 ,
INTR_1 = 501, INTR_2 = 502 , INTR_3 = 503, INTR_4 = 504, INTR_5 = 505,
INTR_6 = 506, INTR_7 = 507,
BEQ = 40, BEQ_WR = 41, BNE = 42,
BNE_WR = 43, BLEZ = 45, BGTZ = 46, BLEZ_WR =47 , BGTZ_WR = 48,
JUMP   = 50,  JR_CALC = 51, JR = 52 , JAL = 53, JC = 54,
JN   = 55, JZ = 56, JV = 57,
BREAK  = 510, ILLEGAL_OP = 511,
INPUT  = 60, OUTPUT = 61,SETIE = 62,WB_INPUT = 63, WB_OUTPUT = 64,
INPUT_CALC = 65, RETI = 66, RETI_2 = 67, RETI_3 = 68, RETI_4 = 69,
RETI_5 = 70, RETI_6 = 71, RETI_7 = 72, RETI_8 = 73,

R_Type   = 6'h000_000,

E_KEY    = 6'b01_1111,
BGEZ = 80, BLTZ = 81, NO_OP = 82,  BSLR = 85, BSRR = 86,
BGEZ_WR = 87, BLTZ_WR = 88;


//////////////////////////////////
//          STATE REGISTERS     //
//////////////////////////////////

reg [8:0] state;// 512 possible states

//////////////////////////////////
//              Flag Regsiter   //
//////////////////////////////////
output reg ps_n,ps_z,ps_c,ps_v; // Present state flags
reg ns_n,ns_z,ns_c,ns_v;        // Next state flags

always@(posedge sys_clk , posedge reset)
    begin
    if(reset)
    {ps_n,ps_z,ps_c,ps_v} <= 4'b0000;
    else
    {ps_n,ps_z,ps_c,ps_v} <= {ns_n,ns_z,ns_c,ns_v};
    end
//////////////////////////////////
//  Interrupt Flag Register //
//////////////////////////////////
reg ps_intr;
reg ns_intr;

always@(posedge sys_clk , posedge reset)
    begin
    if(reset)
    ps_intr <= 1'b0;
    else
    ps_intr <= ns_intr;
    end
////////////////////////////////////////////////
//              MIPS CONTROL UNIT                    //
//          MOORE FINITE STATE MACHINE               //
////////////////////////////////////////////////
```

```
always@(posedge sys_clk, posedge reset)

    if(reset)
    begin
  @(negedge sys_clk)
   begin
    // RTL: PC <-- 32'h0,  ALU_Out <-- 32'h3FC,  int_ack <-- 0
    {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
    {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
    {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h15;
    {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                       int_ack = 1'b0;
    S_Sel = 1'b0; D_OUT_Sel = 2'b00;
    ld_Flags = 1'b0;
    #1 {ns_n,ns_z,ns_c,ns_v} <= 4'b0000;
    ns_intr <= 1'b0;
    state = RESET;
end
    end

  else
  case (state)
            ////////////////////////////////////////////////////////////////////////
            //                   RESET STATE
//
            ////////////////////////////////////////////////////////////////////////
            FETCH:
            if( int_ack == 1'b0 & intr == 1'b1 & ps_intr == 1'b1)
             begin // ** Interrupt Recieved; Head to ISR ***
            @(negedge sys_clk)
             begin// control word for deasserting everything
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h00;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                       int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_c,ps_v};
            ns_intr <= ns_intr;
            state = INTR_1;
            end


             end// end if
            else
             begin
             if( (int_ack == 1'b1 & intr == 1'b1) | intr == 1'b0 | ps_intr == 1'b0)
             begin
             int_ack = 1'b0;
             // RTL: IR<-- IM[PC]; PC+4
             @(negedge sys_clk)
             begin
             {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_1_1; {IO_wr,IO_rd} = 2'b0_0;
             {im_cs, im_rd, im_wr} = 3'b1_1_0; D_In_Sel = 1'b0;
             {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h00;
             {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                       int_ack = 1'b0;
             S_Sel = 1'b0; D_OUT_Sel = 2'b00;
             ld_Flags = 1'b0;
             #1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_c,ps_v};
             ns_intr <= ns_intr;
             state = DECODE;
             end
             end//end if
             end
            ////////////////////////////////////////////////////////////////////////
            //                   RESET STATE
//
            ////////////////////////////////////////////////////////////////////////
            RESET:
            // RTL:  $sp <-- ALU_OUT(32'h3FC)
            begin
            @(negedge sys_clk)
```

```
                begin
                {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
                {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
                {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b1_11_0_0_010;    FS = 5'h00;
                {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
                S_Sel = 1'b0; D_OUT_Sel = 2'b00;
                ld_Flags = 1'b0;
                #1 {ns_n,ns_z,ns_c,ns_v} <= 4'b0000;
                ns_intr <= 1'b0;
                state = FETCH;
                end
                end

                ////////////////////////////////////////////////////////////////////////
                //                            DECODE STATE
//
                ////////////////////////////////////////////////////////////////////////
                DECODE:
                begin
                @(negedge sys_clk)
                begin
                //if(OPCODE == R-Type Instr)
                if(IR[31:26] ==  R_Type)
                begin// It is an R-Type Instruction
                // RTL: RS <-- $rs , RT <-- $rt
                {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
                {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
                {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h00;
                {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
                S_Sel = 1'b0; D_OUT_Sel = 2'b00;
                ld_Flags = 1'b0;
                #1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_c,ps_v};
                ns_intr <= ns_intr;
                case (IR[5:0])//case (funct)
                6'h0D : state = BREAK;
                6'h20 : state = ADD;
                6'h21 : state = ADDU;
                6'h22 : state = SUB;
                6'h23 : state = SUBU;
                6'h02 : state = SRL;
                6'h03 : state = SRA;
                6'h2A : state = SLT;
                6'h00 : state = SLL;
                6'h10 : state = MFHI;
                6'h12 : state = MFLO;
                6'h18 : state = MULT;
                6'h1a : state = DIV;
                6'h25 : state = OR;
                6'h24 : state = AND;
                6'h26 : state = XOR;
                6'h27 : state = NOR;
                6'h2b : state = SLTU;
                6'h08 : state = JR_CALC;
                6'h1F : state = SETIE;
                default: state =  ILLEGAL_OP;
                endcase
                end// end for if(OPCODE == R-Type)

                else if(IR[31:26] == E_KEY)
                begin// its an enhanced type
                {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
                {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
                {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h00;
                {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
                S_Sel = 1'b0; D_OUT_Sel = 2'b00;
                ld_Flags = 1'b0;
                #1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_c,ps_v};
                ns_intr <= ns_intr;

                case (IR[25:21])
                5'h00: state = BLTZ;  //done(?)
```

```
5'h01: state = BGEZ;  //done(?)
5'h03: state = NO_OP; //done{?}
5'h04: state = BSLR;     //done barrell shft left (with rotate)
5'h06: state = BSRR;  //done barrell shift right (with rotate)
5'h09: state = JC;//new
5'h0A: state = JZ;//new
5'h0B: state = JN;//new
5'h0C: state = JV;//new
endcase
end

else // Its a I\J-Type Instruction
begin
{pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
{im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
{D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_1_0_000;    FS = 5'h00;
{dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
S_Sel = 1'b0; D_OUT_Sel = 2'b00;
ld_Flags = 1'b0;
#1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_c,ps_v};
ns_intr <= ns_intr;
case (IR[31:26])//case (funct)
6'h0D : state = ORI;
6'h0F : state = LUI;
6'h08 : state = ADDI;
6'h0A : state = SLTI;
6'h23 : state = LW_CALC;
6'h2B : state = SW;
6'h02 : state = JUMP;
6'h03 : state = JAL;
6'h04 : begin state = BEQ; T_Sel = 0; end
6'h05 : begin state = BNE; T_Sel = 0; end
6'h06 : state = BLEZ;
6'h07 : state = BGTZ;
6'h0B : state = SLTIU;
6'h0C : state = ANDI;
6'h0E : state = XORI;
6'h1C : state = INPUT_CALC;
6'h1D : state = OUTPUT;
6'h1E : state = RETI;
default: state =  ILLEGAL_OP;
endcase
end//end for else for I and J Type Instructions
end//end for @(negedge sys_clk)
end//end for DECODE state
////////////////////////////////////////////////////////////////////////
//      JC                                        Jump Carry                  //
////////////////////////////////////////////////////////////////////////
JC:
// RTL: PC <-- {PC+4[31:28],Address,7'b0000000}
begin
@(negedge sys_clk)
if(ps_c)
begin
{pc_sel, pc_ld, pc_inc, ir_ld} = 5'b11_1_0_0; {IO_wr,IO_rd} = 2'b0_0;
{im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
{D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h00;
{dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
S_Sel = 1'b0; D_OUT_Sel = 2'b00;
ld_Flags = 1'b0;
#1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_z,ps_v};
ns_intr <= ps_intr;
state = FETCH;
end
else
begin
{pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
{im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
{D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h00;
{dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
S_Sel = 1'b0; D_OUT_Sel = 2'b00;
```

```
ld_Flags = 1'b0;
#1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_z,ps_v};
ns_intr <= ps_intr;
state = FETCH;
end
end// end for JC state
////////////////////////////////////////////////////////////////////////
//      JZ                                              Jump Zero                   //
////////////////////////////////////////////////////////////////////////
JZ:
// RTL: PC <-- {PC+4[31:28],Address,7'b0000000}
begin
@(negedge sys_clk)
if(ps_z)
begin
{pc_sel, pc_ld, pc_inc, ir_ld} = 5'b11_1_0_0; {IO_wr,IO_rd} = 2'b0_0;
{im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
{D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h00;
{dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
S_Sel = 1'b0; D_OUT_Sel = 2'b00;
ld_Flags = 1'b0;
#1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_z,ps_v};
ns_intr <= ps_intr;
state = FETCH;
end
else
begin
{pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
{im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
{D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h00;
{dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
S_Sel = 1'b0; D_OUT_Sel = 2'b00;
ld_Flags = 1'b0;
#1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_z,ps_v};
ns_intr <= ps_intr;
state = FETCH;
end
end// end for JZ state
////////////////////////////////////////////////////////////////////////
//      JN                                              Jump Negative               //
////////////////////////////////////////////////////////////////////////
JN:
// RTL: PC <-- {PC+4[31:28],Address,7'b0000000}
begin
@(negedge sys_clk)
if(ps_n)
begin
{pc_sel, pc_ld, pc_inc, ir_ld} = 5'b11_1_0_0; {IO_wr,IO_rd} = 2'b0_0;
{im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
{D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h00;
{dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
S_Sel = 1'b0; D_OUT_Sel = 2'b00;
ld_Flags = 1'b0;
#1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_z,ps_v};
ns_intr <= ps_intr;
state = FETCH;
end
else
begin
{pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
{im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
{D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h00;
{dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
S_Sel = 1'b0; D_OUT_Sel = 2'b00;
ld_Flags = 1'b0;
#1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_z,ps_v};
ns_intr <= ps_intr;
state = FETCH;
end
end// end for JN state
////////////////////////////////////////////////////////////////////////
```

```
//      JV                                              Jump Overflow           //
////////////////////////////////////////////////////////////////////////////////
JV:
// RTL: PC <-- {PC+4[31:28],Address,7'b0000000}
begin
@(negedge sys_clk)
if(ps_v)
begin
{pc_sel, pc_ld, pc_inc, ir_ld} = 5'b11_1_0_0; {IO_wr,IO_rd} = 2'b0_0;
{im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
{D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h00;
{dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
S_Sel = 1'b0; D_OUT_Sel = 2'b00;
ld_Flags = 1'b0;
#1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_z,ps_v};
ns_intr <= ps_intr;
state = FETCH;
end
else
begin
{pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
{im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
{D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h00;
{dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
S_Sel = 1'b0; D_OUT_Sel = 2'b00;
ld_Flags = 1'b0;
#1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_z,ps_v};
ns_intr <= ps_intr;
state = FETCH;
end
end// end for JV state
////////////////////////////////////////////////////////////////////////////////
//      BGEZ           part 1/2   Branch if greater than or equal to zero    //
////////////////////////////////////////////////////////////////////////////////
BGEZ :
// RTL: ALU_OUT <-- $rt
begin
@(negedge sys_clk)
begin
{pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
{im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
{D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h01;
{dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
S_Sel = 1'b0; D_OUT_Sel = 2'b00;
ld_Flags = 1'b0;
#1 {ns_n,ns_z,ns_c,ns_v} <= {N,Z,C,V};// new
ns_intr <= ps_intr;
state = BGEZ_WR;
end
end// end for BGEZ state
////////////////////////////////////////////////////////////////////////////////
//                                BGEZ     part 2/2
//
////////////////////////////////////////////////////////////////////////////////
BGEZ_WR:
// RTL: if(rs => 0) PCP4 + Sign Extention IR[15:0] << 2
//        else
//               PC <- PC + 4
begin
    @(negedge sys_clk)
    begin
    if(ps_n == 1'b0 | ps_z == 1)
    begin
    {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_1_0_0; {IO_wr,IO_rd} = 2'b0_0;
    {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
    {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h00;
    {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
    S_Sel = 1'b0; D_OUT_Sel = 2'b00;
    ld_Flags = 1'b0;
    #1 {ns_n,ns_z,ns_c,ns_v} = {ps_n,ps_z,ps_c,ps_v};
    ns_intr <= ns_intr;
```

```
            state = FETCH;
            end
            else
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h00;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                        int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} = {ps_n,ps_z,ps_c,ps_v};
            ns_intr <= ns_intr;
            state = FETCH;
            end
            end
         end// end for BGEZ_WR state
        ////////////////////////////////////////////////////////////////////////
        //      BLTZ           part 1/2             Branch if less than zero         //
        ////////////////////////////////////////////////////////////////////////
        BLTZ :
        // RTL: ALU_OUT <-- $rt
        begin
        @(negedge sys_clk)
        begin
        {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
        {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
        {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h01;
        {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                        int_ack = 1'b0;
        S_Sel = 1'b0; D_OUT_Sel = 2'b00;
        ld_Flags = 1'b0;
        #1 {ns_n,ns_z,ns_c,ns_v} <= {N,Z,C,V};// new
        ns_intr <= ps_intr;
        state = BLTZ_WR;
        end
        end// end for BLTZ state
        ////////////////////////////////////////////////////////////////////////
        //                              BLTZ     part 2/2
//
        ////////////////////////////////////////////////////////////////////////
        BLTZ_WR:
        // RTL: if(rs < 0) PCP4 + Sign_Extention_IR[15:0] << 2
        //        else
        //               PC <- PC + 4
        begin
            @(negedge sys_clk)
            begin
            if(ps_n == 1'b1)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_1_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h00;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                        int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} = {ps_n,ps_z,ps_c,ps_v};
            ns_intr <= ns_intr;
            state = FETCH;
            end
            else
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h00;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                        int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} = {ps_n,ps_z,ps_c,ps_v};
            ns_intr <= ns_intr;
            state = FETCH;
            end
            end
```

```
 end// end for BLTZ_WR state
////////////////////////////////////////////////////////////////////////
//      NO_OP                                   No Operation                //
////////////////////////////////////////////////////////////////////////
NO_OP :
// Occupies 3 clock ticks
begin
@(negedge sys_clk)
begin
{pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
{im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
{D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h00;
{dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                       int_ack = 1'b0;
S_Sel = 1'b0; D_OUT_Sel = 2'b00;
ld_Flags = 1'b0;
#1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_c,ps_v};// new
ns_intr <= ps_intr;
state = WB_alu;
end
end// end for BSL state
////////////////////////////////////////////////////////////////////////
//      BSLR                                    Barrell Shift Left (w/rotate)     //
////////////////////////////////////////////////////////////////////////
BSLR :
// RTL: ALU OUT <-- $rt<<shamt
begin
@(negedge sys_clk)
begin
{pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
{im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
{D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h1A;
{dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                       int_ack = 1'b0;
S_Sel = 1'b0; D_OUT_Sel = 2'b00;
ld_Flags = 1'b0;
#1 {ns_n,ns_z,ns_c,ns_v} <= {N,Z,C,V};// new
ns_intr <= ps_intr;
state = WB_alu;
end
end// end for BSL state
////////////////////////////////////////////////////////////////////////
//      BSR                                         Barrell Shift Right      //
////////////////////////////////////////////////////////////////////////
BSRR :
// RTL: ALU_OUT <-- $rt>>shamt
begin
@(negedge sys_clk)
begin
{pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
{im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
{D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h1B;
{dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                       int_ack = 1'b0;
S_Sel = 1'b0; D_OUT_Sel = 2'b00;
ld_Flags = 1'b0;
#1 {ns_n,ns_z,ns_c,ns_v} <= {N,Z,C,V};// new
ns_intr <= ps_intr;
state = WB_alu;
end
end// end for BSL state
////////////////////////////////////////////////////////////////////////
//                                 ADD
//
////////////////////////////////////////////////////////////////////////
ADD :
// RTL: ALU OUT <-- $rs + $rt
begin
@(negedge sys_clk)
begin
{pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
{im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
{D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h02;
{dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                       int_ack = 1'b0;
```

```
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {N,Z,C,V};// new
            ns_intr <= ps_intr;
            state = WB_alu;
            end
            end// end for ADD state
            ////////////////////////////////////////////////////////////////////////
            //                                ADDU
//
            ////////////////////////////////////////////////////////////////////////
            ADDU :
            // RTL: ALU_OUT <-- $rs + $rt
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h03;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {N,Z,C,V};// new
            ns_intr <= ps_intr;
            state = WB_alu;
            end
            end// end for ADDU state
            ////////////////////////////////////////////////////////////////////////
            //                                SUB
//
            ////////////////////////////////////////////////////////////////////////
            SUB :
            // RTL: ALU OUT <-- $rs - $rt
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h04;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {N,Z,C,V};// new
            ns_intr <= ns_intr;
            state = WB_alu;
            end
            end// end for SUB state
            ////////////////////////////////////////////////////////////////////////
            //                                SUBU
//
            ////////////////////////////////////////////////////////////////////////
            SUBU :
            // RTL: ALU_OUT <-- $rs - $rt
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h05;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {N,Z,C,V};// new
            ns_intr <= ns_intr;
            state = WB_alu;
            end
            end// end for SUBU state
            ////////////////////////////////////////////////////////////////////////
            //                                SRL
//
            ////////////////////////////////////////////////////////////////////////
```

```
            SRL :
            // RTL: ALU OUT <-- $rt >> 1
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h0D;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {N,Z,C,V};
            ns_intr <= ns_intr;
            state = WB_alu;
            end
            end// end for SRL
            ////////////////////////////////////////////////////////////////////////
            //                                SRA
//
            ////////////////////////////////////////////////////////////////////////
            SRA :
            // RTL: ALU OUT <-- $rt >> 1
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h0E;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {N,Z,C,V};
            ns_intr <= ns_intr;
            state = WB_alu;
            end
            end// end for SRA
            ////////////////////////////////////////////////////////////////////////
            //                                SLL
//
            ////////////////////////////////////////////////////////////////////////
            SLL :
            // RTL: ALU_OUT <-- $rt << 1
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h0C;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {N,Z,C,V};
            ns_intr <= ns_intr;
            state = WB_alu;
            end
            end// end for SLL
            ////////////////////////////////////////////////////////////////////////
            //                                SLT
//
            ////////////////////////////////////////////////////////////////////////
            SLT :
            //                                    true     false
            // RTL: ALU_OUT <-- ($rs<$rt)? 1    :   0
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h06;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
```

```
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {N,Z,C,V};
            ns_intr <= ns_intr;
            state = WB_alu;
            end
            end// end for SLT
            ////////////////////////////////////////////////////////////////////////
            //                                MULT
//
            ////////////////////////////////////////////////////////////////////////
            MULT :
            // RTL: {Y_LO,Y_HI} <-- $rs * $rt
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_1_000;    FS = 5'h1E;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {N,Z,C,V};
            ns_intr <= ns_intr;
            state = FETCH;
            end
            end// end for MULT state
            ////////////////////////////////////////////////////////////////////////
            //                                DIV
//
            ////////////////////////////////////////////////////////////////////////
            DIV :
            // RTL: {Y_LO,Y_HI} <-- $rs / $rt
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_1_000;    FS = 5'h1F;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {N,Z,C,V};
            ns_intr <= ns_intr;
            state = FETCH;
            end
            end// end for DIV state
            ////////////////////////////////////////////////////////////////////////
            //                                OR
//
            ////////////////////////////////////////////////////////////////////////
            OR :
            // RTL: ALU OUT <-- $rs | $rt
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h09;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} = {N,Z,C,V};
            ns_intr = ns_intr;
            state = WB_alu;
            end
            end// end for OR state
            ////////////////////////////////////////////////////////////////////////
            //                                NOR
//
            ////////////////////////////////////////////////////////////////////////
            NOR :
```

```
            // RTL: ALU_OUT <-- ~($rs | $rt)
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h0B;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} = {N,Z,C,V};
            ns_intr = ns_intr;
            state = WB_alu;
            end
            end// end for NOR state
            ////////////////////////////////////////////////////////////////////////
            //                                XOR
//
            ////////////////////////////////////////////////////////////////////////
            XOR :
            // RTL: ALU_OUT <-- $rs ^ $rt
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h0A;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {N,Z,C,V};
            ns_intr <= ns_intr;
            state = WB_alu;
            end
            end// end for XOR state
            ////////////////////////////////////////////////////////////////////////
            //                            SLTU  (unsigned values)                                  //
            ////////////////////////////////////////////////////////////////////////
            SLTU :
            //                                      true     false
            // RTL: ALU_OUT <-- ($rs<$rt)? 1    :   0
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h07;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
           ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {N,Z,C,V};
            ns_intr <= ns_intr;
            state = WB_alu;
            end
            end// end for SLTU state
            ////////////////////////////////////////////////////////////////////////
            //                                AND
//
            ////////////////////////////////////////////////////////////////////////
            AND :
            // RTL: ALU_OUT <-- $rs & $rt
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h08;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {N,Z,C,V};
```

```
            ns_intr <= ns_intr;
            state = WB_alu;
            end
            end// end for AND state
            ////////////////////////////////////////////////////////////////////////
            //                                ORI
//
            ////////////////////////////////////////////////////////////////////////
            ORI :
            // RTL: ALU OUT <-- $rs | $rt(se 16)
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h17;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {N,Z,C,V};
            ns_intr <= ns_intr;
            state = WB_imm;
            end
            end// end for ORI state
            ////////////////////////////////////////////////////////////////////////
            //                                XORI
//
            ////////////////////////////////////////////////////////////////////////
            XORI :
            // RTL: ALU_OUT <-- $rs ^ $rt(se_16)
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h18;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {N,Z,C,V};
            ns_intr <= ns_intr;
            state = WB_imm;
            end
            end// end for XORI state
            ////////////////////////////////////////////////////////////////////////
            //                                ANDI
//
            ////////////////////////////////////////////////////////////////////////
            ANDI :
            // RTL: ALU OUT <-- $rs ^ $rt(se 16)
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h16;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {N,Z,C,V};
            ns_intr <= ns_intr;
            state = WB_imm;
            end
            end// end for ANDI state
            ////////////////////////////////////////////////////////////////////////
            //                                SLTIU        (unsigned)
//
            ////////////////////////////////////////////////////////////////////////
            SLTIU :
            //                                                    true     false
            // RTL: ALU_OUT <-- if ( $rs<$rt(se_16) )?  1    :   0
```

```
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h07;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {N,Z,C,V};
            ns_intr <= ns_intr;
            state = WB_imm;
            end
            end// end for SLTIU state
            ////////////////////////////////////////////////////////////////////////
            //                                LUI
//
            ////////////////////////////////////////////////////////////////////////
            LUI :
            // RTL: ALU OUT <-- { RT [15:0] , 16'h0}
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h19;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {N,Z,C,V};
            ns_intr <= ns_intr;
            state = WB_imm;
            end
            end// end for LUI state
            ////////////////////////////////////////////////////////////////////////
            //                                ADDI
//
            ////////////////////////////////////////////////////////////////////////
            ADDI :
            // RTL: ALU_OUT <-- $rs + $rt(se_16)
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h02;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {N,Z,C,V};
            ns_intr <= ns_intr;
            state = WB_imm;
            end
            end// end for ADDI state
            ////////////////////////////////////////////////////////////////////////
            //                                SLTI
//
            ////////////////////////////////////////////////////////////////////////
            SLTI :
            // RTL: ALU_OUT <-- $rs + $rt(se_16)
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h06;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {N,Z,C,V};
            ns_intr <= ns_intr;
```

```
            state = WB_imm;
            end
            end// end for SLTI state
            ////////////////////////////////////////////////////////////////////////
            //                                SW
//
            ////////////////////////////////////////////////////////////////////////
            SW :
            // RTL: ALU_OUT <-- $rs + $rt(se_16)
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h02;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                       int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} = {ps_n,ps_z,ps_c,ps_v};
            ns_intr <= ns_intr;
            state = WB_mem;
            end
            end// end for SW state
            ////////////////////////////////////////////////////////////////////////
            //                                OUTPUT
//
            ////////////////////////////////////////////////////////////////////////
            OUTPUT :
            // RTL: ALU_OUT <-- $rs + $rt(se_16)
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h02;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                       int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_n,ps_c,ps_v};
            ns_intr <= ns_intr;
            state = WB_OUTPUT;
            end
            end// end for INPUT state
            ////////////////////////////////////////////////////////////////////////
            //                                WB OUTPUT
//
            ////////////////////////////////////////////////////////////////////////
            WB_OUTPUT :
            // RTL : I/O[ALU_OUT] <-- RT(rt)
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b1_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_010;    FS = 5'h00;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                       int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_c,ps_v};
            ns_intr <= ns_intr;
            state = FETCH;
            end
            end// end for WB_mem state
            ////////////////////////////////////////////////////////////////////////
            //                                INPUT_CALC
//
            ////////////////////////////////////////////////////////////////////////
            INPUT_CALC :
            // RTL: ALU_OUT <-- $rs + $rt(se_16)
            begin
            @(negedge sys_clk)
```

```
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h02;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                           int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_c,ps_v};
            ns_intr <= ns_intr;
            state = INPUT;
            end
            end// end for OUTPUT_CALC state
            ////////////////////////////////////////////////////////////////////
            //                        INPUT (load word from I/O memory)                 //
            ////////////////////////////////////////////////////////////////////
            INPUT:
            // RTL: D_In <-- I_O_Mem[ALU_OUT]
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_1;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b1;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_010;    FS = 5'h00;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                           int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} = {ps_n,ps_z,ps_c,ps_v};
            ns_intr <= ns_intr;
            state = WB_INPUT;
            end
            end// end for INPUT state
            ////////////////////////////////////////////////////////////////////
            //                              WB INPUT
//
            ////////////////////////////////////////////////////////////////////
            WB_INPUT:
            // RTL : R[rt] <-- D_In
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b1_01_0_0_011;    FS = 5'h00;
            {dm_cs, dm_rd, dm_wr} = 3'b1_0_1;                           int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_c,ps_v};
            ns_intr <= ns_intr;
            state = FETCH;
            end
            end// end for WB_OUTPUT state
            ////////////////////////////////////////////////////////////////////
            //                          LW_CALC (Effective Address Calculation)       //
            ////////////////////////////////////////////////////////////////////
            LW_CALC :
            // RTL: ALU OUT <-- $rs + $rt(se 16)
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h02;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                           int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} = {ps_n,ps_z,ps_c,ps_v};
            ns_intr <= ns_intr;
            state = LW;
            end
            end// end for LW_CALC state
```

```
            ////////////////////////////////////////////////////////////////////////
            //                          LW  (load word from memory)                 //
            ////////////////////////////////////////////////////////////////////////
            LW :
            // RTL: D_In <-- dM[ALU_OUT]
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_010;    FS = 5'h02;
            {dm_cs, dm_rd, dm_wr} = 3'b1_1_0;                       int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} = {ps_n,ps_z,ps_c,ps_v};
            ns_intr <= ns_intr;
            state = WB_LW;
            end
            end// end for LW state

            ////////////////////////////////////////////////////////////////////////
            //                              BEQ     part 1/2
//
            ////////////////////////////////////////////////////////////////////////
            BEQ :
            // RTL:  ALU_OUT <-- $rs - $rt
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h04;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                       int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} = {N,Z,C,V};
            ns_intr <= ns_intr;
            state = BEQ_WR;
            end
            end// end for BEQ state
            ////////////////////////////////////////////////////////////////////////
            //                              BEQ     part 2/2
//
            ////////////////////////////////////////////////////////////////////////
            BEQ_WR:
            // RTL: if(RS ==  RT) PCP4 + {SE14IR[15],IR[15:0],2'b00}
            //        else
            //               PC <- PC + 4
            begin
                @(negedge sys_clk)
                begin
                if(ps_z == 1'b1)
                    begin
                {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_1_0_0; {IO_wr,IO_rd} = 2'b0_0;
                {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
                {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h00;
                {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                       int_ack = 1'b0;
                S_Sel = 1'b0; D_OUT_Sel = 2'b00;
                ld_Flags = 1'b0;
                #1 {ns_n,ns_z,ns_c,ns_v} = {ps_n,ps_z,ps_c,ps_v};
                ns_intr <= ns_intr;
                state = FETCH;
                    end
                else
                    begin
                    {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
                    {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
                    {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h00;
                    {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                       int_ack = 1'b0;
                    S_Sel = 1'b0; D_OUT_Sel = 2'b00;
                    ld_Flags = 1'b0;
```

```
            #1 {ns_n,ns_z,ns_c,ns_v} = {ps_n,ps_z,ps_c,ps_v};
            ns_intr <= ns_intr;
            state = FETCH;
            end
        end
     end// end for ADD state
    ////////////////////////////////////////////////////////////////////////
    //                              BNE     part 1/2
//
    ////////////////////////////////////////////////////////////////////////
    BNE :
    // RTL:  ALU_OUT <-- $rs - $rt
    begin
    @(negedge sys_clk)
    begin
    {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
    {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
    {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h04;
    {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                       int_ack = 1'b0;
    S_Sel = 1'b0; D_OUT_Sel = 2'b00;
    ld_Flags = 1'b0;
    #1 {ns_n,ns_z,ns_c,ns_v} = {N,Z,C,V};
    ns_intr <= ns_intr;
    state = BNE_WR;
    end
    end// end for ADD state
    ////////////////////////////////////////////////////////////////////////
    //                              BNE     part 2/2
//
    ////////////////////////////////////////////////////////////////////////
    BNE_WR:
    // RTL: if(RS ==  RT) PCP4 + {SE14IR[15],IR[15:0],2'b00}
    //        else
    //               PC <- PC + 4
    begin
        @(negedge sys_clk)
        begin
        if(ps_z == 1'b0)
            begin
        {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_1_0_0; {IO_wr,IO_rd} = 2'b0_0;
        {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
        {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h00;
        {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                       int_ack = 1'b0;
        S_Sel = 1'b0; D_OUT_Sel = 2'b00;
        ld_Flags = 1'b0;
        #1 {ns_n,ns_z,ns_c,ns_v} = {ps_n,ps_z,ps_c,ps_v};
        ns_intr <= ns_intr;
        state = FETCH;
            end
        else
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h00;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                       int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} = {ps_n,ps_z,ps_c,ps_v};
            ns_intr <= ns_intr;
            state = FETCH;
            end
        end
     end// end for BEQ_WR state
    ////////////////////////////////////////////////////////////////////////
    //                              BLEZ   part 1/2
//
    ////////////////////////////////////////////////////////////////////////
    BLEZ :
    // RTL:  ALU_OUT <-- $rs
    begin
    @(negedge sys_clk)
```

```
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h00;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                       int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} = {N,Z,C,V};
            ns_intr <= ns_intr;
            state = BLEZ_WR;
            end
            end// end for BLEZ state
            ////////////////////////////////////////////////////////////////////////
            //                              BLEZ     part 2/2
//
            ////////////////////////////////////////////////////////////////////////
            BLEZ_WR:
            // RTL: if(rs <= 0) PCP4 + Sign_Extention_IR[15:0] << 2
            //        else
            //               PC <- PC + 4
            begin
                @(negedge sys_clk)
                begin
                if(ps_z == 1'b1 | ps_n == 1'b1)
                begin
                {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_1_0_0; {IO_wr,IO_rd} = 2'b0_0;
                {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
                {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h00;
                {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                       int_ack = 1'b0;
                S_Sel = 1'b0; D_OUT_Sel = 2'b00;
                ld_Flags = 1'b0;
                #1 {ns_n,ns_z,ns_c,ns_v} = {ps_n,ps_z,ps_c,ps_v};
                ns_intr <= ns_intr;
                state = FETCH;
                end
                else
                begin
                {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
                {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
                {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h00;
                {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                       int_ack = 1'b0;
                S_Sel = 1'b0; D_OUT_Sel = 2'b00;
                ld_Flags = 1'b0;
                #1 {ns_n,ns_z,ns_c,ns_v} = {ps_n,ps_z,ps_c,ps_v};
                ns_intr <= ns_intr;
                state = FETCH;
                end
                end
             end// end for BLEZ_WR state
            ////////////////////////////////////////////////////////////////////////
            //                              BGTZ   part 1/2
//
            ////////////////////////////////////////////////////////////////////////
            BGTZ :
            // RTL:  ALU_OUT <-- $rs
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h00;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                       int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} = {N,Z,C,V};
            ns_intr <= ns_intr;
            state = BGTZ_WR;
            end
            end// end for BGTZ state
            ////////////////////////////////////////////////////////////////////////
```

```
            //                              BGTZ     part 2/2
//
            ////////////////////////////////////////////////////////////////////////
            BGTZ_WR:
            // RTL: if(rs > 0) PCP4 + Sign_Extention_IR[15:0] << 2
            //        else
            //               PC <- PC + 4
            begin
                @(negedge sys_clk)
                begin
                if(!(ps_z == 1'b1 | ps_n == 1'b1))
                begin
                {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_1_0_0; {IO_wr,IO_rd} = 2'b0_0;
                {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
                {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h00;
                {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
                S_Sel = 1'b0; D_OUT_Sel = 2'b00;
                ld_Flags = 1'b0;
                #1 {ns_n,ns_z,ns_c,ns_v} = {ps_n,ps_z,ps_c,ps_v};
                ns_intr <= ps_intr;
                state = FETCH;
                end
                else
                begin
                {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
                {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h00;
                {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
                S_Sel = 1'b0; D_OUT_Sel = 2'b00;
                ld_Flags = 1'b0;
                #1 {ns_n,ns_z,ns_c,ns_v} = {ps_n,ps_z,ps_c,ps_v};
                ns_intr <= ps_intr;
                state = FETCH;
                end
                end
             end// end for BGTZ_WR state
            ////////////////////////////////////////////////////////////////////////
            //                              SETIE
//
            ////////////////////////////////////////////////////////////////////////
            SETIE :
            // RTL: interrupt enable
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h00;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_c,ps_v};
           ns_intr <= 1'b1;
            state = FETCH;
            end
            end// end for SETIE state

            ////////////////////////////////////////////////////////////////////////
            //                              RETI
//
            ////////////////////////////////////////////////////////////////////////
            RETI :
            // RTL: RS <-- RF[$sp]
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h00;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
            S_Sel = 1'b1; D_OUT_Sel = 2'b00;
```

```
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_c,ps_v};
           ns_intr <= ps_intr;
            state = RETI_2;
            end
            end// end for RETI state
            ////////////////////////////////////////////////////////////////////////
            //                                RETI 2
//
            ////////////////////////////////////////////////////////////////////////
            RETI_2 :
            // RTL: ALU_OUT <-- RS
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h00;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_c,ps_v};
           ns_intr <= ps_intr;
            state = RETI_3;
            end
            end// end for RETI_2 state
            ////////////////////////////////////////////////////////////////////////
            //                                RETI_3
//
            ////////////////////////////////////////////////////////////////////////
            RETI_3 :
            // RTL:  Present State Flags <--- M{ALU_OUT] , RS <-- Rf[$sp]
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_010;    FS = 5'h00;
            {dm_cs, dm_rd, dm_wr} = 3'b1_1_0;                          int_ack = 1'b0;
            S_Sel = 1'b1; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b1;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {N,Z,C,V};
           ns_intr <= ps_intr;
            state = RETI_4;
            end
            end// end for RETI_3 state
            ////////////////////////////////////////////////////////////////////////
            //                                RETI_3
//
            ////////////////////////////////////////////////////////////////////////
            RETI_4 :
            // RTL:  ALU OUT <-- RS - 4
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h12;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_c,ps_v};
           ns_intr <= ps_intr;
            state = RETI_5;
            end
            end// end for RETI_3 state

            ////////////////////////////////////////////////////////////////////////
            //                                RETI_4
//
            ////////////////////////////////////////////////////////////////////////
```

```
            RETI_5 :
            // RTL:  D In <-- M[ALU OUT]  ,    Rd[$sp] <-- ALU OUT
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b1_11_0_0_010;    FS = 5'h00;
            {dm_cs, dm_rd, dm_wr} = 3'b1_1_0;                          int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_c,ps_v};
           ns_intr <= ps_intr;
            state = RETI_6;
            end
            end// end for RETI_4 state
            ////////////////////////////////////////////////////////////////////////
            //                              RETI_5
//
            ////////////////////////////////////////////////////////////////////////
            RETI_6 :
            // RTL: RS <-- RF[$sp] , PC <-- D In
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b10_1_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_011;    FS = 5'h00;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
            S_Sel = 1'b1; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_c,ps_v};
           ns_intr <= ps_intr;
            state = RETI_7;
            end
            end// end for RETI_5 state
            ////////////////////////////////////////////////////////////////////////
            //                              RETI_6
//
            ////////////////////////////////////////////////////////////////////////
            RETI_7 :
            // RTL: ALU_OUT <-- RS - 4
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h12;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_c,ps_v};
           ns_intr <= ps_intr;
            state = RETI_8;
            end
            end// end for RETI_6 state
            ////////////////////////////////////////////////////////////////////////
            //                              RETI_7
//
            ////////////////////////////////////////////////////////////////////////
            RETI_8:
            // RTL: Rd[$sp] <-- ALU OUT
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b1_11_0_0_010;    FS = 5'h00;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
```

```
            #1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_c,ps_v};
           ns_intr <= ps_intr;
            state = FETCH;
            end
            end// end for RETI_7 state

            ////////////////////////////////////////////////////////////////////////
            //                          JUMP
//
            ////////////////////////////////////////////////////////////////////////
            JUMP:
            // RTL: PC <-- {PC+4[31:28],Address,2'b00}
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b01_1_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h00;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_z,ps_v};
            ns_intr <= ps_intr;
            state = FETCH;
            end
            end// end for JUMP state
            ////////////////////////////////////////////////////////////////////////
            //                                JR_CALC
//
            ////////////////////////////////////////////////////////////////////////
            JR_CALC :
            // RTL: ALU_OUT <-- R[rs]
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h00;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_c,ps_v};
            ns_intr <= ns_intr;
            state = JR;
            end
            end// end for JR_CALC
            ////////////////////////////////////////////////////////////////////////
            //                                JR
//
            ////////////////////////////////////////////////////////////////////////
            JR :
            // RTL: PC <-- ALU OUT[rs]
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b10_1_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_010;    FS = 5'h00;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_c,ps_v};
            ns_intr <= ns_intr;
            state = FETCH;
            end
            end// end for JR
            ////////////////////////////////////////////////////////////////////////
            //                                JAL
//
            ////////////////////////////////////////////////////////////////////////
            JAL :
```

```
            // RTL: PC <-- {PC+4[31:28],Address,2'b00}
            //        R[$ra] <-- PC
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b01_1_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b1_10_0_0_100;    FS = 5'h00;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                       int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_c,ps_v};
            ns_intr <= ns_intr;
            state = FETCH;
            end
            end// end for JAL
            ////////////////////////////////////////////////////////////////////////
            //                              BREAK                                 NOT DONE
//
            ////////////////////////////////////////////////////////////////////////
            BREAK :
            begin
            @(negedge sys_clk)
            begin
            $display("BREAK INSTRUCTION FETCH %t",$time);
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h00;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                       int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_c,ps_v};
            ns_intr <= ns_intr;
            $display(" R E G I S T E R ' S  A F T E R  B R E A K");
            $display(" ");
            Dump_Registers32;
            $display(" ");
            Dump_Memory;
            Dump_I_O_Memory;
            $finish;
            end
            end
            ////////////////////////////////////////////////////////////////////////
            //                        ILLEGAL OP                                    NOT DONE  //
            ////////////////////////////////////////////////////////////////////////
            ILLEGAL_OP :
            begin
            @(negedge sys_clk)
            begin
            $display("ILLEGAL OPCODE FETCHED %t",$time);
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_000;    FS = 5'h00;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                       int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_c,ps_v};
            ns_intr <= ns_intr;
            Dump_Registers32; //task to output MIPS RegFile
            Dump_PC_and_IR; // task to output current values of Program Counter
                            // and Instruction Register
            $finish;
            end
            end


            ////////////////////////////////////////////////////////////////////////
            //                        INTERRUPT 1                                   NOT DONE  //
            ////////////////////////////////////////////////////////////////////////
            INTR_1 :
            // PC  gets address of interrupt vector ; Save PC in $ra
```

```
            // RTL : R[$ra] <-- PC , ALU_OUT <-- 0x3FC
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b1_10_0_0_100;    FS = 5'h15;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                       int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_c,ps_v};
            interrupt_enable <= interrupt_enable;
            state = INTR_2;
            end
            end
            ////////////////////////////////////////////////////////////////////////
            //                          INTERRUPT 2                                 NOT DONE  //
            ////////////////////////////////////////////////////////////////////////
            INTR_2 :
            // RTL : D in <-- dM[ALU OUT(0x3FC)]
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_010;    FS = 5'h00;
            {dm_cs, dm_rd, dm_wr} = 3'b1_1_0;                       int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_c,ps_v};
            interrupt_enable <= interrupt_enable;
            state = INTR_3;
            end
            end
            ////////////////////////////////////////////////////////////////////////
            //                          INTERRUPT 3                                 NOT DONE  //
            ////////////////////////////////////////////////////////////////////////
            INTR_3 :
            // Reload PC with address of ISR; ack the intr; goto fetch
            // control word assignments for PC <-- D_in(dm[0x3FC]); int_ack <-- 1
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b10_1_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_011;    FS = 5'h00;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                       int_ack = 1'b1;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_c,ps_v};
            interrupt_enable <= interrupt_enable;
            state = FETCH;
            end
            end
/*////////////////////////////////INTR FOR MEM 14//////////////////////////////
            ////////////////////////////////////////////////////////////////////////
            //                          INTERRUPT 1                                 //
            ////////////////////////////////////////////////////////////////////////
            INTR_1 :
            // RTL : ALU_OUT <-- 0x3FC
            begin
            @(negedge sys clk)
            begin
            {pc sel, pc ld, pc inc, ir ld} = 5'b00 0 0 0; {IO wr,IO rd} = 2'b0 0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_010;    FS = 5'h15;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                       int_ack = 1'b0;
            S Sel = 1'b0; D OUT Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns n,ns z,ns c,ns v} <= {ps n,ps z,ps c,ps v};
            ns_intr <= ns_intr;
```

```
state = INTR_2;
end
end
////////////////////////////////////////////////////////////////////////
//                        INTERRUPT 2                                 //
////////////////////////////////////////////////////////////////////////
INTR_2 :
// RTL: D In <--M[ALU OUT(0x3FC)], RS <-- RF[$sp]
begin
@(negedge sys clk)
begin
{pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
{im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
{D En, DA Sel, T Sel, HILO ld, Y Sel} = 8'b0 00 0 0 010;    FS = 5'h00;
{dm_cs, dm_rd, dm_wr} = 3'b1_1_0;                       int_ack = 1'b0;
S Sel = 1'b1; D OUT Sel = 2'b00;
ld_Flags = 1'b0;
#1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_c,ps_v};
ns_intr <= ns_intr;
state = INTR 3;
end
end
////////////////////////////////////////////////////////////////////////
//                        INTERRUPT 3                                 //
////////////////////////////////////////////////////////////////////////
INTR_3 :
// RTL: PC<-- D In  , ALU OUT <-- RS + 4
begin
@(negedge sys_clk)
begin
{pc_sel, pc_ld, pc_inc, ir_ld} = 5'b10_1_0_0; {IO_wr,IO_rd} = 2'b0_0;
{im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
{D En, DA Sel, T Sel, HILO ld, Y Sel} = 8'b0 00 0 0 011;    FS = 5'h11;
{dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                       int_ack = 1'b0;
S_Sel = 1'b1; D_OUT_Sel = 2'b00;
ld_Flags = 1'b0;
#1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_c,ps_v};
ns_intr <= ns_intr;
state = INTR 4;
end
end
////////////////////////////////////////////////////////////////////////
//                        INTERRUPT 4                                 //
////////////////////////////////////////////////////////////////////////
INTR 4 :
// RTL: Rd[$sp] <-- ALU_OUT ,  M[ALU_OUT(0x3FC)] <-- PC
begin
@(negedge sys_clk)
begin
{pc sel, pc ld, pc inc, ir ld} = 5'b00 0 0 0; {IO wr,IO rd} = 2'b0 0;
{im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
{D En, DA Sel, T Sel, HILO ld, Y Sel} = 8'b1 11 0 0 010;    FS = 5'h00;
{dm_cs, dm_rd, dm_wr} = 3'b1_0_1;                       int_ack = 1'b0;
S_Sel = 1'b0; D_OUT_Sel = 2'b01;
ld_Flags = 1'b0;
#1 {ns n,ns z,ns c,ns v} <= {ps n,ps z,ps c,ps v};
ns intr <= ns intr;
state = INTR_5;
end
end

////////////////////////////////////////////////////////////////////////
//                        INTERRUPT 5                                 //
////////////////////////////////////////////////////////////////////////
INTR_5 :
// RTL: RS <-- RF[$sp]
begin
@(negedge sys clk)
begin
{pc sel, pc ld, pc inc, ir ld} = 5'b00 0 0 0; {IO wr,IO rd} = 2'b0 0;
{im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
```

```
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_010;    FS = 5'h00;
            {dm_cs, dm_rd, dm_wr} = 3'b0 0 0;                       int_ack = 1'b0;
            S_Sel = 1'b1; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_c,ps_v};
            ns_intr <= ns_intr;
            state = INTR_6;
            end
            end
           ////////////////////////////////////////////////////////////////////////
            //                        INTERRUPT 6                                 //
            ////////////////////////////////////////////////////////////////////////
            INTR_6 :
            // RTL: ALU OUT <-- RS + 4
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D En, DA Sel, T Sel, HILO ld, Y Sel} = 8'b0 00 0 0 010;    FS = 5'h11;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                       int_ack = 1'b0;
            S Sel = 1'b0; D OUT Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_c,ps_v};
            ns intr <= ns intr;
            state = INTR_7;
            end
            end
            ////////////////////////////////////////////////////////////////////////
            //                        INTERRUPT 7                                 //
            ////////////////////////////////////////////////////////////////////////
            INTR_7 :
            // RTL: M[ALU OUT] <-- {N,Z,C,V}, Rd[$sp] <-- ALU OUT
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D En, DA Sel, T Sel, HILO ld, Y Sel} = 8'b1 11 0 0 010;    FS = 5'h00;
            {dm_cs, dm_rd, dm_wr} = 3'b1_0_1;                       int_ack = 1'b1;
            S_Sel = 1'b0; D_OUT_Sel = 2'b10;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_c,ps_v};
            ns_intr <= ns_intr;
            state = FETCH;
            end
            end
////////////////////////////////////////////////////////////////////////////*/
            ////////////////////////////////////////////////////////////////////////
            //                              MFHI
//
            ////////////////////////////////////////////////////////////////////////
            MFHI :
            // RTL : R[rd] <-- Y_HI
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b1_00_0_0_000;    FS = 5'h00;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                       int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_c,ps_v};
            ns_intr <= ns_intr;
            state = FETCH;
            end
            end// end of MFHI state
            ////////////////////////////////////////////////////////////////////////
            //                              MFLO
//
```

```
            ////////////////////////////////////////////////////////////////////////
            MFLO :
            // RTL : R[rd] <-- Y_LO
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b1_00_0_0_001;    FS = 5'h00;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                        int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_c,ps_v};
            ns_intr <= ns_intr;
            state = FETCH;
            end
            end// end of MFLO state
            ////////////////////////////////////////////////////////////////////////
            //                                WB_alu
//
            ////////////////////////////////////////////////////////////////////////
            WB_alu :
            // RTL : R[rd] <-- ALU_OUT
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b1_00_0_0_010;    FS = 5'h00;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                        int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_c,ps_v};
            ns_intr <= ns_intr;
            state = FETCH;
            end
            end// end for WB_alu state
            ////////////////////////////////////////////////////////////////////////
            //                                WB imm
//
            ////////////////////////////////////////////////////////////////////////
            WB_imm :
            // RTL : R[rt] <-- ALU_OUT
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b1_01_0_0_010;    FS = 5'h00;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                        int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_c,ps_v};
           ns_intr <= ns_intr;
            state = FETCH;
            end
            end// end for WB_imm state
            ////////////////////////////////////////////////////////////////////////
            //                                WB_mem
//
            ////////////////////////////////////////////////////////////////////////
            WB_mem :
            // RTL : M[ALU_OUT(rs+se_16)] <-- RT(rt)
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b0_00_0_0_010;    FS = 5'h00;
            {dm_cs, dm_rd, dm_wr} = 3'b1_0_1;                        int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
```

```
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} = {ps_n,ps_z,ps_c,ps_v};
            ns_intr <= ns_intr;
            state = FETCH;
            end
            end// end for WB_mem state
            ////////////////////////////////////////////////////////////////////////
            //                                WB LW
//
            ////////////////////////////////////////////////////////////////////////
            WB_LW :
            // RTL : R[rt] <-- D_In
            begin
            @(negedge sys_clk)
            begin
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0; {IO_wr,IO_rd} = 2'b0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0; D_In_Sel = 1'b0;
            {D_En, DA_Sel, T_Sel, HILO_ld, Y_Sel} = 8'b1_01_0_0_011;    FS = 5'h00;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;                          int_ack = 1'b0;
            S_Sel = 1'b0; D_OUT_Sel = 2'b00;
            ld_Flags = 1'b0;
            #1 {ns_n,ns_z,ns_c,ns_v} <= {ps_n,ps_z,ps_c,ps_v};
            ns_intr <= ns_intr;
            state = FETCH;
            end
            end// end for WB_mem state
            endcase



            task Dump_Registers16;// Display registers  0 to 15
            begin
            for ( i = 0 ; i < 16 ; i = i +1)
                begin
                 $display("time = %t, Register %h = %h",$time,
                             i, Top_lvlTB.uut.IDP_uut.RF0.Data[i]);
                end//end for
            end
            endtask//end Dump_Register16 task

            task Dump_Registers32;// Display register 0 to 31
            begin
            for ( i = 0 ; i < 32 ; i = i +1)
                begin
                 $display("time = %t, Register %h = %h",$time,i,
                            Top_lvlTB.uut.IDP_uut.RF0.Data[i]);
                end//end Dump_Registers32 for
            end
            endtask//end Dump_Registers32 task

            task Dump_PC_and_IR;// Display contents of PC and IR registers
            begin
            $display("time = %t , PC = %h, IR = %h",$time,
                    Top_lvlTB.uut.IU_uut.PC,Top_lvlTB.uut.IU_uut.IR);
            end
            endtask// end Dump_PCand_IR task
            task Dump_Memory;// Display contents of Memory
            begin// Displaying Memory locations M[C0] to M[FF]
            for ( i =  960; i < 1012 ; i = i +1)
                begin
                 $display("Memory [%h] =  %3'h",i,Top_lvlTB.dm_uut.Mem[i]);
                end//end Dump_Registers32 for
            end
            endtask// end Dump_Memory

            task Dump_I_O_Memory;// Display contents of I_O_Memory
            begin// Displaying Memory locations I_0_M[C0] to I_0_M[FF]
            for ( i = 192 ; i < 256 ; i = i +1)
                begin
                 $display("I_O_Memory [%h] =  %3'h",i,Top_lvlTB.IOMEM.Mem[i]);
```

```
            end//end Dump_Registers32 for
        end
        endtask// end Dump_Memory

endmodule
```

III.B.2 Integer Data path

```
`timescale 1ns / 1ps
/***************************** C E C S  4 4 0 ******************************
 *
 * File Name:  IntegerDataPath_2.v
 * Project:    Intstructon Unit
 * Designer:   Kassandra Flores, Adan Hernandez
 * Email:      kassandra.flores@student.csulb.edu,
 *                  adan.hernandez@student.csulb.edu
 * Rev. No.:   Version 1.0
 * Rev. Date:  10/09/2016
 *
 * Purpose: To create a data path between the 32 registers inside the
 * register file and the ALU. All logic and arithmetic operations can
 * operate on the contents of each register inside the register file aswell
 * with any 32 bit value present at the DT port.The destination
 * location of any ALU operation or data present on the DY port can
 * be selected by the DA_Sel port. 1 Being it is the register
 * specified by the 5 bit D_Addr port or 0 being the 5 bit value specified
 * by the T_Addr port.
 ***************************************************************************/
module IntegerDataPath_2(clk,reset,D_En,D_Addr,T_Addr,DT,T_Sel,C,V,N,Z,DY,PC_in,
                                Y_Sel,D_OUT,S_Addr,FS,HILO_ld,ALU_OUT,DA_Sel,I_O_input,
                                D_In_Sel,S_Sel,D_OUT_Sel,Flags,shamt);
    input clk;
    input reset;

////////////////////////////////
// Register File inputs & wires
////////////////////////////////
    input D_En;//Register File write enable
    input [1:0] DA_Sel;// Destination locaton select
    input [4:0] D_Addr;//Destination Address
    input [4:0] S_Addr;//Selected register Source 1
    input [4:0] T_Addr;//Selected register Source 2
    input [4:0] shamt; //Shift Amount into ALU
    input [3:0] Flags; //Flag input from control unit
    input S_Sel;         //Selects S_addr source //new
    input[1:0]D_OUT_Sel;   //Data out select
    wire [4:0] D_Addr_w;//Selected destination address
    wire [4:0] S_Addr_w;//Selected register source address
    wire [31:0] S;//contents of register source 1
    wire [31:0] T;//contents of register source 2

reg [31:0] RS,RT;

assign D_Addr_w = (DA_Sel == 2'b00)? D_Addr:
                        (DA_Sel == 2'b01)? T_Addr:
                        (DA_Sel == 2'b10)? 5'b11111:// return address
                                           5'b11101;// stack pointer


//////////////////////////////
// ALU input/outputs & wires
//////////////////////////////
    input [4:0] FS; // Function Select
    output C;//carry
    output V;//overflow
    output N;//negative
    output Z;//zero

    wire [31:0] Y_LO;
    wire [31:0] Y_HI;
//////////////////////////////
// Inputs into T_MUX
//////////////////////////////
   input [31:0] DT;
    input T_Sel;
//////////////////////////////
// Input into HI reg
```

```
//////////////////////////////
    input HILO_ld;// Load control of HI and LO regs
//////////////////////////////
// Input into Y MUX
//////////////////////////////
    input [31:0] DY;
    input [31:0] I_O_input; //new
    input [31:0] PC_in;
    input [2:0] Y_Sel;// Output select
//////////////////////////////
// Integer Datapath Outputs
//////////////////////////////
    output [31:0] ALU_OUT;//ALU out
    output [31:0] D_OUT;//Data out
    wire [31:0] ALU_OUT_W;

   assign ALU_OUT_W = ALU_OUT;
//////////////////////////////
//  HI and LO regs
//////////////////////////////
    reg [31:0] HI;
     reg [31:0] LO;
////////////////////////////////////////////////
//          Register File
////////////////////////////////////////////////
Register_File_32x32
    RF0
     (
    .clk(clk),
    .reset(reset),
    .S(S),
    .T(T),
    .D(ALU_OUT_W),
    .S_Addr(S_Addr_w),
    .T_Addr(T_Addr),
    .D_Addr(D_Addr_w),
    .D_En(D_En)
    );

////////////////////////////////////////////////////////
//                T_MUX
////////////////////////////////////////////////////////
wire [31:0] T_Wire;//output control
//                          1     :    0
assign T_Wire = (T_Sel)?  DT :  T;
////////////////////////////////////////////////////////
//                    S_Sel (S_Addr mux)                          //
////////////////////////////////////////////////////////
assign S_Addr_w  = (S_Sel)? 5'b11101 : S_Addr;// stack pointer

////////////////////////////////////////////////////////
//               D In MUX                              //
////////////////////////////////////////////////////////
input D_In_Sel;
wire [31:0] Data_in;//new
assign Data_in = (D_In_Sel)? I_O_input : DY;//new

////////////////////////////////////////////////////////
//          ALU
////////////////////////////////////////////////////////
ALU_32
    A0
    (
    .S(RS),
    .T(RT),
    .FS(FS),
    .Y_LO(Y_LO),
    .Y_HI(Y_HI),
    .N(N),
    .Z(Z),
    .V(V),
```

```
    .C(C),
     .shamt(shamt)
    );
////////////////////////////////////////////////////////////
//                HILO and LO Reg
////////////////////////////////////////////////////////////

    always@(posedge clk , posedge reset)
        begin
        if(reset)
            {HI,LO} <= 64'h0;
        else if(HILO_ld)
            {HI,LO} <= {Y_HI,Y_LO};
        else
            {HI,LO}<={HI,LO};
        end

////////////////////////////////////////////////////////////
//      ALU_Out ,PC_In, D_In, RS and RT registers
////////////////////////////////////////////////////////////
reg [31:0] ALU_OUT_reg;
reg [31:0] D_In_reg;
reg [31:0] PC_In_reg;


        always@(posedge clk, posedge reset)
            begin
            if(reset)
                begin
                {ALU_OUT_reg,D_In_reg}<= 64'h0;
                {RS,RT}<=64'h0;
                PC_In_reg <= 32'b0;
                end
            else
                begin
                {ALU_OUT_reg,D_In_reg}<={Y_LO,Data_in};
                {RS,RT}<={S,T_Wire};
                PC_In_reg<=PC_in;
                end
            end
////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////
//                 Y MUX
////////////////////////////////////////////////////////////
assign ALU_OUT = (Y_Sel==3'b000)?            HI :
                     (Y_Sel==3'b001)?           LO :
                     (Y_Sel==3'b010)? ALU_OUT_reg :
                     (Y_Sel==3'b011)?    D_In_reg :
                     (Y_Sel==3'b100)?PC_In_reg : 32'bx;

////////////////////////////////////////////////////////////
//              D OUT assign
////////////////////////////////////////////////////////////
assign D_OUT = (D_OUT_Sel == 2'b00 )?   T_Wire :
                    (D_OUT_Sel == 2'b01 )? PC_In_reg:
                    (D_OUT_Sel == 2'b10 )? {28'b0,Flags}:
                                           T_Wire;//default


endmodule
```

III.B.3 Register File

```
`timescale 1ns / 1ps
/****************************** C E C S  4 4 0 ******************************
 *
 * File Name:  Register_File32x32.v
 * Project:    32 32-bit Register File
 * Designer:   Kassandra Flores
 * Email:      kassandra.flores@student.csulb.edu
 * Rev. No.:   Version 1.0
 * Rev. Date:  9/23/2016
 *
 * Purpose: To create 32 32-bit registers with D being the data available
 * to be written into the registers and D_Addr, being the register selected
 * to write to. Only when D En is high and D Add is not register 0, will the
 * contents at port D be written to the selected internal registers. S and
 * T are outputs that are able to dsplay current values held by internal
 * registers. S_Addr controls which register will appear on the S port and
 * T_Addr will control which register appears on the T port.
 ***************************************************************************/
module Register_File_32x32(clk,reset,S,T,D,S_Addr,T_Addr,D_Addr,D_En);


    input clk;
    input reset;

////////////////////////////////
// Data values present at input
////////////////////////////////
    input [31:0] D;
////////////////////////////////
/// Data Enable
////////////////////////////////
    input D_En;
////////////////////////////////
// Register Selects
////////////////////////////////
    input [4:0] D_Addr;
    input [4:0] S_Addr;
    input [4:0] T_Addr;
////////////////////////////////
// Ports available to display
// contents held in register file
////////////////////////////////
    output  [31:0] S;
    output  [31:0] T;
////////////////////////////////
// Internal data registers
////////////////////////////////
    reg [31:0] Data [31:0] ;

    always@(posedge clk , posedge reset)
        begin
            if(reset)
                Data[0] <= 32'h0;// Only r0 is effected by reset

            if(D_En && (D_Addr != 5'b000_00))
                Data[D_Addr] <= D;//If r0 is  not selected and D_En is high
                                  //then Data at port D is placed into register

            end

    assign S = Data[S_Addr];// Selects which register will be displayed by S
                            // by using S_Addr to specify which register
    assign T = Data[T_Addr];//Selects which register will be displayed by T
                            // by using T_Addr to specify which register


endmodule
```

III.B.4 Arithmetic Log Unit

```
`timescale 1ns / 1ps
/***************************** C E C S  4 4 0 ****************************
 *
 * File Name:  ALU_.32v
 * Project:    Integer Datapath
 * Designer:   Kassandra Flores
 * Email:      kassandra.flores@student.csulb.edu
 * Rev. No.:   Version 1.0
 * Rev. Date:  8/23/2016
 *
 * Purpose: To perform arithemtic or logic operations on the inputed operands
 *             and set the status flags depending on the selected operation.
 *              ALU Operation selection is 5 bits in length and all S and T
 *              inputs are 32 bits wide.
 *************************************************************************/
  module ALU_32(S,T,FS,Y_LO,Y_HI,N,Z,V,C,shamt
    );
////////////////////////////////
// Paramters
////////////////////////////////
     parameter

     MUL_OP = 5'H1E, DIV_OP = 5'H1F,BSLR = 5'h1A , BSRR = 5'h1B;

////////////////////////////////
// INPUTS
////////////////////////////////
    input [31:0] S;
    input [31:0] T;
    input [4:0] shamt;
    input [4:0] FS; // Function Select
////////////////////////////////
// OUTPUTS
////////////////////////////////
    output wire N,Z,V,C;
    output wire [31:0] Y_LO;
    output wire [31:0] Y_HI;
////////////////////////////////
// WIRES
////////////////////////////////
  wire [31:0] MIPS_Prod;
  wire [63:0] MPY_Prod;
  wire [31:0] DIV_Prod;
  wire [31:0] DIV_Rem;
  wire [31:0] BS_Y;
  wire V_m;
  wire C_m;
////////////////////////////////
// ALU OPs ( EXLUDING DIV & MUL )
////////////////////////////////
MIPS_32
  M0(
    .S(S),
    .T(T),
    .Y(MIPS_Prod),
    .FS(FS),
    .C(C_m),
    .V(V_m),
     .shamt(shamt)
    );
////////////////////////////////
// DIV
////////////////////////////////
DIV_32
 D0(
    .S(S),
    .T(T),
    .Product(DIV_Prod),
    .Remainder(DIV_Rem),
```

```
    .FS(FS)
    );
////////////////////////////////
// MUL
////////////////////////////////
MPY_32
 MPY0(
    .S(S),
    .T(T),
    .Y(MPY_Prod),
    .FS(FS)
    );
////////////////////////////////
//  Barrell Shift              //
////////////////////////////////
Barrell Shift
    Bs0(
    .FS(FS),
    .T(T),
    .shamt(shamt),
    .Y(BS_Y)
    );
////////////////////////////////
// Y_LO and Y_HI MUX
////////////////////////////////

assign Y_LO = (FS == MUL_OP)? MPY_Prod [31:0] :
              (FS == DIV_OP)? DIV_Prod :
                  (FS == BSLR | FS == BSRR )? BS_Y:
                    MIPS_Prod; //default
assign Y_HI = (FS == MUL_OP)? MPY_Prod [63:32]
                                      : (FS == DIV_OP)? DIV_Rem  : 16'H0000;
////////////////////////////////
// Carry and Overflow flags
////////////////////////////////

assign {C,V} =(FS != MUL_OP || FS != DIV_OP
                   || FS != BSRR || FS != BSLR) ? {C_m,V_m} : 1'b0;

////////////////////////////////
// Negative and Zero flags
////////////////////////////////
assign N = (FS == MUL_OP)? (Y_HI[31])         : (Y_LO[31]);
assign Z = (FS == MUL_OP || FS == DIV_OP)? (~|{Y_HI,Y_LO}) : (~|Y_LO );

endmodule
```

III.B.5 General ALU

```
`timescale 1ns / 1ps
/****************************** C E C S  4 4 0 ******************************
 *
 * File Name:  MIPS .32v
 * Project:    Integer Datapath
 * Designer:   Kassandra Flores
 * Email:      kassandra.flores@student.csulb.edu
 * Rev. No.:   Version 1.0
 * Rev. Date:  8/23/2016
 *
 * Purpose:  ALU Operations that will perform all arithmaic and logic
 *              operations except Divison and Multiplication and set the
 *           appropriate carry and overfow.
 *
 ************************************************************************/
module MIPS_32(S,T,Y,FS,C,V,shamt);
////////////////////////////////
// INPUTS
////////////////////////////////
    input [31:0] S;
```

```
    input [31:0] T;
    input [4:0] shamt;
    input [4:0] FS; // Function Select
////////////////////////////////
// OUTPUTS
////////////////////////////////
    output reg [31:0] Y;
    output reg C;
    output reg V;
////////////////////////////////
// ALU Operations
////////////////////////////////
    parameter
        PASS_S  = 5'H00,   PASS_T  = 5'H01,  ADD     = 5'H02,
        ADDU    = 5'H03,   SUB     = 5'H04,  SUBU    = 5'H05,
        SLT     = 5'H06,   SLTU    = 5'H07,  AND     = 5'H08,
        OR      = 5'H09,   XOR     = 5'H0A,  NOR     = 5'H0B,
        SLL     = 5'H0C,   SRL     = 5'H0D,  SRA     = 5'H0E,
        INC     = 5'H0F,   DEC     = 5'H10,  INC4    = 5'H11,
        DEC4    = 5'H12,   ZEROS   = 5'H13,  ONES    = 5'H14,
        SP_INIT = 5'H15,   ANDI    = 5'H16,  ORI     = 5'H17,
        XORI    = 5'H18,   LUI = 5'H19,

        UNUSED = 1'bX, SET = 1'b1, OFF = 1'b0;
////////////////////////////////
// Function Select Decoder
////////////////////////////////
integer int_T;

    always@(*)//Decoder of inputs
    begin
            C = 0;// carry
            V = 0;// overflow
           Y = 32'H0; // Function Ouput
            case(FS)

            PASS_S: Y = S;

            PASS_T: Y = T;

            ADDU:// Addition Unsigned
                    begin
                    {C,Y} = S + T;
                    V = C;
                    end
            ADD: // Addition Signed
                    begin
                    {C,Y} = S + T;
                    V = (S[31] == T[31])? (T[31]!=Y[31]) : 1'b0;
                    end
            SUB:// Subtraction Signed
                    begin
                    {C,Y} = S - T;
                    V = (S[31] ^ T[31])? (T[31] == Y[31]) : 1'b0;
                    end
            SUBU:// Subtraction Unsigned
                    begin
                    {C,Y} = S - T;
                    V = C;
                    end

////////////////////////////////////////
// SLT(Signed): If S is less T , Y = 1
//          Else, Y = 0
////////////////////////////////////////
            SLT:    Y = (S[31] == T[31])? (S<T) : (S[31] & ~T[31]);
////////////////////////////////////////
// SLT(Unsigned): If S is less T , Y = 1
//          Else, Y = 0
////////////////////////////////////////
            SLTU: Y = (S<T)? 1'b1: 1'b0;
```

```
//////////////////////////////////////////
// AND
//////////////////////////////////////////
            AND:  Y = S & T;
//////////////////////////////////////////
// OR
//////////////////////////////////////////
            OR: Y = S | T ;
//////////////////////////////////////////
// XOR
//////////////////////////////////////////
            XOR:    Y = S^T;
//////////////////////////////////////////
// NOR
//////////////////////////////////////////
            NOR:    Y = ~(S|T);
//////////////////////////////////////////
// SLL : Shift Left
//////////////////////////////////////////
            SLL:    {C,Y} = T << shamt;
//////////////////////////////////////////
// SRL : Shift Right
//////////////////////////////////////////
            SRL:
            begin
            C= T[0];
            Y = T >> shamt;
            end
//////////////////////////////////////////
// SRA : Shift Right Arithmetic
//////////////////////////////////////////

            SRA:
            begin
            int_T = T;
            {C,Y}= int_T >>> shamt; // performs sign extension
            end
//////////////////////////////////////////
// INC : Increment S by 1'b1 (value one)
//////////////////////////////////////////
            INC:
           begin
            {C,Y} = S + 1'b1;
            V = (!S[31]&Y[31]);
            end

//////////////////////////////////////////
// DEC: Decrament S by 1'b1 (value one)
//////////////////////////////////////////
            DEC:
                    begin
                    {C,Y} = S - 1'b1;
                    V =(S[31]&!Y[31]);
                    end
//////////////////////////////////////////
// INC4:  Increment S by 3'b100 (value 4)
//////////////////////////////////////////
            INC4:
                    begin
                    {C,Y} = S + 3'b100;
                    V = (!S[31]&Y[31]);
                    end
//////////////////////////////////////////
// DEC4:  Decrement S by 3'b100 (value 4)
//////////////////////////////////////////
            DEC4:
                    begin
                    {C,Y} = S - 3'b100;
                    V = (S[31]&!Y[31]);
                    end
```

```
            ZEROS:    Y= 32'H000000000;
            ONES:    Y= 32'HFFFFFFFF;
            SP_INIT:     Y = 32'h3FC;
            ANDI:     Y = S & {16'h0, T[15:0]};
            ORI:        Y = S | {16'h0, T[15:0]};
            XORI:       Y = S ^ {16'h0, T[15:0]};
            LUI:        Y = {T[15:0], 16'h0};

         endcase
        end
endmodule
```

III.B.6 Divide

```
`timescale 1ns / 1ps
/****************************** C E C S  4 4 0 ******************************
 *
 * File Name:  DIV_.32v
 * Project:    Integer Datapath
 * Designer:   Kassandra Flores
 * Email:      kassandra.flores@student.csulb.edu
 * Rev. No.:   Version 1.0
 * Rev. Date:  8/23/2016
 *
 * Purpose: To perform ALU Operation: Divison. Division of operands will be
 *          S divided by T. Product of division will be outputted into 32
 *          bit reg typ. Remainder of divison will be outputted into 32
 *              bit reg type.
 *
 *************************************************************************/
module DIV_32(S,T,Product,Remainder,FS
    );
////////////////////////////////
// INPUTS
////////////////////////////////
    input [31:0] S;
    input [31:0] T;
    input [4:0] FS; // Function Select
////////////////////////////////
// OUTPUTS
////////////////////////////////
    output reg [31:0]Product;
    output reg [31:0] Remainder;

    integer int_S, int_T;
    // casted to int so verilog will know they are
    // eithor a negative or positive number.
    always@(*)
        begin
        int_S = S;
        int_T = T;
        Product = int_S / int_T;
        Remainder = int_S % int_T;
        end

endmodule
```

III.B.7 Multiply

```
`timescale 1ns / 1ps
/****************************** C E C S  4 4 0 ******************************
 *
 * File Name:  MPY_.32v
 * Project:    Integer Datapath
 * Designer:   Kassandra Flores
 * Email:      kassandra.flores@student.csulb.edu
 * Rev. No.:   Version 1.0
 * Rev. Date:  8/23/2016
 *
 * Purpose: To perform ALU Operation: Multiplication. Multiplication of
 *              operands will be S * T. Product of mulptiplcation will be
 *              placed into a 32 bit reg type output
 *
 ***************************************************************************/
module MPY_32(S,T,Y,FS
    );
////////////////////////////////
// INPUTS
////////////////////////////////
    input [31:0] S;
    input [31:0] T;
    input [4:0] FS; // Function Select
////////////////////////////////
// OUTPUTS
////////////////////////////////
    output reg [63:0] Y;

    integer int_S, int_T;
    // casted to int so verilog will know they are
    // eithor a negative or positive number.
    always@(*)
        begin
        int_S = S;
        int_T = T;
        Y = int_S * int_T;
        end

endmodule
```

.

..

III.B.8 Barrel Shift

```
`timescale 1ns / 1ps
/****************************** C E C S  4 4 0 ******************************
 *
 * File Name:  BarrelShift_2.v
 * Project:    Senior Project
 * Designer:   Kassandra Flores, Adan Hernandez
 * Email:      kassandra.flores@student.csulb.edu,
 *                  adan.hernandez@student.csulb.edu
 * Rev. No.:   Version 1.0
 * Rev. Date:  10/09/2016
 *
 * Purpose:    To perform 32bit barrel shift right and left. Amount of times
 *             data will be rotated in either direction will be dependent on
 *             the shift amount selected.
 ***************************************************************************/
module Barrell_Shift(FS,T,shamt,Y);
    input [31:0] T;       //Input value
    input [4:0] FS;       //Function Select
    input [4:0] shamt;    //shift amount
    output reg [31:0] Y; //product

   parameter BSLR = 5'h1A , BSRR = 5'h1B;
    always@(*)
        begin
        case(FS)
            BSLR:
            case(shamt)
            5'h00: Y = T;
            5'h01: Y = {T[30:0],T[31]};
            5'h02: Y = {T[29:0],T[31:30]};
            5'h03: Y = {T[28:0],T[31:29]};
            5'h04: Y = {T[27:0],T[31:28]};
            5'h05: Y = {T[26:0],T[31:27]};
            5'h06: Y = {T[25:0],T[31:26]};
            5'h07: Y = {T[24:0],T[31:25]};
            5'h08: Y = {T[23:0],T[31:24]};
            5'h09: Y = {T[22:0],T[31:23]};
            5'h0A: Y = {T[21:0],T[31:22]};
            5'h0B: Y = {T[20:0],T[31:21]};
            5'h0C: Y = {T[19:0],T[31:20]};
            5'h0D: Y = {T[18:0],T[31:19]};
            5'h0E: Y = {T[17:0],T[31:18]};
            5'h0F: Y = {T[16:0],T[31:17]};
            5'h10: Y = {T[15:0],T[31:16]};
            5'h11: Y = {T[14:0],T[31:15]};
            5'h12: Y = {T[13:0],T[31:14]};
            5'h13: Y = {T[12:0],T[31:13]};
            5'h14: Y = {T[11:0],T[31:12]};
            5'h15: Y = {T[10:0],T[31:11]};
            5'h16: Y = {T[9:0],T[31:10]};
            5'h17: Y = {T[8:0],T[31:9]};
            5'h18: Y = {T[7:0],T[31:8]};
            5'h19: Y = {T[6:0],T[31:7]};
            5'h1A: Y = {T[5:0],T[31:6]};
            5'h1B: Y = {T[4:0],T[31:5]};
            5'h1C: Y = {T[3:0],T[31:4]};
            5'h1D: Y = {T[2:0],T[31:3]};
            5'h1E: Y = {T[1:0],T[31:2]};
            5'h1F: Y = {T[0],T[31:1]};

            default: Y = T;
            endcase

            BSRR :
            case(shamt)
            5'h00 : Y = T;
            5'h01 : Y = {T[0],T[31:1]};
            5'h02 : Y = {T[1:0],T[31:2]};
            5'h03 : Y = {T[2:0],T[31:3]};
```

```
            5'h04 : Y = {T[3:0],T[31:4]};
            5'h05 : Y = {T[4:0],T[31:5]};
            5'h06 : Y = {T[5:0],T[31:6]};
            5'h07 : Y = {T[6:0],T[31:7]};
            5'h08 : Y = {T[7:0],T[31:8]};
            5'h09 : Y = {T[8:0],T[31:9]};
            5'h0A : Y = {T[9:0],T[31:10]};
            5'h0B : Y = {T[10:0],T[31:11]};
            5'h0C : Y = {T[11:0],T[31:12]};
            5'h0D : Y = {T[12:0],T[31:13]};
            5'h0E : Y = {T[13:0],T[31:14]};
            5'h0F : Y = {T[14:0],T[31:15]};
            5'h10 : Y = {T[15:0],T[31:16]};
            5'h11 : Y = {T[16:0],T[31:17]};
            5'h12 : Y = {T[17:0],T[31:18]};
            5'h13 : Y = {T[18:0],T[31:19]};
            5'h14 : Y = {T[19:0],T[31:20]};
            5'h15 : Y = {T[20:0],T[31:21]};
            5'h16 : Y = {T[21:0],T[31:22]};
            5'h17 : Y = {T[22:0],T[31:23]};
            5'h18 : Y = {T[23:0],T[31:24]};
            5'h19 : Y = {T[24:0],T[31:25]};
            5'h1A : Y = {T[25:0],T[31:26]};
            5'h1B : Y = {T[26:0],T[31:27]};
            5'h1C : Y = {T[27:0],T[31:28]};
            5'h1D : Y = {T[28:0],T[31:29]};
            5'h1E : Y = {T[29:0],T[31:30]};
            5'h1F : Y = {T[30:0],T[31:31]};
            default : Y = T;
            endcase

            default: Y = 32'b0;
            endcase
        end
endmodule
```

III.B.9 Instruction Unit

```
`timescale 1ns / 1ps
/****************************** C E C S  4 4 0 ******************************
 *
 * File Name:  Instruction_Unit.v
 * Project:    Intstructon Unit
 * Designer:   Kassandra Flores, Adan Hernandez
 * Email:      kassandra.flores@student.csulb.edu,
 *                  adan.hernandez@student.csulb.edu
 * Rev. No.:   Version 1.0
 * Rev. Date:  10/09/2016
 *
 * Purpose: To create an instruction unit in which keeps track of where in
 * the program we are , fetches instructions from memory and places these
 * fetched instructions in the instruction register.The current value
 * of the program counter is used to select which address in instruction
 * memory will be read. This read instruction is then loaded onto
 * the instrucrton register. Only bits [11:0] of the PC (program counter)
 * are used to address the instrution memory
 ***************************************************************************/
module Instruction_Unit(clk,reset,PC_in,im_cs,im_wr,im_rd,pc_ld,pc_inc,pc_sel,
                                PC_out,ir_ld,IR_out,SE_16);
////////////////////////////////
//               INPUTS           //
////////////////////////////////
input clk;
input reset;
input im_cs,im_wr,im_rd; // Instruction Memory Controls
input pc_ld;                  // PC reg load control
input pc_inc;                 // PC reg increment control
input [1:0] pc_sel;      // PC load source select
input [31:0] PC_in;      // PC Input
input ir_ld;                  // IR register load control
////////////////////////////////
//               OUTPUTS          //
////////////////////////////////
output [31:0] PC_out;    // PC output
output [31:0] IR_out;    // IR output
output [31:0] SE_16;          // IR Sign Extension output
wire [11:0] PC_out_w;    // PC Current Value
assign PC_out_w = PC_out [11:0];//PC Current Value

////////////////////////////////
//           INTERNAL REGS     //
////////////////////////////////
reg [31:0] PC;//Program Counter
reg [31:0] IR;//Instruction Regiser


////////////////////////////////
//       Instruction Mem       //
////////////////////////////////
wire [31:0] D_Out;

Data_Memory
    Instruction Memory
    (
    .clk(clk),
    .dm_cs(im_cs),
    .dm_wr(im_wr),
    .dm_rd(im_rd),
    .D_Out(D_Out),
    .Addr(PC_out_w),
    .D_In(32'h0000_0000)
    );
////////////////////////////////
//          Sign Extension        //
////////////////////////////////
assign SE_16 = {{16{IR[15]}},IR[15:0]};
```

```
assign IR_out = IR;//assign IR_out to contents of IR

//////////////////////////////
//          PC Input Mux         //
//////////////////////////////
wire [31:0] PC_in_w;//Input into PC register
assign PC_in_w = (pc_sel == 2'b00)?     PC+{SE_16[29:0],2'b00}:
                     (pc_sel == 2'b01)? {PC[31:28],IR[25:0],2'b00}:
                    (pc_sel == 2'b10)?                      PC_in:
                                              {PC[31:24],IR[20:0],2'b00};
//////////////////////////////
//          PC Register          //
//////////////////////////////
always@(posedge clk , posedge reset)
    begin
    if(reset)
        PC<= 32'h0000_0000;
    else if(pc_inc)
            PC<= PC + 3'b100;
    else if(pc_ld)
            PC<= PC_in_w;
    else
            PC<= PC;
    end
assign PC_out = PC;
//////////////////////////////
//          IR Register          //
//////////////////////////////
always@(posedge clk, posedge reset)
    begin
    if(reset)
        IR<=32'h0000_0000;
    else if(ir_ld)
        IR<=D_Out;
    else
        IR<=IR;
    end


endmodule
```

III.B.10 I/O Memory

```
`timescale 1ns / 1ps
/***************************** C E C S  4 4 0 ****************************
 *
 * File Name:  IO_MEm.v
 * Project:    Senior Project
 * Designer:   Kassandra Flores, Adan Hernandez
 * Email:      kassandra.flores@student.csulb.edu,
 *                  adan.hernandez@student.csulb.edu
 * Rev. No.:   Version 1.0
 * Rev. Date:  10/09/2016
 *
 * Purpose:    To create a 0 to 4095 byte addressable I/O memory locations.
 *             All data placed into the byte addressable memory locations
 *             will be in Big Endian format. Reading from memory will be
 *             asynchronous and writing to memory is synchronous. To perform
 *             A read operation, imrd must be asserted to perform a write
 *             Imwr must be asserted. The IO is cable of requesting a interrup
 *
 *************************************************************************/
module I_O(sys_clk,intr , // System signals
              Addr,          // Memory Address
              D_In,          // Data to be loaded into Memory
              intr_req,      // Interrupt request from I/O module
              intr_ack,      // Interrupt Acknowledge
              IO_wr,         // Write (synchronous)
              IO_rd,         // Read (asynchronous)
              IO_D_Out       // Data out
    );
input  sys_clk;           // System Clock
input  intr;              // Interrupt Request
input  [11:0] Addr;       // Memory is organized as 4096x8 byte addressible
                             // memory (i.e 1024x32).Big Endian Format
input  [31:0] D_In;       // Data in
output reg intr_req;      // Interrupt request from I/O module
input  intr_ack;          // Interrupt Acknowledge
input  IO_wr,IO_rd;       // I/O read and write control signals
output [31:0] IO_D_Out;   // contents of memory location specified by Addr

////////////////////////////////////////
// Byte Addressible Memory Locations
////////////////////////////////////////
reg [7:0] Mem [4095:0];

assign IO_D_Out = ({IO_rd &!IO_wr})?
                       {Mem[Addr],Mem[Addr + 3'b001],Mem[Addr+3'b010],Mem[Addr + 3'b011]}
                        : 32'bz;
////////////////////////////////////////
//       Writing to Memory
////////////////////////////////////////
always@(posedge sys_clk)
    begin
    if(IO_wr&!IO_rd)
        begin
        {Mem[Addr],Mem[Addr + 3'b001],Mem[Addr+3'b010],Mem[Addr + 3'b011]} <= D_In;
        end
    else
        begin
        {Mem[Addr],Mem[Addr + 3'b001],Mem[Addr+3'b010],Mem[Addr + 3'b011]} <=
        {Mem[Addr],Mem[Addr + 3'b001],Mem[Addr+3'b010],Mem[Addr + 3'b011]};
        end
    end

always@(posedge sys_clk)
    if(intr_ack)   intr_req<=0;
    else if (intr) intr_req<=1;
    else           intr_req<=0;

endmodule
```

III.B.11 Data Memory/Instruction Memory

```
`timescale 1ns / 1ps
/***************************** C E C S  4 4 0 ****************************
 *
 * File Name:  Data_Memory.v
 * Project:    Integer Datapath
 * Designer:   Kassandra Flores
 * Email:      kassandra.flores@student.csulb.edu
 * Rev. No.:   Version 1.0
 * Rev. Date:  8/23/2016
 *
 * Purpose: To create a 0 to 4095 byte addressable memory locations. All data
 * placed into the byte addressable memory locations will be in Big Endian
 * format. Reading from memory is asynchronous and writing to memory is
 * synchronous. To perform a read operation , dm_cs and dm_rd must be
 * asserted. To perform a write, dm cs and dm wr must be asserted.
 ************************************************************************/
module Data_Memory(clk,dm_cs,dm_wr,dm_rd,D_Out,Addr,D_In);
////////////////////////////////////
//           Inputs
////////////////////// ///////////////
input clk;
input dm_cs;                    //Chip Select
input dm_wr;                    //Write   (Synchronous)
input dm_rd;                    //Read    (Asynchronous)
input [11:0] Addr;         //Memory is organized as 4096x8 byte addressible
                                //memory (i.e 1024x32).Big Endian Format
input [31:0] D_In;         //Data in
////////////////////////////////////

////////////////////////////////////
//            Outputs
////////////////////////////////////
output [31:0] D_Out;       //Data out
////////////////////////////////////

//////////////////////////////////////////
// Byte Addressible Memory Locations
//////////////////////////////////////////
reg [7:0] Mem [4095:0];

//////////////////////////////////////////
//       Reading from Memory
//////////////////////////////////////////

assign D_Out  = ({dm_rd & dm_cs&!dm_wr})?
                {Mem[Addr],Mem[Addr + 3'b001],Mem[Addr+3'b010],Mem[Addr + 3'b011]}
                : 32'bz;

//////////////////////////////////////////
//       Writing to Memory
//////////////////////////////////////////
always@(posedge clk)
    begin
    if(dm_cs & dm_wr&!dm_rd)
    begin
    {Mem[Addr],Mem[Addr + 3'b001],Mem[Addr+3'b010],Mem[Addr + 3'b011]} <= D_In;
    end
    else
    begin
    {Mem[Addr],Mem[Addr + 3'b001],Mem[Addr+3'b010],Mem[Addr + 3'b011]} <=
    {Mem[Addr],Mem[Addr + 3'b001],Mem[Addr+3'b010],Mem[Addr + 3'b011]};
    end
    end
endmodule
```

III.C Memory Modules

III.C.1 Memory Module 1

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

10 22 00 01  //             beq  $01, $02, no_eq
10 23 00 03  //             beq  $01, $03, yes_eq
3c 0e ff ff  // no_eq:     lui  $14, 0xFFFF
35 ce ff ff  //             ori  $14, 0xFFFF
00 00 00 0d  //             break

00 00 70 20  // yes_eq:    add  $14, $0, $0

14 23 00 01  //             bne  $01, $03, no_ne
14 22 00 03  //             bne  $01, $02, yes_ne
3c 0f ff ff  // no_ne:     lui  $15, 0xFFFF
35 ef ff ff  //             ori  $15, 0xFFFF
00 00 00 0d  //             break

00 00 78 20  // yes_ne:    add  $15, $0, $0
3c 0d 10 01  //             lui  $13, 0x1001
35 ad 00 c0  //             ori  $13, 0x00C0
ad a1 00 00  //             sw   $01, 0($13)
00 00 00 0d  //             break
```

Loading Registers with initial contents

R01 = 0x12345678

R02 = 0x87654321

R03 = R02 = 0x87654321

Checks if the "branch on equal" instruction is working.

If BEQ R01 and R02 causes a branch → " branch on equal" instruction **doesn't work**

and R14 will equal 0xFFFFFFFF to show that the instruction **doesn't work**

If BEQ R01 and R03 causes a branch → "branch on equal" instruction **works**

And R14 will equal 0x00000000 to show that the instruction works

Loads R13 with value and stores value of register in memory location

R13 = 0x100100c0

Mem[R13+0] = R01 ← Uses register R13 added with the value 0 to select which memory address to write register R01 into

Checks if the "branch on not equal" instruction is working.

If BNE R01 and RO3 causes a branch → "branch on not equal " instruction **doesn't work**

and R15 will equal oxFFFFFFFF to show that the instruction **doesn't work**

if BNE R01 and R02 causes a branch → " branch on not equal" **works**

and R15 will equal 0x00000000 to show that the instruction **works**

III.C.2 Memory Module 2

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

00 01 08 42  // top:  srl  $01, $01, 1
ad e1 00 00  //       sw   $01, 0($15)
21 ef 00 04  //       addi $15, $15, 4
20 42 ff ff  //       addi $02, $02, -1
14 40 ff fb  //       bne  $02, $00, top
08 10 00 0c  //       j    exit
00 00 00 0d  //       break

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

Loading Registers with initial contents

R01 = 0xFFFFFFFF

R02 = 0x00000010

R15 = 0x100100c0

Loop that repeats till Register content equals zero

The loop performs the following :

1. Shifts the contents of R01 to the right (effectively dividing the value held inside by 2)
2. Stores the value held in R01 into memory location specified by Register R15
3. Increments R15 held value by 4
4. Decrements R02 held value by 1
5. Repeats steps 1 to 4 if value in R02 doesn't equal 0.

Once **R02 equals 0**, the loop will break and jump to the next instruction

Loads R14 with value

R14 = 0x5A5A3C3C

III.C.3 Memory Module 3

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

00 01 08 43  // top:       sra  $01, $01, 1
ad e1 00 00  //            sw   $01, 0($15)
21 ef 00 04  //            addi $15, $15, 4
20 42 ff ff  //            addi $02, $02, -1
14 40 ff fb  //            bne  $02, $00, top

08 10 00 0c  //            j    exit
00 00 00 0d  //            break

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

Loading Registers with initial contents

R01 = 0x8000FFFF

R02 = 0x00000010

R15 = 0x100100c0

Loop that repeats till Register content equals zero

The loop performs the following:

1. Shifts the contents R01 to the right (shifting in 0 if the most significant bit is 0 or 1 if the most significant bit is 1)
2. Stores the value held in R01 into memory location specified by Register R15
3. Increments R15 held value by 4
4. Decrements R02 held value by 1
5. Repeats steps 1 to 4 if value in R02 doesn't equal 0.

Once **R02 equals 0**, the loop will break and jump to the next instruction

Loads R14 with value

R14 = 0x5a5a3c3c

III.C.4 Memory Module 4

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

00 01 08 40  // top:       sll  $01, $01, 1
ad e1 00 00  //             sw   $01, 0($15)
21 ef 00 04  //             addi $15, $15, 4
20 42 ff ff  //             addi $02, $02, -1
00 02 18 2a  //             slt  $03, $00, $02
14 60 ff fa  //             bne  $03, $00, top

08 10 00 0d  //             j    exit
00 00 00 0d  //             break

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

Loading Registers with initial contents

R01 = 0xFFFFFFFF

R02 = 0x00000010

R15 = 0x100100c0

Loop that repeats till Register content equals zero

The loop performs the following:

1. Shifts the contents R01 to the left once (effectively doubling its value)
2. Stores the value held in R01 into memory location specified by Register R15
3. Increments R15 held value by 4
4. Decrements R02 held value by 1
5. Checks if value held in R00 is less than R02. If R00 is less than R02, a value of 1 is placed into R03. If R00 is greater than R02, a value 0 is placed into R03
6. Repeats steps 1 to 5 if value in R03 doesn't equal 0.

Once **R03 equals 0** , the loop will break and jump to the next instruction

Loads R14 with value

R14 = 0x5a5a3c3c

III.C.5 Memory Module 5

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

00 01 08 40  // top:       sll  $01, $01, 1
ad e1 00 00  //             sw   $01, 0($15)
21 ef 00 04  //             addi $15, $15, 4
20 42 00 01  //             addi $02, $02, 1
28 43 00 00  //             slti $03, $02, 0
14 60 ff fa  //             bne  $03, $00, top

08 10 00 0d  //             j    exit
00 00 00 0d  //             break

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

Loading Registers with initial contents

R01 = 0xFFFFFFFF

R02 = 0xFFFFFFF0

R15 = 0x100100c0

Loop that repeats till Register content equals zero

The loop performs the following:

1. Shifts the contents R01 to the left once (effectively doubling its value)
2. Stores the value held in R01 into memory location specified by Register R15
3. Increments R15 held value by 4
4. Decrements R02 held value by 1
5. Checks if value held in R02 is less than immediate value 0. If R02 is less than 0, a value of 1 is placed into R03. If R02 is greater than 0, a value 0 is placed into R03
6. Repeats steps 1 to 5 if value in R03 doesn't equal 0.

Once **R03 equals 0**, the loop will break and jump to the next instruction

Loads R14 with value

R14 = 0x5a5a3c3c

III.C.6 Memory Module 6

```
v2.0
3c 0f 10 01  // lui  $15, 0x1001
35 ef 00 00  // ori  $15, 0x0000
3c 0e 10 01  // lui  $14, 0x1001
35 ce 00 c0  // ori  $14, 0x00C0
20 0d 00 10  // addi $13, $00, 16
8d e1 00 04  // lw   $01, 04($15)
8d e2 00 08  // lw   $02, 08($15)
8d e3 00 0c  // lw   $03, 12($15)
8d e4 00 10  // lw   $04, 16($15)
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)
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
```

Loading Registers with initial contents

R15 = 0x10010000

R14 = 0x100100c0

R13 = 0x00000010

Loads Memory Contents into Registers

The value held in Register R15, with an added offset, is used to address a memory location. the value at this specified memory location is then loaded into a Register.

R01 = Mem[R15+0]

R02 = Mem[R15+4]

R03 = Mem[R15+8]

R04 = Mem[R15+12]

R05 = Mem[R15+16]

R06 = Mem[R15+20]

R07 = Mem[R15+20]

R08 = Mem[R15+20]

R09 = Mem[R15+20]

R10 = Mem[R15+20]

R11 = Mem[R15+20]

R12= Mem[R15+20]

Loop that Loads Memory and Stores into Memory

The value held in register R15 , with an added offset of 0, is used to address a memory location and load contents from memory and load it into register R17.

The value held in R14 , with an added offset of 0 is used to address a memory location and store the contents of R17 in that memory location

The loop performs the following:

1. Load value held at memory location specified by R15 into R17
2. Store value held in register 17 into memory location specified by R14
3. Increment value held in R15 by 4
4. Increment value held in R14 by 4
5. Decrement value held in R13 by 1
6. Repeats steps 1 to 5 if value in R13 doesn't equal 0.

Once **R13 equals 0** , the loop will break and the break instruction will occur.

III.C.7 Memory Module 7

```
@0
3c 0f 10 01  // main:       lui  $15, 0x1001
35 ef 00 00  //              ori  $15, 0x0000
3c 0e 10 01  //              lui  $14, 0x1001
35 ce 00 c0  //              ori  $14, 0x00C0
20 0d 00 10  //              addi $13, $00, 16
8d e1 00 04  //              lw   $01, 04($15)
8d e2 00 08  //              lw   $02, 08($15)
8d e3 00 0c  //              lw   $03, 12($15)
8d e4 00 10  //              lw   $04, 16($15)
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
00 00 00 0d  //              break

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
```

Loading Registers with initial contents

R15 = 0x10010000

R14 = 0x100100c0

R13 = 0x00000010

Jumps to instruction and saves current pc address

Jumps from the current location in the program to another specified location but saves(links) the current location in the program to register R31.

If R15 equals 0xFFFFFFFF , then "jump and link (jal) " instruction and "jump register (jr) " instruction worked

Loads Memory Contents into Registers

The value held in Register R15, with an added offset, is used to address a memory location. the value at this specified memory location is then loaded into selected Register.

R01 = Mem[R15+4] R02 = Mem[R15+8]

R03 = Mem[R15+12] R04 = Mem[R15+16]

R05 = Mem[R15+20] R06 = Mem[R15+24]

R07 = Mem[R15+28] R08 = Mem[R15+32]

R09 = Mem[R15+36] R10 = Mem[R15+40]

R11 = Mem[R15+44] R12= Mem[R15+48]

Loop that Loads Memory and Stores into Memory

The value held in register R15 , with an added offset of 0, is used to address a memory location and load contents from memory and load it into register R17.

The value held in R14 , with an added offset of 0 is used to address a memory location and store the contents of R17 in that memory location

The loop performs the following:

1. Load value held at memory location specified by R15 into R17
2. Store value held in register 17 into memory location specified by R14
3. Increment value held in R15 by 4
4. Increment value held in R14 by 4
5. Decrement value held in R13 by 1
6. Repeats steps 1 to 5 if value in R13 doesn't equal 0.

Once **R13 equals 0** , the loop will break and the jump register (jr) instruction will occur.

If the jump register (jr) works, we will see the pass flag:

R15 = 0xFFFFFFFF

III.C.8 Memory Module 8

```
@0
3c 0f 10 01  // main:      lui  $15, 0x1001
35 ef 00 00  //             ori  $15, 0x0000
8d e1 00 00  //             lw   $01, 00($15)
8d e2 00 04  //             lw   $02, 04($15)
8d e3 00 08  //             lw   $03, 08($15)
8d e4 00 0c  //             lw   $04, 12($15)
8d e5 00 10  //             lw   $05, 16($15)
8d e6 00 14  //             lw   $06, 20($15)
8d e7 00 18  //             lw   $07, 24($15)
00 22 00 18  //             mult $01, $02
00 00 40 12  //             mflo $08
14 a8 00 10  //             bne  $05, $08, fail1
00 62 00 18  //             mult $03, $02
00 00 48 12  //             mflo $09
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
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
14 ad 00 12  //             bne  $05, $13, fail4

3c 0e 00 00  // pass:      lui  $14, 0x0000
35 ce 00 00  //             ori  $14, 0x0000
00 00 00 0d  //             break
3c 0e ff ff  // fail1:     lui  $14, 0xFFFF
35 ce ff ff  //             ori  $14, 0xFFFF
00 00 00 0d  //             break
3c 0e ff ff  // fail2L:    lui  $14, 0xFFFF
35 ce ff fe  //             ori  $14, 0xFFFE
3c 0e ff ff  // fail2H:    lui  $14, 0xFFFF
35 ce ff fd  //             ori  $14, 0xFFFD
00 00 00 0d  //             break
3c 0e ff ff  // fail3L:    lui  $14, 0xFFFF
35 ce ff fc  //             ori  $14, 0xFFFC
00 00 00 0d  //             break
3c 0e ff ff  // fail3H:    lui  $14, 0xFFFF
35 ce ff fb  //             ori  $14, 0xFFFB
00 00 00 0d  //             break
3c 0e ff ff  // fail4:     lui  $14, 0xFFFF
35 ce ff fa  //             ori  $14, 0xFFFA
00 00 00 0d  //             break
```

Loading Registers with initial contents

R15 = 0x10010000

Loads Memory Contents into Registers

The value held in Register R15, with an added offset, is used to address a memory location. the value at this specified memory location is then loaded into the selected Register.

R01 = Mem[R15+0] R02 = Mem[R15+4]

R03 = Mem[R15+8] R04 = Mem[R15+12]

R05 = Mem[R15+16] R06 = Mem[R15+20]

R07 = Mem[R15+24]

Multiplying Registers and performing " branch if not equal"

The values held in two different registers are being multiplied together and their product placed into specified register(s) (product might be placed into 2 registers if product from multiplication exceeds a 32 bit value amount)

R08 = R01 x R02

R09 = R03 x R02

R10 = R03 x R02

R11 = R01 x R04

R12 = R01 x R04

R13 = R03 x R04

If R08 does not equal R05 move from low (mlfo) or multiplication (mult) does not work , and branching to fail flag 1 will occur.

If R06 does not equal R09 move from low (mlfo) or multiplication (mult) does not work , and branching to fail flag 2l will occur.

If R07 does not equal R10 move from high (mlfh) or multiplication (mult) does not work , and branching to fail flag 2h will occur.

Loads Registers with Fail Flag values or pass value

Should either move from high (mfhi), move from low (mflo) or multiplication instruction fail, the following values will be loaded onto register R14 to indicate a failure in one of the previous instructions has occurred.

If none of the above instructions failed, then the value 0x00000000 will be placed into R14 indicating all instructions succeeded

Pass Flag:

R14 = 0x00000000

Fail Flags:

R14 = 0xFFFFFFFF, R14 = 0xFFFFFFFE, R14 = 0xFFFFFFFC

R14 = 0xFFFFFFFB, R14 = 0xFFFFFFFA

III.C.9 Memory Module 9

```
@0
3c 0f 10 01  // main:       lui  $15, 0x1001
35 ef 00 00  //             ori  $15, 0x0000
8d e1 00 00  //             lw   $01, 00($15)
8d e2 00 04  //             lw   $02, 04($15)
8d e3 00 08  //             lw   $03, 08($15)
8d e4 00 0c  //             lw   $04, 12($15)
8d e5 00 10  //             lw   $05, 16($15)
8d e6 00 14  //             lw   $06, 20($15)
8d e7 00 18  //             lw   $07, 24($15)
8d e8 00 1c  //             lw   $08, 28($15)


00 22 00 1a  //             div  $01, $02
00 00 48 12  //             mflo $09
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
00 00 50 10  //             mfhi $10
15 27 00 17  //             bne  $09, $07,
15 48 00 19  //             bne  $10, $08, fail2R


00 24 00 1a  //             div  $01, $04
00 00 48 12  //             mflo $09
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
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
00 0b 60 20  //             add  $12, $00, $11
00 0b 68 20  //             add  $13, $00, $11
00 0b 70 20  //             add  $14, $00, $11
00 00 00 0d  //             break


3c 0e ff ff  // fail1Q:     lui  $14, 0xFFFF
35 ce ff ff  //             ori  $14, 0xFFFF
00 00 00 0d  //             break


3c 0e ff ff  // fail1R:     lui  $14, 0xFFFF
35 ce ff fe  //             ori  $14, 0xFFFE

00 00 00 0d  //             break
3c 0e ff ff  // fail2Q:     lui  $14, 0xFFFF
35 ce ff fd  //             ori  $14, 0xFFFD
00 00 00 0d  //             break

3c 0e ff ff  // fail2R:     lui  $14, 0xFFFF
35 ce ff fc  //             ori  $14, 0xFFFC
00 00 00 0d  //             break
```

Loading Registers with initial contents

R15 = 0x10010000

Loads Memory Contents into Registers

The value held in Register R15, with an added offset, is used to address a memory location. the value at this specified memory location is then loaded into the selected Register.

R01 = Mem[R15+0] R02 = Mem[R15+4]

R03 = Mem[R15+8] R04 = Mem[R15+12]

R05 = Mem[R15+16] R06 = Mem[R15+20]

R07 = Mem[R15+24] R08= Mem[R15+28]

Dividing Registers and performing " branch if not equal"

The following registers are being divided:

R01/R02 , R03/R02 , R03/R04

The Quotient of whatever division is occurring is placed into R09

The Remainder of whatever division is occurring is placed into R10

When dividing R01/R02, if R09 does not equal R05, branching to fail flag 1Q will occur

When dividing R01/R02, if R10 does not equal R06, branching to fail flag 1R will occur

When dividing R03/R02, if R09 does not equal R07, branching to fail flag 2Q will occur

When dividing R03/R02, if R10 does not equal R08, branching to fail flag 2R will occur

When dividing R01/R04, if R09 does not equal R07, branching to fail flag 3Q will occur

When dividing R01/R04, if R10 does not equal R06, branching to fail flag 3R will occur

```
3c 0e ff ff  // fail3Q:     lui  $14, 0xFFFF
35 ce ff fb  //             ori  $14, 0xFFFB
00 00 00 0d  //             break
3c 0e ff ff  // fail3R:     lui  $14, 0xFFFF
35 ce ff fa  //             ori  $14, 0xFFFA
00 00 00 0d  //             break
3c 0e ff ff  // fail4Q:     lui  $14, 0xFFFF
35 ce ff f9  //             ori  $14, 0xFFF9
00 00 00 0d  //             break
3c 0e ff ff  // fail4R:     lui  $14, 0xFFFF
35 ce ff f8  //             ori  $14, 0xFFF8
00 00 00 0d  //             break
```

Load Register with Fail Flag values or Pass Values

Should all division operations **succeed** then the following **pass flag** values will be placed into R11 , R12, R13, R14

Should any of the division operations fail, then the appropriate fail flag will be placed into register R14

Pass Flags:

R11 = 0x00000000 , R12 = 0x00000000
R13 = 0x00000000, R14 = 0x00000000

Fail Flags:

R14 = 0xFFFFFFFF, R14 = 0xFFFFFFFE,
R14 = 0xFFFFFFFD, R14 = 0xFFFFFFFC,
R14 = 0xFFFFFFFB, R14 = 0xFFFFFFFA,
R14 = 0xFFFFFFF9, R14 = 0xFFFFFFF8

III.C.10 Memory Module 10

```
@0


3c 0f 10 01  // main:       lui  $15, 0x1001
35 ef 00 c0  //              ori  $15, 0x00C0
20 01 ff 8a  //              addi $01, $00, -118
20 02 00 8a  //              addi $02  $00,  138




0c 10 00 22  //              jal  slt_tests




3c 0d 77 88  //              lui  $13, 0x7788
35 ad 77 88  //              ori  $13, 0x7788
3c 0c 88 77  //              lui  $12, 0x8877
35 8c 88 77  //              ori  $12, 0x8877
3c 0b ff ff  //              lui  $11, 0xFFFF
35 6b ff ff  //              ori  $11, 0xFFFF


01 ac 50 26  //              xor  $10, $13, $12
11 4b 00 02  //              beq  $10, $11, xor_pass
20 0e ff fb  //              addi $14, $00, -5
00 00 00 0d  //              break




01 ac 48 24  // xor_pass:   and  $09, $13, $12
11 20 00 02  //              beq  $09, $00, and_pass
20 0e ff fa  //              addi $14, $00, -6
00 00 00 0d  //              break


01 e2 48 25  // and_pass:   or   $09, $15, $02
3c 08 10 01  //              lui  $08, 0x1001
35 08 00 ca  //              ori  $08, 0x00CA
11 09 00 02  //              beq  $08, $09, or_pass
20 0e ff f9  //              addi $14, $00, -7
00 00 00 0d  //              break
```

Jump to Instruction and save current program counter address

Jumps from the current location in the program to another specified location but saves(links) the current location in the program to register R31.

Loading Registers with initial contents

R13 = 0x77887788 R01 = 0x8877FFFF
R12 = 0xFFFFFFFF

Check if the XOR instruction is working

The following registers are being XOR'ed and having the xor'ed value placed into a selected register

R10 = R13 XOR R12

If R10 equals R11 then the xor instruction works

If the two registers do not equal ,xor instruction failed. Failure flag value 0xFFFFFFFA will be loaded into R14

Check if the AND instruction is working

The following registers are being AND'ed and having the and'ed value placed into a selected register

R09 = R12 AND R13

If R09 equals R00 then the AND instruction works

If the two registers do not equal ,AND instruction failed. Failure flag value 0xFFFFFFF9 will be loaded into R14

Check if the OR instruction is working

The following registers are being OR'ed and having the or'ed value placed into a selected register

R09 = R15 AND R02

R08 is being placed with the following value : 0x100100CA

If R09 equals R08 then the OR instruction works

If the two registers do not equal ,OR instruction failed. Failure flag value 0xFFFFFFF9 will be loaded into R14

```
01 e2 48 27  // or_pass:    nor  $09, $15, $02
3c 08 ef fe  //             lui  $08, 0xEFFE
35 08 ff 35  //             ori  $08, 0xFF35
11 09 00 02  //             beq  $08, $09, nor_pass
20 0e ff f8  //             addi $14, $00, -8
00 00 00 0d  //             break




ad e8 00 10  // nor pass:   sw   $08, 0x10($15)
00 00 70 20  //             add  $14, $00, $00
00 00 00 0d  //             break
             //

00 22 18 2a  // slt_tests: slt  $03, $01, $02
14 60 00 02  //             bne  $03, $00, slt1
20 0e ff ff  //             addi $14, $00, -1
00 00 00 0d  //             break
20 04 00 c0  // slt1:       addi $04, $00, 0xC0
ad e4 00 00  //             sw   $04, 0x00($15)

00 41 18 2b  //             sltu $03, $02, $01
14 60 00 02  //             bne  $03, $00, slt2
20 0e ff fe  //             addi $14, $00, -2
00 00 00 0d  //             break
20 05 00 c4  // slt2:       addi $05, $00, 0xC4
ad e5 00 04  //             sw   $05, 0x04($15)

00 41 18 2a  //             slt  $03, $02, $01
10 60 00 02  //             beq  $03, $00, slt3
20 0e ff fd  //             addi $14, $00, -3
00 00 00 0d  //             break
20 06 00 c8  // slt3:       addi $06, $00, 0xC8
ad e6 00 08  //             sw   $06, 0x08($15)

00 22 18 2b  //             sltu $03, $01, $02
10 60 00 02  //             beq  $03, $00, slt4
20 0e ff fc  //             addi $14, $00, -4
00 00 00 0d  //             break
20 07 00 cc  // slt4:       addi $07, $00, 0xCC
ad e7 00 0c  //             sw   $07, 0x0C($15)
03 e0 00 08  //             jr   $31
```

Check if the NOR instruction is working

The following registers are being NOR'ed and having the nor'ed value placed into a selected register

R09 = R15 AND R02

R08 = 0xFFFEFF35

If R09 equals R08 then the NOR instruction **works**

If the two registers do not equal ,NOR instruction failed. Failure flag value 0xFFFFFFF8 will be loaded into R14

Loading Memory with register contents and Pass Flags

The value held in register R15 , with an added offset of 16 is used to address a memory location and stores contents from register R08 into the selected memory location.

After storing into memory, **if all instructions worked** R14 gets the value 0x00000000

Check if the SLT and SLTU instruction is working

The following registers are being compared and the result of their comparison is being stored into a selected register

Unisgned values

R03 = if (R02< R01) R03 gets 1 , else R10 gets 0

R03 = if (R01< R02) R03 gets 1 , else R10 gets 0

Signed Values

R03 = if (R02< R01) R03 gets 1 , else R10 gets 0

R03 = if (R01< R02) R03 gets 1 , else R10 gets 0

--------Example of SLT test on registers R02 and R01-----

R03 = if (R02< R01) R10 gets 1 , else R10 gets 0

If R03 is equal to R00, slt instruction failed, load fail flag 0xFFFFFFFF into R14

If R03 is not equal to R00, slt instruction **worked**, load **pass flag** 0x000000C0 into Memory location Mem[R14]

III.C.11 Memory Module 11

```
@0
3c 0f 10 01  // main:       lui  $15, 0x1001
35 ef 00 c0  //             ori  $15, 0x00C0
20 01 ff 8a  //             addi $01, $00, -118
20 02 00 8a  //             addi $02  $00,  138

0c 10 00 08  //             jal  blt_tests
ad e1 00 18  //             sw   $01, 0x18($15)
ad e2 00 1c  //             sw   $02, 0x1C($15)
00 00 00 0d  //             break

18 20 00 02  // blt_tests: blez $01, blez_p1
20 0e ff ff  //             addi $14, $00, -1
00 00 00 0d  //             break
20 03 00 c0  // blez_p1:    addi $03, $00, 0xC0
ad e3 00 00  //             sw   $03, 0x00($15)
18 40 00 03  //             blez $02, blez_f2
20 04 00 c4  //             addi $04, $00, 0xC4
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
00 00 00 0d  //             break
18 00 00 02  // blez_p2:    blez $0, blez_p3
20 0e ff fd  //             addi $14, $00, -3
00 00 00 0d  //             break
20 05 00 c8  // blez_p3:    addi $05, $00, 0xC8
ad e5 00 08  //             sw   $05, 0x08($15)

1c 40 00 02  //             bgtz $02, bgtz_p1
20 0e ff fc  //             addi $14, $00, -4
00 00 00 0d  //             break
20 06 00 cc  // bgtz_p1:    addi $06, $00, 0xCC
ad e6 00 0c  //             sw   $06, 0x0C($15)
1c 20 00 03  //             bgtz $01, bgtz_f2
20 07 00 d0  //             addi $07, $00, 0xD0
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
00 00 00 0d  //             break
1c 20 00 03  // bgtz_p2:    bgtz $01, bgtz_f3
20 08 00 d4  //             addi $08, $00, 0xD4
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
00 00 00 0d  //             break
20 0e 00 00  // bgtz_p3:    addi $14, $00, 0
03 e0 00 08  //             jr   $31
```

Loading Registers with initial contents

R15 = 0x100100C0 R01 = 0xFFFFFF8A
R02 = 0x0000008A

Jump to Instruction and save current program counter address

Jumps from the current location in the program to another specified location but saves(links) the current location in the program to register R31.

Loading Memory with Pass Flags The value held in register R15 , with an added offset is used to address a memory location and stores contents from a register into the selected memory location.

If all instructions worked, pass values:

R01 will be stored into memory location M[R15+24]

R02 will be stored into memory location M[R15+28]

Check if the "Branch if less than or equal Zero (BLTZ) " instruction is working

Registers being evaluated are:

R01 R02 R00

----------Example of BLTZ test on register R01--------

If R01 is not less than or equal to zero

BLTZ instruction failed. Fail flag value 0xFFFFFFFF will be loaded into R14

If R01 is less than or equal to zero

BLTZ instruction worked. Pass flag value held in R03 will be stored into Mem[R15]

Check if the "Branch if greater than Zero (BGTZ) " instruction is working

Registers being evaluated are:

R01 R02

----------Example of BGTZ test on register R01-------

If R02 is not greater than to zero

BGTZ instruction failed. Fail flag value 0xFFFFFFFC will be loaded into R14

If R02 is greater than zero

BGTZ instruction worked. Pass flag value held in R06 will be stored into Mem[R15+12]

Jump to location value held in Register

Jumps from the current location in the program to a specified location value held inside of a register

Jumping to location specified in R31

III.C.12 Memory Module 12

```
@0
3c 0f 10 01  // main:       lui  $15, 0x1001
35 ef 00 c0  //             ori  $15, 0x00C0
20 01 ff 8a  //             addi $01, $00, -118
20 02 00 8a  //             addi $02  $00,  138


0c 10 00 1a  //             jal  sltiu_tests



3c 0d ff ff  //             lui  $13, 0xFFFF
35 ad 55 55  //             ori  $13, 0x5555
3c 0c ff ff  //             lui  $12, 0xFFFF
35 8c fa f5  //             ori  $12, 0xFAF5
3c 0b ff ff  //             lui  $11, 0xFFFF
35 6b ff ff  //             ori  $11, 0xFFFF
3c 0a 00 00  //             lui  $10, 0x0000
35 4a f0 f0  //             ori  $10, 0xF0F0


39 a9 aa aa  //             xori $09, $13, 0xAAAA
01 2b 40 22  //             sub  $08, $09, $11


11 00 00 02  //             beq  $08, $00, xor_p1
20 0e ff f9  //             addi $14, $00, -7
00 00 00 0d  //             break


31 87 f5 fa  // xor_p1:     andi $07, $12, 0xF5FA
00 ea 40 22  //             sub  $08, $07, $10


11 00 00 02  //             beq  $08, $00, xor_p2
20 0e ff f8  //             addi $14, $00, -8
00 00 00 0d  //             break


ad e1 00 18  // xor_p2:     sw   $01, 0x18($15)
00 00 00 0d  //             break
00 00 00 0d  //             break


             // sltiu_tests:
2c 23 ff 8b  //             sltiu  $03, $01, -117
14 60 00 02  //             bne    $03, $00, slt1_p1
20 0e ff ff  //             addi   $14, $00, -1
00 00 00 0d  //             break
20 04 00 c0  // slt1 p1:    addi   $04, $00, 0xC0
ad e4 00 00  //             sw     $04, 0x00($15)
```

Loading Registers with initial contents

R15 = 0x100100C0 R01 = 0xFFFFFF8A
R02 = 0x0000008A

Jump to Instruction and save current program counter address

Jumps from the current location in the program to another specified location but saves(links) the current location in the program to register R31.

Loading Registers with initial contents

R13 = 0xFFFF5555 R12 = 0xFFFFFAF5
R11 = 0xFFFFFFFF R10 = 0x0000F0F0

Loading Registers with values produced from arithmetic operations

R09 = R13 XORI 0xAAAA , R08 = R09 SUB R11

Check if the XORI instruction is working

Register being evaluated are: R08

If R08 is not equal to zero

XORI instruction failed. Fail flag value will be loaded into R14

Loadng Registers with value produced from ANDI operations

R07 = R12 ANDI 0xF5FA

Check if the ANDI instruction is working

Register being evaluated are: R08

If R08 is not equal to zero

XORI instruction failed. Fail flag value will be loaded into R14

Loading Memory with Pass Flags The value held in register R15 , with an added offset is used to address a memory location and stores contents from a register into the selected memory location.

If all instructions worked, pass values:

R01 will be stored into memory location M[R15+24]

```
2c 23 ff 89  //            sltiu  $03, $01, -119
10 60 00 02  //            beq    $03, $00, slt_p2
20 0e ff fe  //            addi   $14, $00, -2
00 00 00 0d  //            break
20 05 00 c4  // slt_p2:    addi   $05, $00, 0xC4
ad e5 00 04  //            sw     $05, 0x04($15)



2c 23 ff 8a  //            sltiu  $03, $01, -118
10 60 00 02  //            beq    $03, $00, slt_p3
20 0e ff fd  //            addi   $14, $00, -3
00 00 00 0d  //            break
20 06 00 c8  // slt_p3:    addi   $06, $00, 0xC8
ad e6 00 08  //            sw     $06, 0x08($15)

2c 43 00 8b  //            sltiu  $03, $02, 0x008B
14 60 00 02  //            bne    $03, $00, slt1_p4
20 0e ff fc  //            addi   $14, $00, -4
00 00 00 0d  //            break
20 07 00 cc  // slt1 p4:   addi   $07, $00, 0xCC
ad e7 00 0c  //            sw     $07, 0x0C($15)

2c 43 00 89  //            sltiu  $03, $02, 0x0089
10 60 00 02  //            beq    $03, $00, slt_p5
20 0e ff fb  //            addi   $14, $00, -5
00 00 00 0d  //            break
20 08 00 d0  // slt_p5:    addi   $08, $00, 0xD0
ad e8 00 10  //            sw     $08 0x10($15)

2c 43 00 8a  //            sltiu  $03, $02, 0x008A
10 60 00 02  //            beq    $03, $00, slt_p6
20 0e ff fa  //            addi   $14, $00, -6
00 00 00 0d  //            break
20 06 00 d4  // slt_p6:    addi   $06, $00, 0xD4
ad e6 00 14  //            sw     $06, 0x14($15)
20 0e 00 00  //            addi   $14, $00, 0
      03 e0 00 08  //            jr   $31
```

Check if the SLTI instruction is working

A register and a value are being compared and the result of their comparison is being stored into a selected register

R03 = if (R01< -119) R03 gets 1 , else R10 gets 0

R03 = if (R01< -118) R03 gets 1 , else R10 gets 0

R03 = if (R02< 0xFFFFFF8B) R03 gets 1 , else R10 gets 0

R03 = if (R02< 0xFFFFFF89) R03 gets 1 , else R10 gets 0

R03 = if (R02< 0xFFFFFF8A) R03 gets 1 , else R10 gets 0

--------Example of SLTI test on registers R02 and R01-----

R03 = if (R01< -119) R03 gets 1 , else R10 gets 0

If R03 is not equal to R00, slti instruction failed, load fail flag 0xFFFFFFFF into R14

If R03 is equal to R00, slti instruction worked, load pass flag 0x000000C04 into Memory location Mem[R15+4]

Jump to location value held in Register

Jumps from the current location in the program to a specified location value held inside of a register

Jumping to location specified in R31

III.C.13 Memory Module 13

```
@0
00 00 00 1f  // main:       setie
3c 01 12 34  //             lui  $01, 0x1234
34 21 56 78  //             ori  $01, 0x5678
3c 02 87 65  //             lui  $02, 0x8765
34 42 43 21  //             ori  $02, 0x4321
3c 03 ab cd  //             lui  $03, 0xABCD
34 63 ef 01  //             ori  $03, 0xEF01
3c 04 01 fe  //             lui  $04, 0x01FE
34 84 dc ba  //             ori  $04, 0xDCBA
3c 05 5a 5a  //             lui  $05, 0x5A5A
34 a5 5a 5a  //             ori  $05, 0x5A5A
3c 06 ff ff  //             lui  $06, 0xFFFF
34 c6 ff ff  //             ori  $06, 0xFFFF
3c 07 ff ff  //             lui  $07, 0xFFFF
34 e7 ff 00  //             ori  $07, 0xFF00

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
ac ef 00 00  //             sw   $15, 0($07)
00 00 00 0d  //             break

@200
```

Enable Interrupts

Loading Registers with initial contents

R01 = 0x12345678
R02 = 0x87654321
R03 = 0xABCDEG01
R04 = 0x01FEDCBA
R05 = 0x5A5A5A5A
R06 = 0xFFFFFFFF
R07 = 0xFFFFFF00

Loading Registers with values produced from arithmetic operations

R09 = R06 + R08 R08 = R06 + R07

R10 = R06 + R09 R11 = R06 + R10

R12 = R06 + R11 R13 = R06 + R12

R14 = R06 + R13 R15 = R06 + R14

Loading Register with initial contents

R13 = 0x100103F0

Load Memory with register value

Store value held in R15 into memory location specified by R07

Mem[R07] = R15

Beginning of ISR

```
3c 10 10 01  // isr:        lui    $16, 0x1001
36 10 00 c0  //             ori    $16, 0x00C0
3c 11 80 00  //             lui    $17, 0x8000
36 31 ff ff  //             ori    $17, 0xFFFF
20 12 00 10  //             addi   $18, $0, 0x10

76 11 00 00  // out_IO:     output $17, 0($16)
00 11 88 43  //             sra    $17, $17, 1
22 10 00 04  //             addi   $16, $16, 4
22 52 ff ff  //             addi   $18, $18, -1
16 40 ff fb  //             bne    $18, $00, out_IO
```

Loading Registers with initial contents

R16 = 0x100100C0 R17 = 0x00C08000

R18 = 0x00000010

Loop that outputs into the I/O Memory

The value held in register R16 , with an added offset of 0, is used to address I/0 memory location and output contents of R17 into the specified I/0 memory location

The loop performs the following:

1. Output value held at R17 into I/0 location specified by R16
2. Shift right arithmetic R17 once
3. Increment value held in R16 by 4
4. Decrement value held in R18 by 1
5. Repeats steps 1 to 5 if value in R18 doesn't equal 0.

Once **R18 equals 0** , the loop will break

```
3c 10 10 01  //          lui    $16, 0x1001
36 10 00 c0  //          ori    $16, 0x00C0
72 13 00 00  //          input  $19,  0($16)
72 14 00 04  //          input  $20,  4($16)
72 15 00 08  //          input  $21,  8($16)
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
```

Loading Registers with initial contents

R16 = 0x100100C0

Inputs from the I/O Memory to a Register

The value held in register R16 , with an added offset , is used to address I/0 memory location and inputs at the specified I/O memory location into a register.

R19 = I_0_Mem[R16]

R20 = I_0_Mem[R16+4]

R21 = I_0_Mem[R16+8]

R22 = I_0_Mem[R16+12]

R23 = I_0_Mem[R16+16]

R24 = I_0_Mem[R16+20]

Jump to location value held in Register

Jumps from the current location in the program to a specified location value held inside of a register

Jumping to location specified in R31

III.C.14 Memory Module 14

```
@0
00 00 00 1f  // main:       setie
3c 01 12 34  //             lui  $01, 0x1234
34 21 56 78  //             ori  $01, 0x5678
3c 02 87 65  //             lui  $02, 0x8765
34 42 43 21  //             ori  $02, 0x4321
3c 03 ab cd  //             lui  $03, 0xABCD
34 63 ef 01  //             ori  $03, 0xEF01
3c 04 01 fe  //             lui  $04, 0x01FE
34 84 dc ba  //             ori  $04, 0xDCBA
3c 05 5a 5a  //             lui  $05, 0x5A5A
34 a5 5a 5a  //             ori  $05, 0x5A5A
3c 06 ff ff  //             lui  $06, 0xFFFF
34 c6 ff ff  //             ori  $06, 0xFFFF
3c 07 ff ff  //             lui  $07, 0xFFFF
34 e7 ff 00  //             ori  $07, 0xFF00

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
ac ef 00 00  //             sw   $15, 0($07)
00 00 00 0d  //             break

@200

3c 10 10 01  // isr:        lui    $16, 0x1001
36 10 00 c0  //             ori    $16, 0x00C0
3c 11 80 00  //             lui    $17, 0x8000
36 31 ff ff  //             ori    $17, 0xFFFF
20 12 00 10  //             addi   $18, $0, 0x10

76 11 00 00  // out IO:     output $17, 0($16)
00 11 88 43  //             sra    $17, $17, 1
22 10 00 04  //             addi   $16, $16, 4
22 52 ff ff  //             addi   $18, $18, -1
16 40 ff fb  //             bne    $18, $00, out_IO
```

Enable Interrupts

Loading Registers with initial contents

R01 = 0x12345678
R02 = 0x87654321
R03 = 0xABCDEG01
R04 = 0x01FEDCBA
R05 = 0x5A5A5A5A
R06 = 0xFFFFFFFF
R07 = 0xFFFFFF00

Loading Registers with values produced from arithmetic operations

R09 = R06 + R08 R08 = R06 + R07

R10 = R06 + R09 R11 = R06 + R10

R12 = R06 + R11 R13 = R06 + R12

R14 = R06 + R13 R15 = R06 + R14

Loading Register with initial contents

R13 = 0x100103F0

Load Memory with register value

Store value held in R15 into memory location specified by R07

Mem[R07] = R15

Beginning of ISR

Loading Registers with initial contents

R16 = 0x100100C0 R17 = 0x00C08000

R18 = 0x00000010

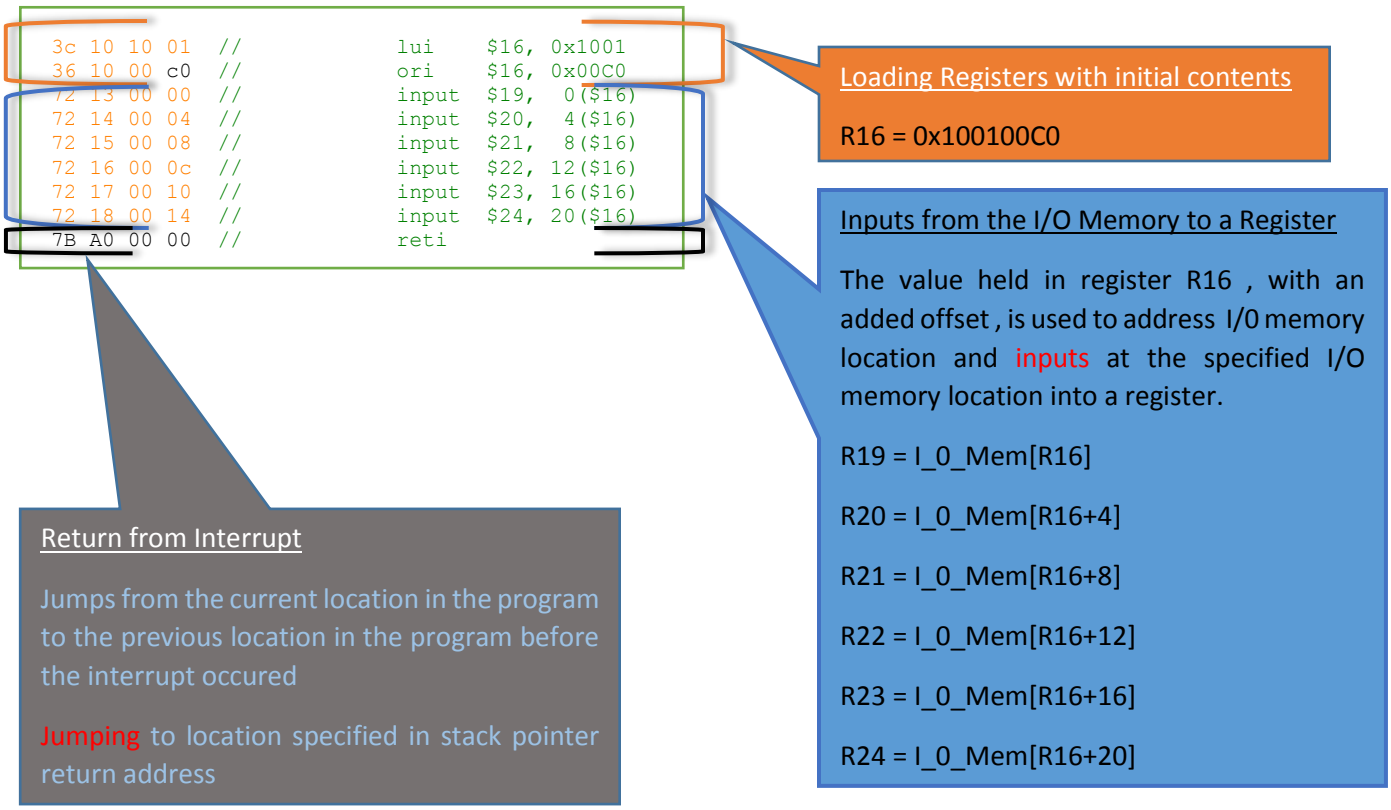
Loop that outputs into the I/O Memory

The value held in register R16 , with an added offset of 0, is used to address I/0 memory location and output contents of R17 into the specified I/0 memory location

The loop performs the following:

6. Output value held at R17 into I/0 location specified by R16
7. Shift right arithmetic R17 once
8. Increment value held in R16 by 4
9. Decrement value held in R18 by 1
10. Repeats steps 1 to 5 if value in R18 doesn't equal 0.

Once **R18 equals 0** , the loop will break


3c 10 10 01 // lui $16, 0x1001
36 10 00 c0 // ori $16, 0x00C0
72 13 00 00 // input $19, 0($16)
72 14 00 04 // input $20, 4($16)
72 15 00 08 // input $21, 8($16)
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
Loading Registers with initial contents
R16 = 0x100100C0
Inputs from the I/O Memory to a Register
The value held in register R16 , with an added offset , is used to address I/0 memory location and inputs at the specified I/O memory location into a register.
R19 = I_0_Mem[R16]
R20 = I_0_Mem[R16+4]
R21 = I_0_Mem[R16+8]
R22 = I_0_Mem[R16+12]
R23 = I_0_Mem[R16+16]
R24 = I_0_Mem[R16+20]
Return from Interrupt
Jumps from the current location in the program to the previous location in the program before the interrupt occured
Jumping to location specified in stack pointer return address

III.C.15 Enhanced Module

```
@0
00 00 00 1f  // main:       setie

3c 01 12 34  //             lui  $01, 0x1234
34 21 56 78  //             ori  $01, 0x5678
3c 02 87 65  //             lui  $02, 0x8765
34 42 43 21  //             ori  $02, 0x4321
3c 03 ab cd  //             lui  $03, 0xABCD
34 63 ef 01  //             ori  $03, 0xEF01
3c 04 01 fe  //             lui  $04, 0x01FE
34 84 dc ba  //             ori  $04, 0xDCBA
3c 05 5a 5a  //             lui  $05, 0x5A5A
34 a5 5a 5a  //             ori  $05, 0x5A5A
3c 06 ff ff  //             lui  $06, 0xFFFF
34 c6 ff ff  //             ori  $06, 0xFFFF
3c 07 ff ff  //             lui  $07, 0xFFFF
34 e7 ff 00  //             ori  $07, 0xFF00

7c c3 50 c0 //              BSRR $10,#07,#3
7c 8A 58 c0 //              BSLR $11,#10,#3
7c 27 00 16 //              BGEZ $07, fail_BGEZ
7c 20 00 01 //              BGEZ $00, pass_BGEZ
00 00 00 0d //              break


7c 00 00 16  // pass_BGEZ: BLTZ $01,fail_BLTZ
7c 07 00 01  //             BLTZ $07,pass_BLTZ
00 00 00 0d  //             break

00 c7 48 22 // pass_BLTZ:  sub $09,$06,$07

7c 60 00 00 //              no_op
7c 60 00 00 //              no_op
7c 60 00 00 //              no_op


7d 20 00 2e //              JC fail_jc
3c 0f f0 f0 // pass_JC :   lui $15, 0xF0F0
35 ef f0 f0 //              ori $15, 0xF0F0
7d 80 00 31 //              JV fail_jv

3c 10 70 A0 // pass_JV :   lui $16, 0x70A0
36 10 70 A0 //              ori $16, 0x70A0
7d 60 00 34 //              JN fail_jn

3c 11 B0 B0 // pass_JN :   lui $17, 0xB0B0
36 31 B0 B0 //              ori $17, 0xB0B0
7d 40 00 36 /               JZ fail_jz
3c 18 C0 C0 // pass_JZ :   lui $24, 0xC0C0
37 18 C0 C0 //              ori $24, 0xC0C0
00 00 00 0d  //             break



3c 0e ff ff  // fail_BGEZ: lui $14, 0xFFFF
35 ce ff ff  //             ori  $14, 0xFFFF
00 00 00 0d  //             break
3c 0e ff ff  // fail_BLTZ  lui  $14, 0xFFFF
35 ce ff fe  //             ori  $14, 0xFFFE
00 00 00 0d  //             break
3c 0e ff ff  // fail_JC:   lui  $14, 0xFFFF
```

Enable Interrupts

Loading Registers with initial contents

R01 = 0x12345678 R02 = 0x87654321
R03 = 0xABCDEF01 R04 = 0x01FEDCBA
R05 = 0x5A5A5A5A R06 = 0xFFFFFFFF
R07 = 0xFFFFFF00

Shifting left and right with rotate

R10 ← R07 >> rot 3 R11 ← R07 << rot 3

Testing Branch If Greater Than or Equal to Zero

R07 < 0 so should not branch

R00 => 0, so it should branch

Testing Branch If Greater Than Zero

R01 > 0 , so should not branch

R07 < 0 , so should branch

Success flag for all enhanced branches

No operations to b performed

Testing the jump if status flags and their respective success flags:

Since the most recent operation or R09 ← R06 – R07 did not produce a carry, negative number, zero or overflow, all the following success flags should be loaded to the corresponding jump if flag type.

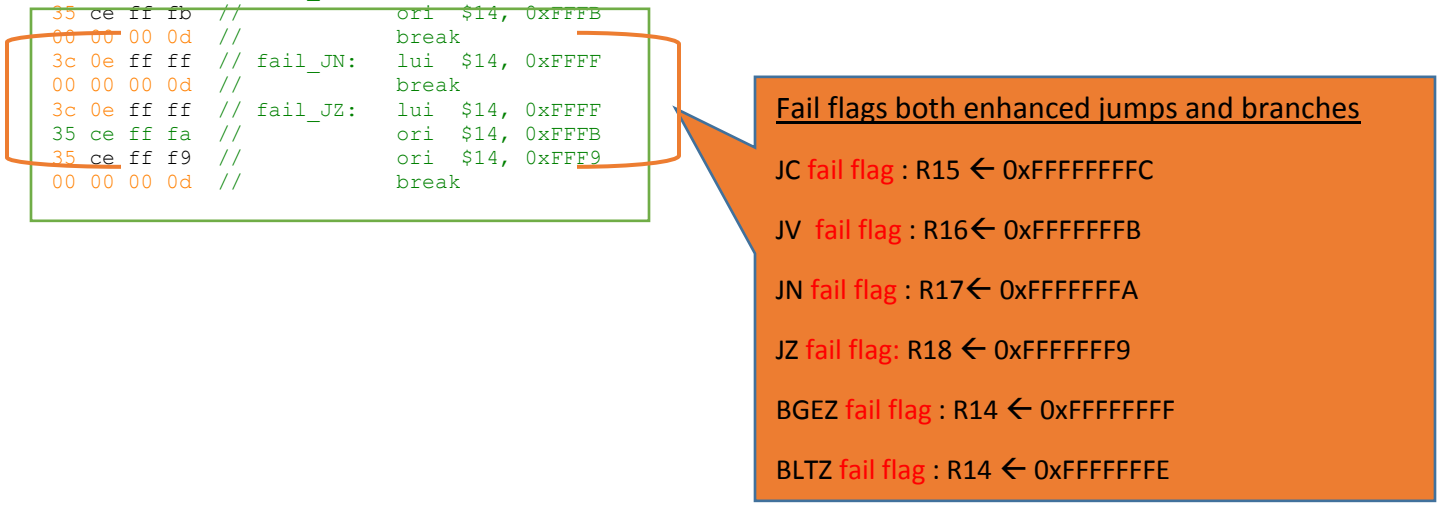
JC Success flag : R15 ← 0xF0F0F0F0

JV Success Flag: R16← 0x70A0&0A0

JN Success Flag : R17← 0xB0B0B0B0

JZ Success Flag: R18 ← 0xC0C0C0C0

```
35 ce ff fc  //             ori  $14, 0xFFFC        # $r14 <-- 0xFFFFFFFC  (Fail flag 3 JC)   BC
00 00 00 0d  //             break
3c 0e ff ff  // fail_JV:    lui  $14, 0xFFFF
35 ce ff fb  //             ori  $14, 0xFFFB
00 00 00 0d  //             break
3c 0e ff ff  // fail_JN:    lui  $14, 0xFFFF
00 00 00 0d  //             break
3c 0e ff ff  // fail_JZ:    lui  $14, 0xFFFF
35 ce ff fa  //             ori  $14, 0xFFFB
35 ce ff f9  //             ori  $14, 0xFFF9
00 00 00 0d  //             break
```



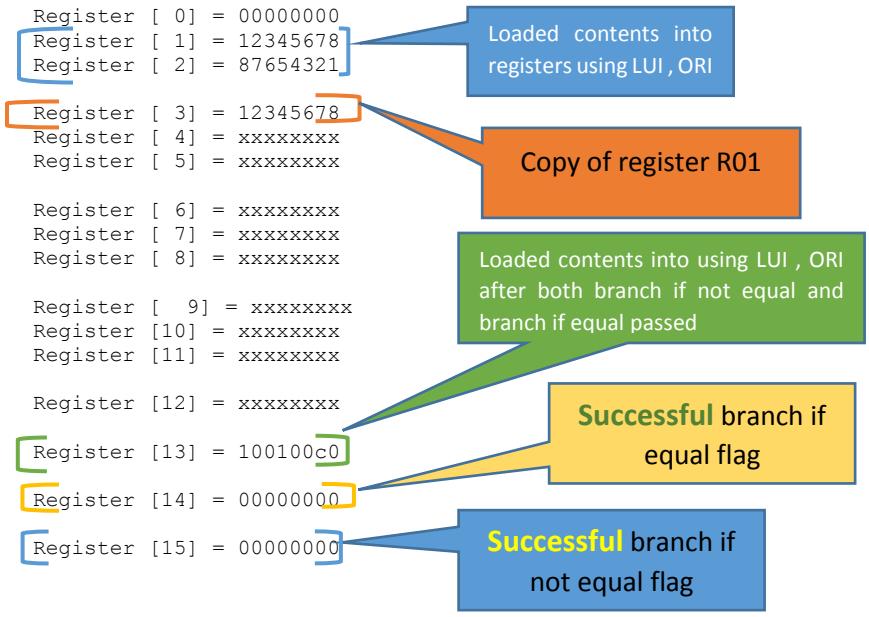
III.D Annotated Verilog Log Files

The following are the verification log files of Memory Module 1 to 14.

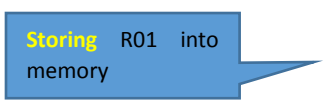
III.D.1 Log File 1

==========Register Dump==========

```
Register [ 0] = 00000000
Register [ 1] = 12345678
Register [ 2] = 87654321

Register [ 3] = 12345678
Register [ 4] = xxxxxxxx
Register [ 5] = xxxxxxxx

Register [ 6] = xxxxxxxx
Register [ 7] = xxxxxxxx
Register [ 8] = xxxxxxxx

Register [  9] = xxxxxxxx
Register [10] = xxxxxxxx
Register [11] = xxxxxxxx

Register [12] = xxxxxxxx

Register [13] = 100100c0

Register [14] = 00000000

Register [15] = 00000000
```



==========Memory Dump==========




```
Memory [0c0] = 12       Memory [0dd] = xx   Memory [0fa] = xx
Memory [0c1] = 34       Memory [0de] = xx   Memory [0fb] = xx
Memory [0c2] = 56       Memory [0df] = xx   Memory [0fc] = xx
Memory [0c3] = 78       Memory [0e0] = xx   Memory [0fd] = xx
Memory [0c4] = xx       Memory [0e1] = xx   Memory [0fe] = xx
Memory [0c5] = xx       Memory [0e2] = xx   Memory [0ff] = xx
Memory [0c6] = xx       Memory [0e3] = xx
Memory [0c7] = xx       Memory [0e4] = xx
Memory [0c8] = xx       Memory [0e5] = xx
Memory [0c9] = xx       Memory [0e6] = xx
Memory [0ca] = xx       Memory [0e7] = xx
Memory [0cb] = xx       Memory [0e8] = xx
Memory [0cc] = xx       Memory [0e9] = xx
Memory [0cd] = xx       Memory [0ea] = xx
Memory [0ce] = xx       Memory [0eb] = xx
Memory [0cf] = xx       Memory [0ec] = xx
Memory [0d0] = xx       Memory [0ed] = xx
Memory [0d1] = xx       Memory [0ee] = xx
Memory [0d2] = xx       Memory [0ef] = xx
Memory [0d3] = xx       Memory [0f0] = xx
Memory [0d4] = xx       Memory [0f1] = xx
Memory [0d5] = xx       Memory [0f2] = xx
Memory [0d6] = xx       Memory [0f3] = xx
Memory [0d7] = xx       Memory [0f4] = xx
Memory [0d8] = xx       Memory [0f5] = xx
Memory [0d9] = xx       Memory [0f6] = xx
Memory [0da] = xx       Memory [0f7] = xx
Memory [0db] = xx       Memory [0f8] = xx
Memory [0dc] = xx       Memory [0f9] = xx
```

III.D.2 Log File 2

```
==========================Register Dump==============================

Register[ 0]= 00000000
Register[ 1]= 0000ffff
Register[ 2]= 00000000

Register[ 3] = xxxxxxxx
Register[ 4] = xxxxxxxx
Register[ 5] = xxxxxxxx
Register[ 6] = xxxxxxxx
Register[ 7] = xxxxxxxx
Register[ 8] = xxxxxxxx
Register[ 9] = xxxxxxxx
Register[ 10]= xxxxxxxx
Register[ 11]= xxxxxxxx
Register[ 12]= xxxxxxxx
Register[ 13]= xxxxxxxx

Register[ 14]= 5a5a3c3c

Register[ 15]= 10010100
```

Contents of R01 are shifted right logical 16 times

Contents loaded using lui, ori after we jump to exit label.

Final Value After incrementing by 4, 16 times through the loop.

Storing R01 into memory after 1st loop

Storing R01 into memory

Big Endian Format

```
==========================Memory Dump==============================

Memory [0c0] = 7f      Memory [0dc] = 00     Memory [0f8] = 00
Memory [0c1] = ff      Memory [0dd] = ff     Memory [0f9] = 01
Memory [0c2] = ff      Memory [0de] = ff     Memory [0fa] = ff
Memory [0c3] = ff      Memory [0df] = ff     Memory [0fb] = ff

Memory [0c4] = 3f      Memory [0e0] = 00     Memory [0fc] = 00
Memory [0c5] = ff      Memory [0e1] = 7f     Memory [0fd] = 00
Memory [0c6] = ff      Memory [0e2] = ff     Memory [0fe] = ff
Memory [0c7] = ff      Memory [0e3] = ff     Memory [0ff] = ff

Memory [0c8] = 1f      Memory [0e4] = 00
Memory [0c9] = ff      Memory [0e5] = 3f
Memory [0ca] = ff      Memory [0e6] = ff
Memory [0cb] = ff      Memory [0e7] = ff

Memory [0cc] = 0f      Memory [0e8] = 00
Memory [0cd] = ff      Memory [0e9] = 1f
Memory [0ce] = ff      Memory [0ea] = ff
Memory [0cf] = ff      Memory [0eb] = ff

Memory [0d0] = 07      Memory [0ec] = 00
Memory [0d1] = ff      Memory [0ed] = 0f
Memory [0d2] = ff      Memory [0ee] = ff
Memory [0d3] = ff      Memory [0ef] = ff

Memory [0d4] = 03      Memory [0f0] = 00
Memory [0d5] = ff      Memory [0f1] = 07
Memory [0d6] = ff      Memory [0f2] = ff
Memory [0d7] = ff      Memory [0f3] = ff

Memory [0d8] = 01      Memory [0f4] = 00
Memory [0d9] = ff      Memory [0f5] = 03
Memory [0da] = ff      Memory [00f6] = ff
Memory [0db] = ff      Memory [00f7] = ff
```

R01 is stored in memory by incrementing 4 bytes each loop until bne is false and breaks out of loop. Data is stored in Big Endian Format.

III.D.3 Log File 3

```
==========================Register Dump================================
Register[ 0]= 00000000
Register[ 1]= ffff8000

Register[ 2]= 00000000

Register[ 3] = xxxxxxxx
Register[ 4] = xxxxxxxx
Register[ 5] = xxxxxxxx
Register[ 6] = xxxxxxxx
Register[ 7] = xxxxxxxx
Register[ 8] = xxxxxxxx
Register[ 9] = xxxxxxxx
Register[ 10]= xxxxxxxx
Register[ 11]= xxxxxxxx
Register[ 12]= xxxxxxxx
Register[ 13]= xxxxxxxx

Register[ 14]= 5a5a3c3c

Register[ 15]= 10010100

==========================Memory Dump================================
```

Contents of R01 are shifted right logical 16 times

Final value after decrementing 16 times through loop.

Contents loaded using lui, ori after jump to exit label.

Final Contents of R15 after 16 loops in top label. Break after loading contents in R14 with lui, ori instruction.

Store data starting at @c0, big Endian. Increment 4 bytes after storing word.

```
Memory [0c0] = c0      Memory [0dc] = ff     Memory [0f8] = ff
Memory [0c1] = 00      Memory [0dd] = 80     Memory [0f9] = ff
Memory [0c2] = 7f      Memory [0de] = 00     Memory [0fa] = 00
Memory [0c3] = ff      Memory [0df] = ff     Memory [0fb] = 01

Memory [0c4] = e0      Memory [0e0] = ff     Memory [0fc] = ff
Memory [0c5] = 00      Memory [0e1] = c0     Memory [0fd] = ff
Memory [0c6] = 3f      Memory [0e2] = 00     Memory [0fe] = 80
Memory [0c7] = ff      Memory [0e3] = 7f     Memory [0ff] = 00

Memory [0c8] = f0      Memory [0e4] = ff
Memory [0c9] = 00      Memory [0e5] = e0
Memory [0ca] = 1f      Memory [0e6] = 00
Memory [0cb] = ff      Memory [0e7] = 3f

Memory [0cc] = f8      Memory [0e8] = ff
Memory [0cd] = 00      Memory [0e9] = f0
Memory [0ce] = 0f      Memory [0ea] = 00
Memory [0cf] = ff      Memory [0eb] = 1f

Memory [0d0] = fc      Memory [0ec] = ff
Memory [0d1] = 00      Memory [0ed] = f8
Memory [0d2] = 07      Memory [0ee] = 00
Memory [0d3] = ff      Memory [0ef] = 0f

Memory [0d4] = fe      Memory [0f0] = ff
Memory [0d5] = 00      Memory [0f1] = fc
Memory [0d6] = 03      Memory [0f2] = 00
Memory [0d7] = ff      Memory [0f3] = 07

Memory [0d8] = ff      Memory [0f4] = ff
Memory [0d9] = 00      Memory [0f5] = fe
Memory [0da] = 01      Memory [0f6] = 00
Memory [0db] = ff      Memory [0f7] = 03
```

Continue storing data @ every 4 bytes per address until bne is false. Data is shifted logical left once per loop(16 times total). Final Contents stored Big Endian.

III.D.4 Log File 4

```
==========================Register Dump==============================
Register[ 0]= 00000000
Register[ 1]= ffff0000

Register[ 2]= 00000000

Register[ 3]= 00000000

Register[ 4]= xxxxxxxx
Register[ 5]= xxxxxxxx
Register[ 6]= xxxxxxxx
Register[ 7]= xxxxxxxx
Register[ 8]= xxxxxxxx
Register[ 9]= xxxxxxxx
Register[ 10]= xxxxxxxx
Register[ 11]= xxxxxxxx
Register[ 12]= xxxxxxxx
Register[ 13]= xxxxxxxx

Register[ 14]= 5a5a3c3c

Register[ 15]= 10010100

==========================Memory Dump==============================
```

Final contents of register R01 starts off with a value of 0xFFFF_FFFF, after logical shifting left 1 bit 16 times through the loop.

Value starts off as 0x10h, after decrementing by -1 through the loop 16 times, bne instruction becomes false, fall through to jump instruction.

R03 is loaded with a value of 0 after false slt instruction. Fall through to jump instruction.

R14 is loaded using lui, ori instruction @exit label.

Final contents of R15 after looping 16 times.

Store data starting at @c0, big Endian. Increment 4 bytes after storing word.

```
Memory [0c0] = ff      Memory [0dc] = ff     Memory [0f8] = ff
Memory [0c1] = ff      Memory [0dd] = ff     Memory [0f9] = ff
Memory [0c2] = ff      Memory [0de] = ff     Memory [0fa] = 80
Memory [0c3] = fe      Memory [0df] = 00     Memory [0fb] = 00

Memory [0c4] = ff      Memory [0e0] = ff     Memory [0fc] = ff
Memory [0c5] = ff      Memory [0e1] = ff     Memory [0fd] = ff
Memory [0c6] = ff      Memory [0e2] = fe     Memory [0fe] = 00
Memory [0c7] = fc      Memory [0e3] = 00     Memory [0ff] = 00

Memory [0c8] = ff      Memory [0e4] = ff
Memory [0c9] = ff      Memory [0e5] = ff
Memory [0ca] = ff      Memory [0e6] = fc
Memory [0cb] = f8      Memory [0e7] = 00

Memory [0cc] = ff      Memory [0e8] = ff
Memory [0cd] = ff      Memory [0e9] = ff
Memory [0ce] = ff      Memory [0ea] = f8
Memory [0cf] = f0      Memory [0eb] = 00

Memory [0d0] = ff      Memory [0ec] = ff
Memory [0d1] = ff      Memory [0ed] = ff
Memory [0d2] = ff      Memory [0ee] = f0
Memory [0d3] = e0      Memory [0ef] = 00

Memory [0d4] = ff      Memory [0f0] = ff
Memory [0d5] = ff      Memory [0f1] = ff
Memory [0d6] = ff      Memory [0f2] = e0
Memory [0d7] = c0      Memory [0f3] = 00

Memory [0d8] = ff      Memory [0f4] = ff
Memory [0d9] = ff      Memory [0f5] = ff
Memory [0da] = ff      Memory [0f6] = c0
Memory [0db] = 80      Memory [0f7] = 00
```

Continue storing data @ every 4 bytes per address until bne is false. Data is shifted logical left once per loop(16 times total). Final Contents stored Big Endian.

III.D.5 Log File 5

```
==========================Register Dump==============================
Register[ 0]= 00000000

Register[ 1]= ffff0000

Register[ 2]= 00000000

Register[ 3]= 00000000

Register[ 4]= xxxxxxxx
Register[ 5]= xxxxxxxx
Register[ 6]= xxxxxxxx
Register[ 7]= xxxxxxxx
Register[ 8]= xxxxxxxx
Register[ 9]= xxxxxxxx
Register[ 10]= xxxxxxxx
Register[ 11]= xxxxxxxx
Register[ 12]= xxxxxxxx
Register[ 13]= xxxxxxxx

Register[ 14]= 5a5a3c3c

Register[ 15]= 10010100

==========================Memory Dump==============================

Memory [0c0] = ff      Memory [0dc] = ff     Memory [0f8] = ff
Memory [0c1] = ff      Memory [0dd] = ff     Memory [0f9] = ff
Memory [0c2] = ff      Memory [0de] = ff     Memory [0fa] = 80
Memory [0c3] = fe      Memory [0df] = 00     Memory [0fb] = 00

Memory [0c4] = ff      Memory [0e0] = ff     Memory [0fc] = ff
Memory [0c5] = ff      Memory [0e1] = ff     Memory [0fd] = ff
Memory [0c6] = ff      Memory [0e2] = fe     Memory [0fe] = 00
Memory [0c7] = fc      Memory [0e3] = 00     Memory [0ff] = 00

Memory [0c8] = ff      Memory [0e4] = ff
Memory [0c9] = ff      Memory [0e5] = ff
Memory [0ca] = ff      Memory [0e6] = fc
Memory [0cb] = f8      Memory [0e7] = 00

Memory [0cc] = ff      Memory [0e8] = ff
Memory [0cd] = ff      Memory [0e9] = ff
Memory [0ce] = ff      Memory [0ea] = f8
Memory [0cf] = f0      Memory [0eb] = 00

Memory [0d0] = ff      Memory [0ec] = ff
Memory [0d1] = ff      Memory [0ed] = ff
Memory [0d2] = ff      Memory [0ee] = f0
Memory [0d3] = e0      Memory [0ef] = 00

Memory [0d4] = ff      Memory [0f0] = ff
Memory [0d5] = ff      Memory [0f1] = ff
Memory [0d6] = ff      Memory [0f2] = e0
Memory [0d7] = c0      Memory [0f3] = 00

Memory [0d8] = ff      Memory [0f4] = ff
Memory [0d9] = ff      Memory [0f5] = ff
Memory [0da] = ff      Memory [0f6] = c0
Memory [0db] = 80      Memory [0f7] = 00
```

R0 added with (-16) to load into R02

Final R01 value shifted 16 times to the left logical.

Value starts at -16. After adding +1(16 times), final value is 0x0.

R03 receives a '0' since R02 is == 0 in slt instruction. Bne instruction is false, falling through @jump instruction.

@exit label load R14 using lui, ori.

Final contents of R15 after break.

Store data starting at @c0, big Endian. Increment 4 bytes after storing word.

Continue storing data @ every 4 bytes per address until bne is false. Data is shifted logical left once per loop (16 times total). Final Contents stored Big Endian.

III.D.6 Log File 6

```
==========================Register Dump==============================

Register[ 0]= 00000000

Register[ 1]= 12345678
Register[ 2]= 89abcdef
Register[ 3]= a5a5a5a5
Register[ 4]= 5a5a5a5a
Register[ 5]= 2468ace0
Register[ 6]= 13579bdf
Register[ 7]= 0f0f0f0f
Register[ 8]= f0f0f0f0
Register[ 9]= 00000009
Register[ 10]= 0000000a
Register[ 11]= 0000000b
Register[ 12]= 0000000c

Register[ 13]= 00000000

Register[ 14]= 10010100
Register[ 15]= 10010040
```

$zero register to compare in bne instruction.

Loaded values to register R01 to R12 from mem. Location 04(R15) to 48(R15).

Value initiates as 16, after decrementing 16 times through the loop, value ends as 0x0

Final Contents of R14 & R15 after bne fails and fall through to break point.

```
==========================Memory Dump==============================

Memory [0c0] = c3      Memory [0dc] = 0f     Memory [0f8] = ff
Memory [0c1] = c3      Memory [0dd] = 0f     Memory [0f9] = ff
Memory [0c2] = c3      Memory [0de] = 0f     Memory [0fa] = ff
Memory [0c3] = c3      Memory [0df] = 0f     Memory [0fb] = f8

Memory [0c4] = 12      Memory [0e0] = f0     Memory [0fc] = 00
Memory [0c5] = 34      Memory [0e1] = f0     Memory [0fd] = 00
Memory [0c6] = 56      Memory [0e2] = f0     Memory [0fe] = 75
Memory [0c7] = 78      Memory [0e3] = f0     Memory [0ff] = cc

Memory [0c8] = 89      Memory [0e4] = 00
Memory [0c9] = ab      Memory [0e5] = 00
Memory [0ca] = cd      Memory [0e6] = 00
Memory [0cb] = ef      Memory [0e7] = 09

Memory [0cc] = a5      Memory [0e8] = 00
Memory [0cd] = a5      Memory [0e9] = 00
Memory [0ce] = a5      Memory [0ea] = 00
Memory [0cf] = a5      Memory [0eb] = 0a

Memory [0d0] = 5a      Memory [0ec] = 00
Memory [0d1] = 5a      Memory [0ed] = 00
Memory [0d2] = 5a      Memory [0ee] = 00
Memory [0d3] = 5a      Memory [0ef] = 0b

Memory [0d4] = 24      Memory [0f0] = 00
Memory [0d5] = 68      Memory [0f1] = 00
Memory [0d6] = ac      Memory [0f2] = 00
Memory [0d7] = e0      Memory [0f3] = 0c

Memory [0d8] = 13      Memory [0f4] = 00
Memory [0d9] = 57      Memory [0f5] = 00
Memory [0da] = 9b      Memory [0f6] = 00
Memory [0db] = df      Memory [0f7] = 0d
```

Store data starting at @c0, big Endian. Increment 4 bytes after storing word.

Final values before bne is false and falls through to break point. Values are stored every 4 bytes, big Endian style.

III.D.7 Log File 7

```
==========================Register Dump==============================

Register[ 0]= 00000000

Register[ 1]= 12345678
Register[ 2]= 89abcdef
Register[ 3]= a5a5a5a5
Register[ 4]= 5a5a5a5a
Register[ 5]= 2468ace0
Register[ 6]= 13579bdf
Register[ 7]= 0f0f0f0f
Register[ 8]= f0f0f0f0
Register[ 9]= 00000009
Register[ 10]= 0000000a
Register[ 11]= 0000000b
Register[ 12]= 0000000c

Register[ 13]= 00000000

Register[ 14]= 10010100

Register[ 15]= ffffffff

==========================Memory Dump==============================

Memory [0c0] = c3      Memory [0dc] = 0f    Memory [0f8] = ff
Memory [0c1] = c3      Memory [0dd] = 0f    Memory [0f9] = ff
Memory [0c2] = c3      Memory [0de] = 0f    Memory [0fa] = ff
Memory [0c3] = c3      Memory [0df] = 0f    Memory [0fb] = f8

Memory [0c4] = 12      Memory [0e0] = f0    Memory [0fc] = 00
Memory [0c5] = 34      Memory [0e1] = f0    Memory [0fd] = 00
Memory [0c6] = 56      Memory [0e2] = f0    Memory [0fe] = 75
Memory [0c7] = 78      Memory [0e3] = f0    Memory [0ff] = cc

Memory [0c8] = 89      Memory [0e4] = 00
Memory [0c9] = ab      Memory [0e5] = 00
Memory [0ca] = cd      Memory [0e6] = 00
Memory [0cb] = ef      Memory [0e7] = 09

Memory [0cc] = a5      Memory [0e8] = 00
Memory [0cd] = a5      Memory [0e9] = 00
Memory [0ce] = a5      Memory [0ea] = 00
Memory [0cf] = a5      Memory [0eb] = 0a

Memory [0d0] = 5a      Memory [0ec] = 00
Memory [0d1] = 5a      Memory [0ed] = 00
Memory [0d2] = 5a      Memory [0ee] = 00
Memory [0d3] = 5a      Memory [0ef] = 0b

Memory [0d4] = 24      Memory [0f0] = 00
Memory [0d5] = 68      Memory [0f1] = 00
Memory [0d6] = ac      Memory [0f2] = 00
Memory [0d7] = e0      Memory [0f3] = 0c

Memory [0d8] = 13      Memory [0f4] = 00
Memory [0d9] = 57      Memory [0f5] = 00
Memory [0da] = 9b      Memory [0f6] = 00
Memory [0db] = df      Memory [0f7] = 0d
```

$zero register to compare in bne instruction.

Loaded values to register R01 to R12 from mem. Location 04(R15) to 48(R15).

Value initiates as 16, after decrementing 16 times through the loop, value ends as 0x0

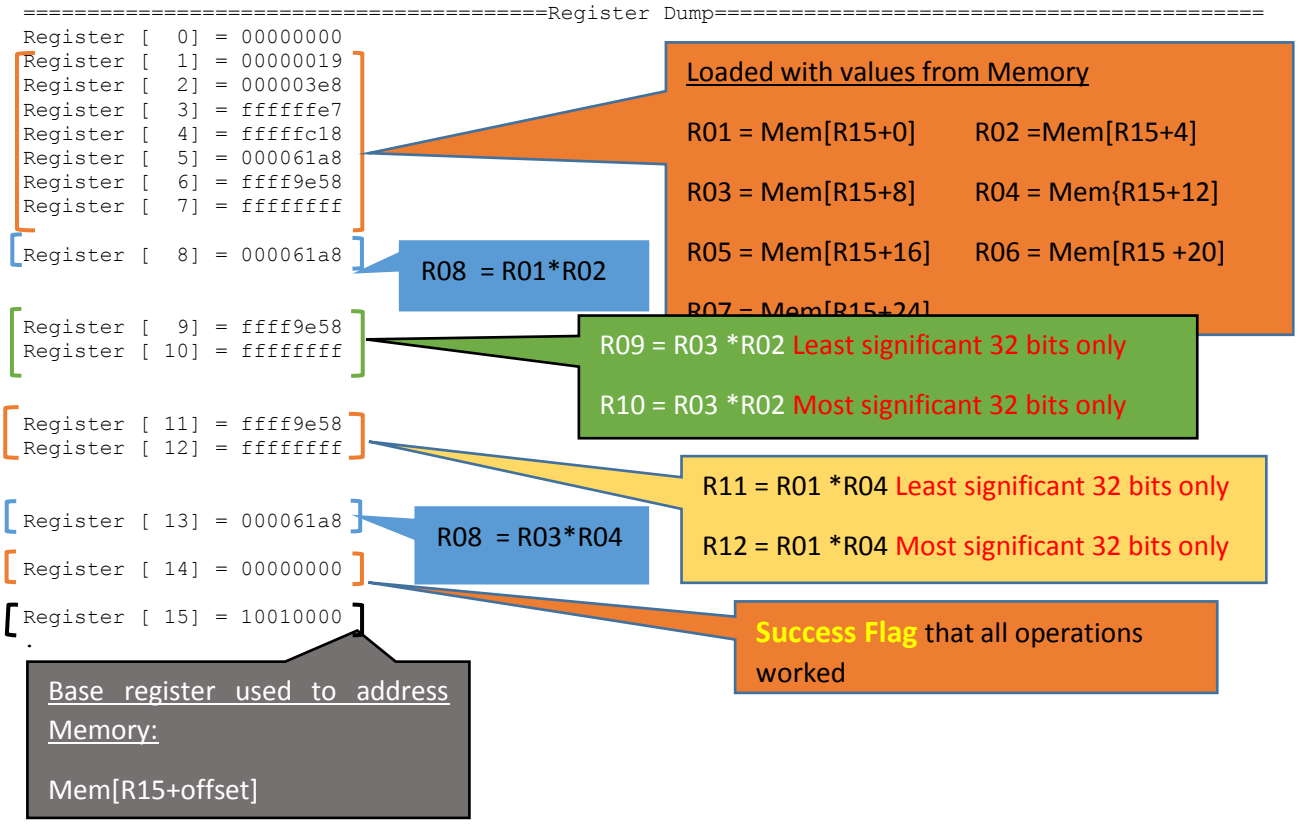
Final Contents of R14 after bne fails and fall through to break point.

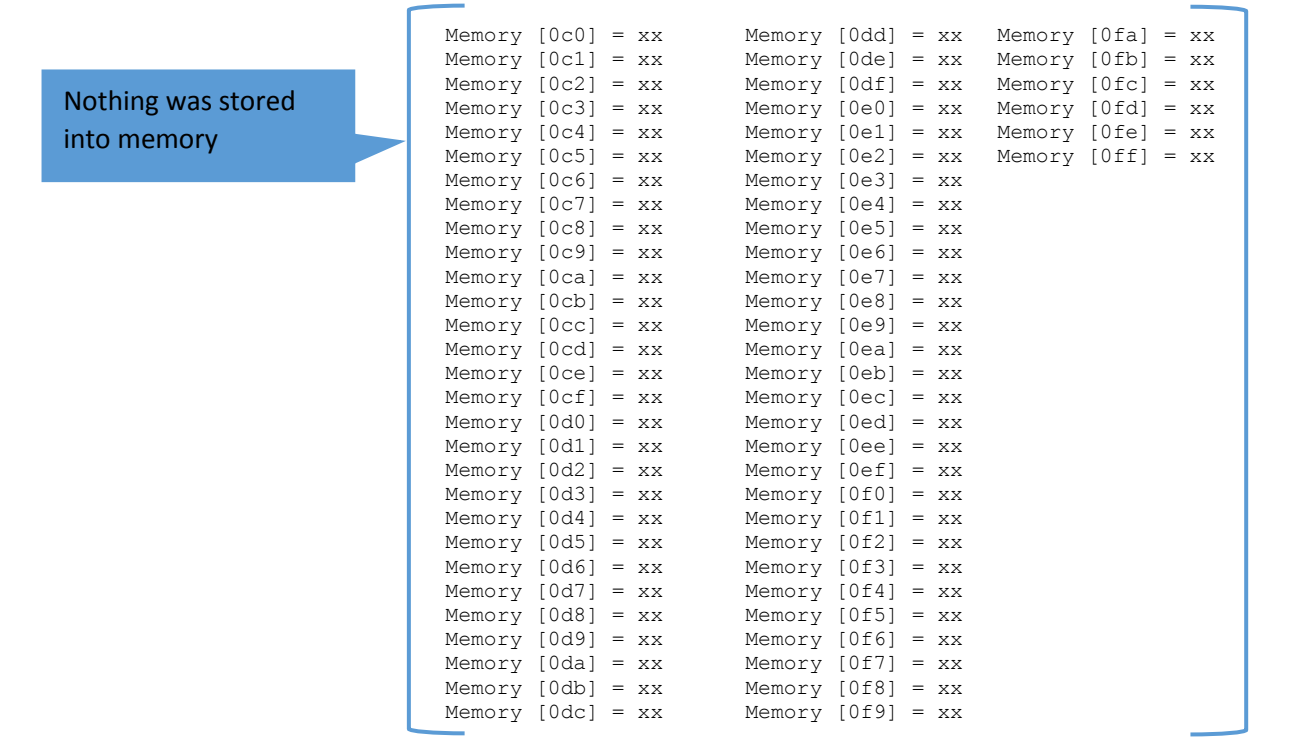
R15 is loaded after returning from jal. R31 saves last address to return from. R15 is loaded using lui, ori instruction.

Final values before bne is false and falls through to break point. Values are stored every 4 bytes, big Endian style.

Store data starting at @c0, big Endian. Increment 4 bytes after storing word.

III.D.8 Log File 8





III.D.9 Log File 9

```
=========================================Register Dump=============================================

Register [  0] = 00000000

Register [  1] = 00040911
Register [  2] = 000003e8
Register [  3] = fffbf6ef
Register [  4] = fffffc18
Register [  5] = 00000108
Register [  6] = 000001d1
Register [  7] = fffffef8
Register [  8] = fffffe2f

Register [  9] = 00000108
Register [ 10] = fffffe2f

Register [ 11] = 00000000
Register [ 12] = 00000000
Register [ 13] = 00000000
Register [ 14] = 00000000

Register [ 15] = 10010000
```

Loaded with values from Memory

R01 = Mem[R15+0] R02 =Mem[R15+4]

R03 = Mem[R15+8] R04 = Mem{R15+12]

R05 = Mem[R15+16] R06 = Mem[R15 +20]

R07 = Mem[R15+24] R08 = Mem[R15+28]

R09 ← Holds Quotient of division operations

R10 ← Holds Remainder of division operations

Success Flags for the follow divisions:

R01/R02 R03/R02 R01/R04 R03/R04

Base register used to address Memory:

Mem[R15+offset]

```
=========================================Memory Dump=============================================

Memory [0c0] = xx       Memory [0dd] = xx    Memory [0fa] = xx
Memory [0c1] = xx       Memory [0de] = xx    Memory [0fb] = xx
Memory [0c2] = xx       Memory [0df] = xx    Memory [0fc] = xx
Memory [0c3] = xx       Memory [0e0] = xx    Memory [0fd] = xx
Memory [0c4] = xx       Memory [0e1] = xx    Memory [0fe] = xx
Memory [0c5] = xx       Memory [0e2] = xx    Memory [0ff] = xx
Memory [0c6] = xx       Memory [0e3] = xx
Memory [0c7] = xx       Memory [0e4] = xx
Memory [0c8] = xx       Memory [0e5] = xx
Memory [0c9] = xx       Memory [0e6] = xx
Memory [0ca] = xx       Memory [0e7] = xx
Memory [0cb] = xx       Memory [0e8] = xx
Memory [0cc] = xx       Memory [0e9] = xx
Memory [0cd] = xx       Memory [0ea] = xx
Memory [0ce] = xx       Memory [0eb] = xx
Memory [0cf] = xx       Memory [0ec] = xx
Memory [0d0] = xx       Memory [0ed] = xx
Memory [0d1] = xx       Memory [0ee] = xx
Memory [0d2] = xx       Memory [0ef] = xx
Memory [0d3] = xx       Memory [0f0] = xx
Memory [0d4] = xx       Memory [0f1] = xx
Memory [0d5] = xx       Memory [0f2] = xx
Memory [0d6] = xx       Memory [0f3] = xx
Memory [0d7] = xx       Memory [0f4] = xx
Memory [0d8] = xx       Memory [0f5] = xx
Memory [0d9] = xx       Memory [0f6] = xx
Memory [0da] = xx       Memory [0f7] = xx
Memory [0db] = xx       Memory [0f8] = xx
Memory [0dc] = xx       Memory [0f9] = xx
```

Nothing was stored into memory

III.D.10 Log File 10

```
=======================================Register Dump=========================================

Register [  0] = 00000000

Register [  1] = ffffff8a
Register [  2] = 0000008a

Register [  3] = 00000000

Register [  4] = 000000c0
Register [  5] = 000000c4
Register [  6] = 000000c8
Register [  7] = 000000cc
Register [  8] = effeff35

Register [  9] = effeff35
Register [ 10] = ffffffff

Register [ 11] = ffffffff
Register [ 12] = 88778877
Register [ 13] = 77887788

Register [ 14] = 00000000

Register [ 15] = 100100c0
```

Loaded contents into registers using ADDI

R01 = 0 -118 , R02 = 0 +138

R3 ← Used to Test SLT and SLTU on the following pairs of registers:

R01 < R02. R02 < R01

Success Flag values to be stored into memory

R9 ← Used to Test AND,OR and NOR of the following registers :

R13 AND R12. R15 OR R02 R15 NOR R03

Success Flags of a working jump and link instruction

Success Flag of entire memory module

Base register used to address Memory:

Mem[R15+offset]

```
=====================================Memory Dump=========================================

Memory [0c0] = 00   Memory [0dd] = xx   Memory [0fa] = xx
Memory [0c1] = 00   Memory [0de] = xx   Memory [0fb] = xx
Memory [0c2] = 00   Memory [0df] = xx   Memory [0fc] = xx
Memory [0c3] = c0   Memory [0e0] = xx   Memory [0fd] = xx
Memory [0c4] = 00   Memory [0e1] = xx   Memory [0fe] = xx
Memory [0c5] = 00   Memory [0e2] = xx   Memory [0ff] = xx
Memory [0c6] = 00   Memory [0e3] = xx
Memory [0c7] = c4   Memory [0e4] = xx
Memory [0c8] = 00   Memory [0e5] = xx
Memory [0c9] = 00   Memory [0e6] = xx
Memory [0ca] = 00   Memory [0e7] = xx
Memory [0cb] = c8   Memory [0e8] = xx
Memory [0cc] = 00   Memory [0e9] = xx
Memory [0cd] = 00   Memory [0ea] = xx
Memory [0ce] = 00   Memory [0eb] = xx
Memory [0cf] = cc   Memory [0ec] = xx
Memory [0d0] = ef   Memory [0ed] = xx
Memory [0d1] = fe   Memory [0ee] = xx
Memory [0d2] = ff   Memory [0ef] = xx
Memory [0d3] = 35   Memory [0f0] = xx
Memory [0d4] = xx   Memory [0f1] = xx
Memory [0d5] = xx   Memory [0f2] = xx
Memory [0d6] = xx   Memory [0f3] = xx
Memory [0d7] = xx   Memory [0f4] = xx
Memory [0d8] = xx   Memory [0f5] = xx
Memory [0d9] = xx   Memory [0f6] = xx
Memory [0da] = xx   Memory [0f7] = xx
Memory [0db] = xx   Memory [0f8] = xx
Memory [0dc] = xx   Memory [0f9] = xx
```

Success value for SLT Instruction.

Mem[R15] = R04

Success value for SLTU Instruction

Mem[R15 + 4] = R05

Success value for SLT Instruction.

Mem[R15+8] = R06

Success value for SLTU Instruction

Mem[R15 + 12] = R07

Success value for NOR Instruction.

Mem[R15+16] = R08

III.D.11 Log File 11

```
=======================================Register Dump=========================================

Register [  0] = 00000000

Register [  1] = ffffff8a
Register [  2] = 0000008a

Register [  3] = 000000c0

Register [  4] = 000000c4
Register [  5] = 000000c8
Register [  6] = 000000cc
Register [  7] = 000000d0
Register [  8] = 000000d4

Register [  9] = xxxxxxxx
Register [ 10] = xxxxxxxx
Register [ 11] = xxxxxxxx
Register [ 12] = xxxxxxxx
Register [ 13] = xxxxxxxx

Register [ 14]  = 00000000

Register [ 15] = 100100c0
```

Loaded contents into registers using ADDI

R01 = 0 -118 , R02 = 0 +138

Success Flag value for functioning " Branch if less than or equal to zero" (BLEZ) instruction.

Success Flag values to be stored into memory

Success Flag of entire memory module

Base register used to address Memory:

Mem[R15+offset]

```
=======================================Memory Dump=========================================

Memory [0c0] = 00
Memory [0c1] = 00
Memory [0c2] = 00
Memory [0c3] = c0
Memory [0c4] = 00
Memory [0c5] = 00
Memory [0c6] = 00
Memory [0c7] = c4
Memory [0c8] = 00
Memory [0c9] = 00
Memory [0ca] = 00
Memory [0cb] = c8
Memory [0cc] = 00
Memory [0cd] = 00
Memory [0ce] = 00
Memory [0cf] = cc
Memory [0d0] = 00
Memory [0d1] = 00
Memory [0d2] = 00
Memory [0d3] = d0
Memory [0d4] = 00
Memory [0d5] = 00
Memory [0d6] = 00
Memory [0d7] = d4
Memory [0d8] = ff
Memory [0d9] = ff
Memory [0da] = ff
Memory [0db] = 8a

Memory [0dc] = 00
Memory [0dd] = 00
Memory [0de] = 00
Memory [0df] = 8a
Memory [0dd] = xx
Memory [0de] = xx
Memory [0df] = xx
Memory [0e0] = xx
Memory [0e1] = xx
Memory [0e2] = xx
Memory [0e3] = xx
Memory [0e4] = xx
Memory [0e5] = xx
Memory [0e6] = xx
Memory [0e7] = xx
Memory [0e8] = xx
Memory [0e9] = xx
Memory [0ea] = xx
Memory [0eb] = xx
Memory [0ec] = xx
Memory [0ed] = xx
Memory [0ee] = xx
Memory [0ef] = xx
Memory [0f0] = xx
Memory [0f1] = xx
Memory [0f2] = xx
Memory [0f3] = xx
Memory [0f4] = xx
emory [0f5] = xx
ory [0f6] = xx
y [0f7] = xx
[0f8] = xx
0f9] = xx

Memory [0fa] = xx
Memory [0fb] = xx
Memory [0fc] = xx
Memory [0fd] = xx
Memory [0fe] = xx
Memory [0ff] = xx
```

Success value for BLEZ and BGTZ Instruction.

Mem[R15] = R03

Success value for BLEZ Instruction

Mem[R15 + 4] = R04

Success value for BLEZ Instruction.

Mem[R15+8] = R05

Success value for BGTZ Instruction

Mem[R15 + 12] = R06

Success value for BGTZ Instruction.

Mem[R15+16] = R07

Success Flag of a working jump and link instruction

Mem[R15+24] = R07

Success Flag of a working jump and link instruction

Mem[R15+28] = R07

Success value for BGTZ Instruction.

Mem[R15+20] = R08

III.D.12 Log File 12

```
==========================================Register Dump=======================================
Register [  0] = 00000000

Register [  1] = ffffff8a
Register [  2] = 0000008a

Register [  3] = 00000000


Register [  4] = 000000c0
Register [  5] = 000000c4
Register [  6] = 000000d4

Register [  7] = 0000f0f0
Register [  8] = 00000000

Register [  9] = ffffffff

Register [ 10] = 0000f0f0
Register [ 11] = ffffffff
Register [ 12] = fffffaf5
Register [ 13] = ffff5555

Register [ 14] = 00000000


Register [ 15] = 100100c0
```

Loaded contents into registers using ADDI

R01 = 0 -118 , R02 = 0 +138

R3 ← results of SLTIU testing

Success Flag values to be stored into memory . Register R06 was stored into memory when it held a previous value of : 0x000000C8

R07 ← R12 ANDI 0xF5FA

R08 ← R07 - R10

R09 ← R13 XORI 0xAAAA

Success Flags of a working jump and link instruction

Success Flag of completed subroutine

Base register used to address Memory:

Mem[R15+offset]

```
==========================================Memory Dump=========================================
Memory [0c0] = 00
Memory [0c1] = 00
Memory [0c2] = 00
Memory [0c3] = c0
Memory [0c4] = 00
Memory [0c5] = 00
Memory [0c6] = 00
Memory [0c7] = c4
Memory [0c8] = 00
Memory [0c9] = 00
Memory [0ca] = 00
Memory [0cb] = c8
Memory [0cc] = 00
Memory [0cd] = 00
Memory [0ce] = 00
Memory [0cf] = cc
Memory [0d0] = 00
Memory [0d1] = 00
Memory [0d2] = 00
Memory [0d3] = d0
Memory [0d4] = 00
Memory [0d5] = 00
Memory [0d6] = 00
Memory [0d7] = d4
Memory [0d8] = ff
Memory [0d9] = ff
Memory [0da] = ff
Memory [0db] = 8a

Memory [0dc] = xx
Memory [0dd] = xx
Memory [0de] = xx
Memory [0df] = xx
Memory [0dd] = xx
Memory [0de] = xx
Memory [0df] = xx
Memory [0e0] = xx
Memory [0e1] = xx
Memory [0e2] = xx
Memory [0e3] = xx
Memory [0e4] = xx
Memory [0e5] = xx
Memory [0e6] = xx
Memory [0e7] = xx
Memory [0e8] = xx
Memory [0e9] = xx
Memory [0ea] = xx
Memory [0eb] = xx
Memory [0ec] = xx
Memory [0ed] = xx
Memory [0ee] = xx
Memory [0ef] = xx
Memory [0f8] = xx
Memory [0f9] = xx

Memory [0f0] = xx
Memory [0f1] = xx
Memory [0f2] = xx
Memory [0f3] = xx
Memory [0f4] = xx
Memory [0f5] = xx
Memory [0f6] = xx
Memory [0f7] = xx
Memory [0fa] = xx
Memory [0fb] = xx
Memory [0fc] = xx
Memory [0fd] = xx
Memory [0fe] = xx
Memory [0ff] = xx
```

Success value for SLTIU instruction

Mem[R15] = R04

Success value for SLTIU instruction

Mem[R15 + 4] = R05

Success value for SLTIU Instruction.

Mem[R15+8] = R06

Success value for SLTIU Instruction

Mem[R15 + 12] = R07

Success value for SLTIU Instruction.

Mem[R15+16] = R08

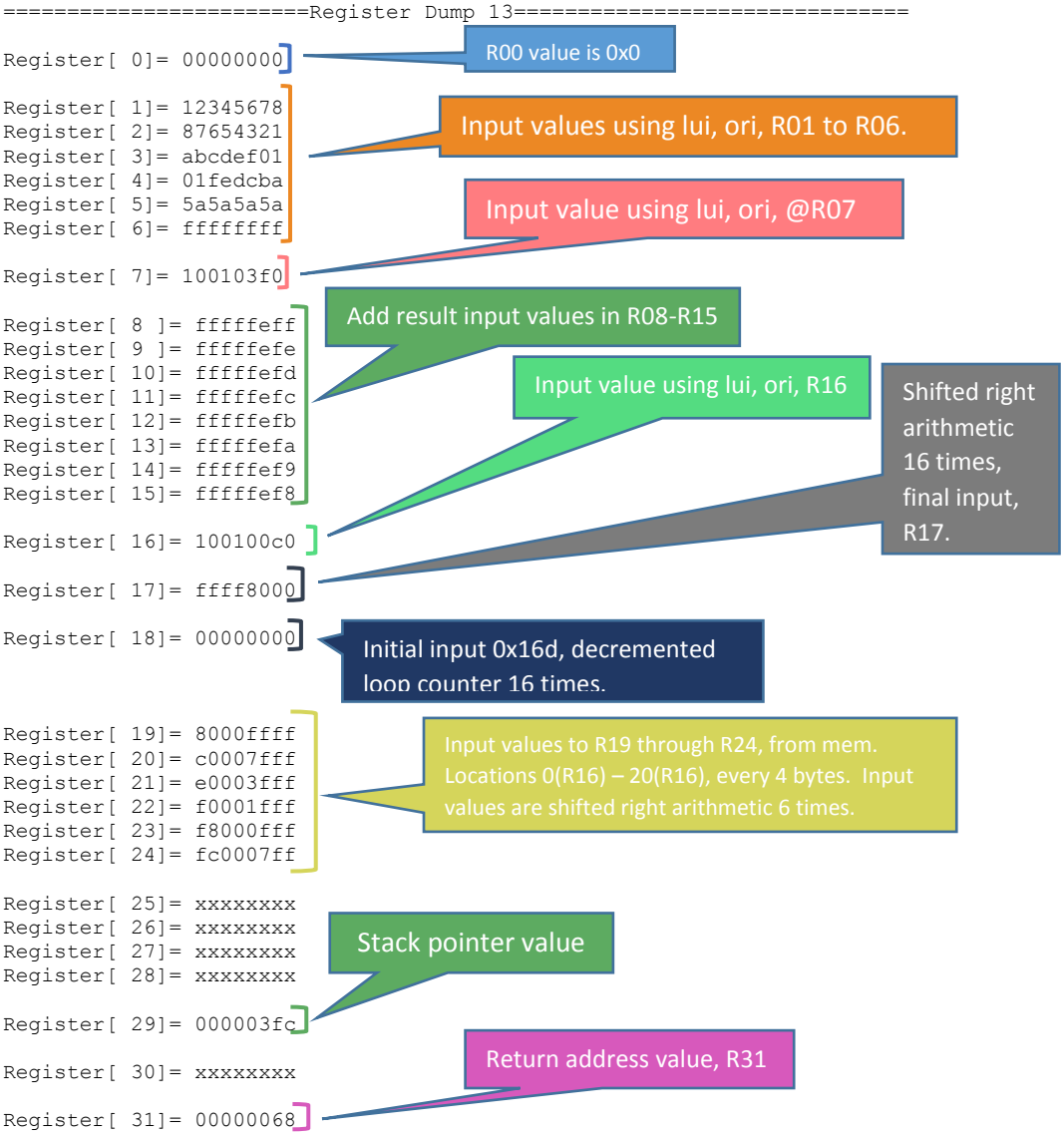
Success Flag of a completely working Memory module

Success value for SLTIU Instruction.

Mem[R15+20] = R06

III.D.13 Log File 13

```
=======================Register Dump 13================================

Register[ 0]= 00000000

Register[ 1]= 12345678
Register[ 2]= 87654321
Register[ 3]= abcdef01
Register[ 4]= 01fedcba
Register[ 5]= 5a5a5a5a
Register[ 6]= ffffffff

Register[ 7]= 100103f0

Register[ 8 ]= ffffffef
Register[ 9 ]= ffffffe
Register[ 10]= ffffffd
Register[ 11]= ffffffc
Register[ 12]= ffffffb
Register[ 13]= ffffffa
Register[ 14]= ffffff9
Register[ 15]= ffffff8

Register[ 16]= 100100c0

Register[ 17]= ffff8000

Register[ 18]= 00000000


Register[ 19]= 8000ffff
Register[ 20]= c0007fff
Register[ 21]= e0003fff
Register[ 22]= f0001fff
Register[ 23]= f8000fff
Register[ 24]= fc0007ff

Register[ 25]= xxxxxxxx
Register[ 26]= xxxxxxxx
Register[ 27]= xxxxxxxx
Register[ 28]= xxxxxxxx

Register[ 29]= 000003fc

Register[ 30]= xxxxxxxx

Register[ 31]= 00000068
```



```
==========================I/O_Mem_Dump_13==============================
I_O_Memory [0c0] = 80       I_O_Memory [0d8] = fe       I_O_Memory [0f0] = ff
I_O_Memory [0c1] = 00       I_O_Memory [0d9] = 00       I_O_Memory [0f1] = f8
I O Memory [0c2] = ff       I O Memory [0da] = 03       I O Memory [0f2] = 00
I_O_Memory [0c3] = ff       I_O_Memory [0db] = ff       I_O_Memory [0f3] = 0f

I_O_Memory [0c4] = c0       I_O_Memory [0dc] = ff       I_O_Memory [0f4] = ff
I_O_Memory [0c5] = 00       I_O_Memory [0dd] = 00       I_O_Memory [0f5] = fc
I_O_Memory [0c6] = 7f       I_O_Memory [0de] = 01       I_O_Memory [0f6] = 00
I O Memory [0c7] = ff       I O Memory [0df] = ff       I O Memory [0f7] = 07

I O Memory [0c8] = e0       I O Memory [0e0] = ff       I O Memory [0f8] = ff
I_O_Memory [0c9] = 00       I_O_Memory [0e1] = 80       I_O_Memory [0f9] = fe
I_O_Memory [0ca] = 3f       I_O_Memory [0e2] = 00       I_O_Memory [0fa] = 00
I_O_Memory [0cb] = ff       I_O_Memory [0e3] = ff       I_O_Memory [0fb] = 03

I O Memory [0cc] = f0       I O Memory [0e4] = ff       I O Memory [0fc] = ff
I_O_Memory [0cd] = 00       I_O_Memory [0e5] = c0       I_O_Memory [0fd] = ff
I_O_Memory [0ce] = 1f       I_O_Memory [0e6] = 00       I_O_Memory [0fe] = 00
I_O_Memory [0cf] = ff       I_O_Memory [0e7] = 7f       I_O_Memory [0ff] = 01

I_O_Memory [0d0] = f8       I_O_Memory [0e8] = ff
I O Memory [0d1] = 00       I O Memory [0e9] = e0
I_O_Memory [0d2] = 0f       I_O_Memory [0ea] = 00
I_O_Memory [0d3] = ff       I_O_Memory [0eb] = 3f

I_O_Memory [0d4] = fc       I_O_Memory [0ec] = ff
I_O_Memory [0d5] = 00       I_O_Memory [0ed] = f0
I O Memory [0d6] = 07       I O Memory [0ee] = 00
I_O_Memory [0d7] = ff       I_O_Memory [0ef] = 1f
```

Input values I/O memory every 4 bytes. Shift right arithmetic 16 times. Start @ddress 0xC0h.

Input I/O memory dump is Big Endian style. Finish @ddress 0xFCh.

```
==========================Mem_Dump_13==============================
Memory [000003c0] = xx      Memory [000003d8] = xx      Memory [000003f0] = ff
Memory [000003c1] = xx      Memory [000003d9] = xx      Memory [000003f1] = ff
Memory [000003c2] = xx      Memory [000003da] = xx      Memory [000003f2] = fe
Memory [000003c3] = xx      Memory [000003db] = xx      Memory [000003f3] = f8

Memory [000003c4] = xx      Memory [000003dc] = xx
Memory [000003c5] = xx      Memory [000003dd] = xx
Memory [000003c6] = xx      Memory [000003de] = xx
Memory [000003c7] = xx      Memory [000003df] = xx

Memory [000003c8] = xx      Memory [000003e0] = xx
Memory [000003c9] = xx      Memory [000003e1] = xx
Memory [000003ca] = xx      Memory [000003e2] = xx
Memory [000003cb] = xx      Memory [000003e3] = xx

Memory [000003cc] = xx      Memory [000003e4] = xx
Memory [000003cd] = xx      Memory [000003e5] = xx
Memory [000003ce] = xx      Memory [000003e6] = xx
Memory [000003cf] = xx      Memory [000003e7] = xx

Memory [000003d0] = xx      Memory [000003e8] = xx
Memory [000003d1] = xx      Memory [000003e9] = xx
Memory [000003d2] = xx      Memory [000003ea] = xx
Memory [000003d3] = xx      Memory [000003eb] = xx

Memory [000003d4] = xx      Memory [000003ec] = xx
Memory [000003d5] = xx      Memory [000003ed] = xx
Memory [000003d6] = xx      Memory [000003ee] = xx
Memory [000003d7] = xx      Memory [000003ef] = xx
```

After returning from Rra address, input R15 value into 0(R7).

III.D.14 Log File 14

```
==========================Register_Dump_14================================

Register [  0] = 00000000

Register [  1] = 12345678
Register [  2] = 87654321
Register [  3] = abcdef01
Register [  4] = 01fedcba
Register [  5] = 5a5a5a5a
Register [  6] = ffffffff

Register [  7] = 100103f0

Register [  8] = fffffeff
Register [  9] = fffffefe
Register [ 10] = fffffefd
Register [ 11] = fffffefc
Register [ 12] = fffffefb
Register [ 13] = fffffefa
Register [ 14] = fffffef9
Register [ 15] = fffffef8

Register [ 16] = 100100c0

Register [ 17] = ffff8000

Register [ 18] = 00000000

Register [ 19] = 8000ffff
Register [ 20] = c0007fff
Register [ 21] = e0003fff
Register [ 22] = f0001fff
Register [ 23] = f8000fff
Register [ 24] = fc0007ff

Register [ 25] = xxxxxxxx
Register [ 26] = xxxxxxxx
Register [ 27] = xxxxxxxx
Register [ 28] = xxxxxxxx

Register [ 29] = 000003fc

Register [ 30] = xxxxxxxx

Register [ 31] = xxxxxxxx
```

R00 value is 0x0

Input values using lui, ori, R01 to R06.

Input value using lui, ori, @R07

Add result input values in R08-R15

Input value using lui, ori, R16

Shifted right arithmetic 16 times, final input, R17.

Initial input 0x16d, decremented loop counter 16 times.

Input values to R19 through R24, from mem. Locations 0(R16) – 20(R16), every 4 bytes. Input values are shifted right arithmetic 6 times.

Stack pointer value

Return address saved on stack, not R31.

```
==========================Memory Dump 14==============================

Memory [000003c0] = xx          Memory [000003d4] = xx          Memory [000003e8] = xx
Memory [000003c1] = xx          Memory [000003d5] = xx          Memory [000003e9] = xx
Memory [000003c2] = xx          Memory [000003d6] = xx          Memory [000003ea] = xx
Memory [000003c3] = xx          Memory [000003d7] = xx          Memory [000003eb] = xx

Memory [000003c4] = xx          Memory [000003d8] = xx          Memory [000003ec] = xx
Memory [000003c5] = xx          Memory [000003d9] = xx          Memory [000003ed] = xx
Memory [000003c6] = xx          Memory [000003da] = xx          Memory [000003ee] = xx
Memory [000003c7] = xx          Memory [000003db] = xx          Memory [000003ef] = xx

Memory [000003c8] = xx          Memory [000003dc] = xx          Memory [000003f0] = ff
Memory [000003c9] = xx          Memory [000003dd] = xx          Memory [000003f1] = ff
Memory [000003ca] = xx          Memory [000003de] = xx          Memory [000003f2] = fe
Memory [000003cb] = xx          Memory [000003df] = xx          Memory [000003f3] = f8

Memory [000003cc] = xx          Memory [000003e0] = xx
Memory [000003cd] = xx          Memory [000003e1] = xx
Memory [000003ce] = xx          Memory [000003e2] = xx
Memory [000003cf] = xx          Memory [000003e3] = xx

Memory [000003d0] = xx          Memory [000003e4] = xx
Memory [000003d1] = xx          Memory [000003e5] = xx
Memory [000003d2] = xx          Memory [000003e6] = xx
Memory [000003d3] = xx          Memory [000003e7] = xx
```

After returning from Rra address, input R15 value into 0(R7).

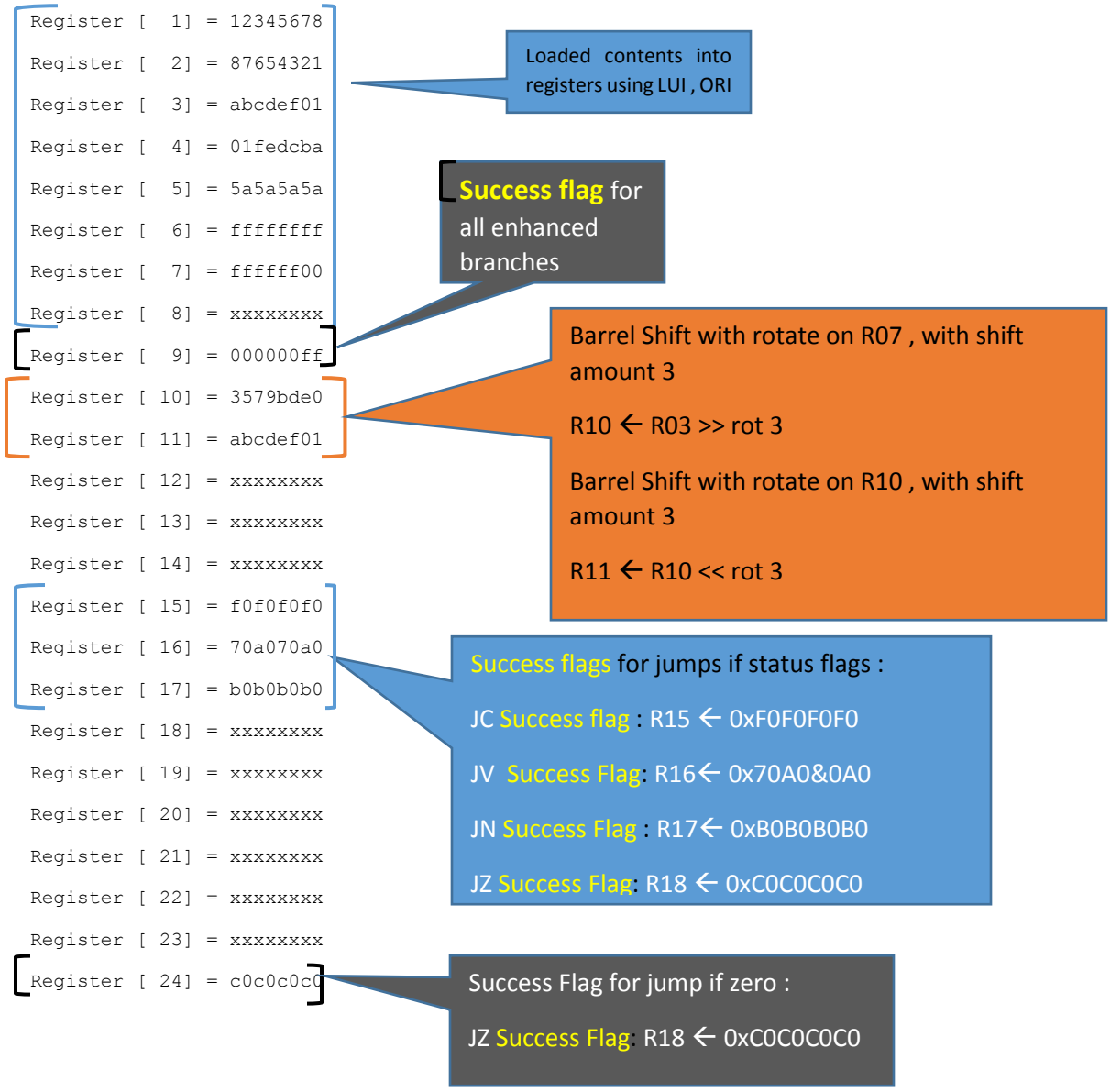
```
==========================I/O_Memory_Dump_14==============================

I_O_Memory [000000c0] = 80      I_O_Memory [000000d8] = fe      I_O_Memory [000000f0] = ff
I O Memory [000000c1] = 00      I O Memory [000000d9] = 00      I O Memory [000000f1] = f8
I_O_Memory [000000c2] = ff      I_O_Memory [000000da] = 03      I_O_Memory [000000f2] = 00
I_O_Memory [000000c3] = ff      I_O_Memory [000000db] = ff      I_O_Memory [000000f3] = 0f

I_O_Memory [000000c4] = c0      I_O_Memory [000000dc] = ff      I_O_Memory [000000f4] = ff
I_O_Memory [000000c5] = 00      I_O_Memory [000000dd] = 00      I_O_Memory [000000f5] = fc
I O Memory [000000c6] = 7f      I O Memory [000000de] = 01      I O Memory [000000f6] = 00
I_O_Memory [000000c7] = ff      I_O_Memory [000000df] = ff      I_O_Memory [000000f7] = 07

I_O_Memory [000000c8] = e0      I_O_Memory [000000e0] = ff      I_O_Memory [000000f8] = ff
I_O_Memory [000000c9] = 00      I_O_Memory [000000e1] = 80      I_O_Memory [000000f9] = fe
I_O_Memory [000000ca] = 3f      I_O_Memory [000000e2] = 00      I_O_Memory [000000fa] = 00
I O Memory [000000cb] = ff      I O Memory [000000e3] = ff      I O Memory [000000fb] = 03

I_O_Memory [000000cc] = f0      I_O_Memory [000000e4] = ff      I_O_Memory [000000fc] = ff
I_O_Memory [000000cd] = 00      I_O_Memory [000000e5] = c0      I_O_Memory [000000fd] = ff
I_O_Memory [000000ce] = 1f      I_O_Memory [000000e6] = 00      I_O_Memory [000000fe] = 00
I O Memory [000000cf] = ff      I O Memory [000000e7] = 7f      I_O_Memory [000000ff] = 01

I O Memory [000000d0] = f8      I O Memory [000000e8] = ff
I_O_Memory [000000d1] = 00      I_O_Memory [000000e9] = e0
I_O_Memory [000000d2] = 0f      I_O_Memory [000000ea] = 00
I_O_Memory [000000d3] = ff      I_O_Memory [000000eb] = 3f

I O Memory [000000d4] = fc      I O Memory [000000ec] = ff
I_O_Memory [000000d5] = 00      I_O_Memory [000000ed] = f0
I_O_Memory [000000d6] = 07      I_O_Memory [000000ee] = 00
I_O_Memory [000000d7] = ff      I_O_Memory [000000ef] = 1f
```

Input values I/O memory every 4 bytes. Shift right arithmetic 16 times. Start @ddress 0xC0h.

Input I/O memory dump is Big Endian style. Finish @ddress 0xFCh.

III.D.15 Enhanced Module

```
=========================================Register Dump=====================================
Register [  0] = 00000000
Register [  1] = 12345678
Register [  2] = 87654321
Register [  3] = abcdef01
Register [  4] = 01fedcba
Register [  5] = 5a5a5a5a
Register [  6] = ffffffff
Register [  7] = ffffff00
Register [  8] = xxxxxxxx
Register [  9] = 000000ff
Register [ 10] = 3579bde0
Register [ 11] = abcdef01
Register [ 12] = xxxxxxxx
Register [ 13] = xxxxxxxx
Register [ 14] = xxxxxxxx
Register [ 15] = f0f0f0f0
Register [ 16] = 70a070a0
Register [ 17] = b0b0b0b0
Register [ 18] = xxxxxxxx
Register [ 19] = xxxxxxxx
Register [ 20] = xxxxxxxx
Register [ 21] = xxxxxxxx
Register [ 22] = xxxxxxxx
Register [ 23] = xxxxxxxx
Register [ 24] = c0c0c0c0
```



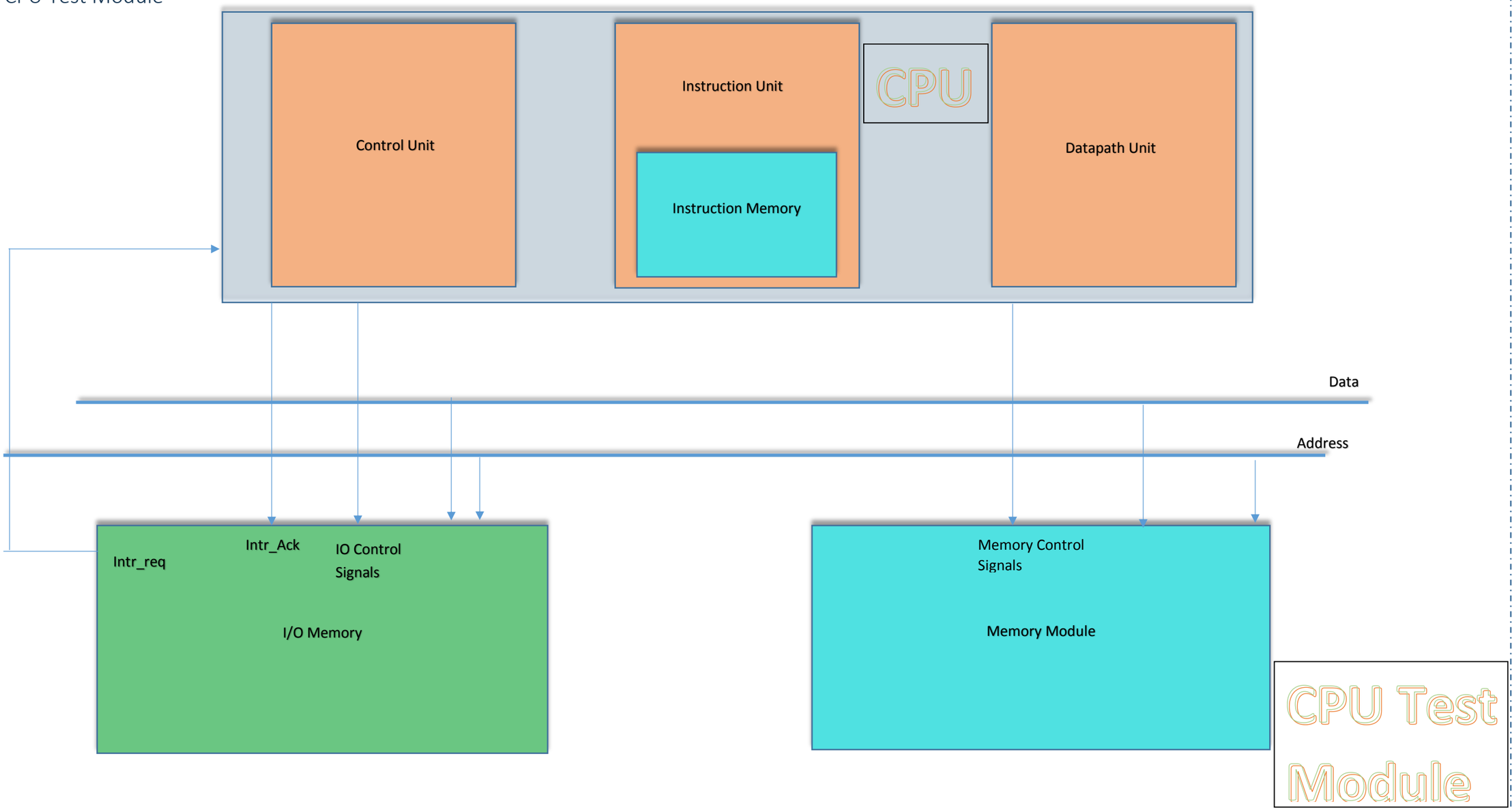
```
=======================================Memory Dump=============================================
```

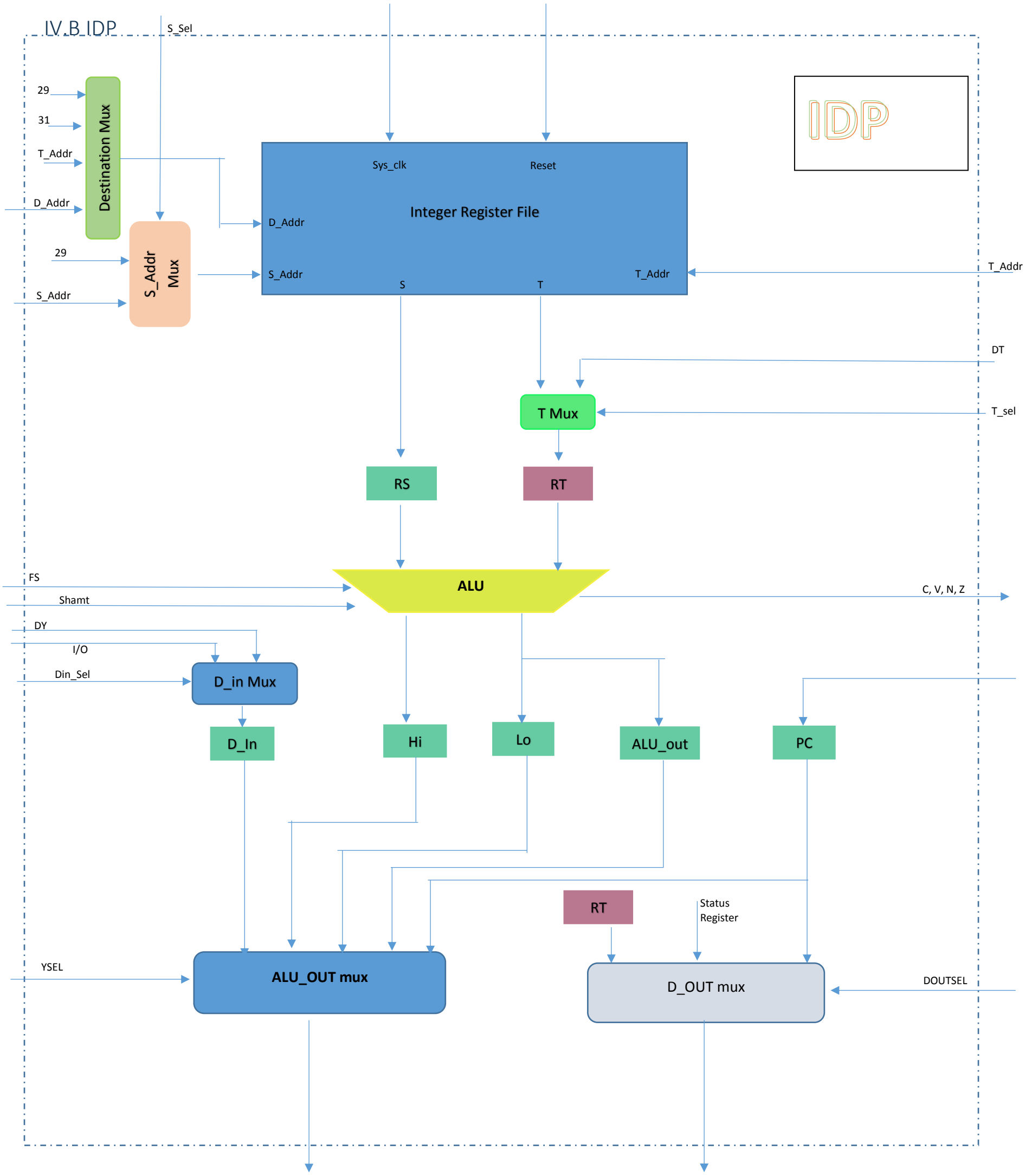
Nothing was stored into memory

```
Memory [0c0] = xx      Memory [0dd] = xx    Memory [0fa] = xx
Memory [0c1] = xx      Memory [0de] = xx    Memory [0fb] = xx
Memory [0c2] = xx      Memory [0df] = xx    Memory [0fc] = xx
Memory [0c3] = xx      Memory [0e0] = xx    Memory [0fd] = xx
Memory [0c4] = xx      Memory [0e1] = xx    Memory [0fe] = xx
Memory [0c5] = xx      Memory [0e2] = xx    Memory [0ff] = xx
Memory [0c6] = xx      Memory [0e3] = xx
Memory [0c7] = xx      Memory [0e4] = xx
Memory [0c8] = xx      Memory [0e5] = xx
Memory [0c9] = xx      Memory [0e6] = xx
Memory [0ca] = xx      Memory [0e7] = xx
Memory [0cb] = xx      Memory [0e8] = xx
Memory [0cc] = xx      Memory [0e9] = xx
Memory [0cd] = xx      Memory [0ea] = xx
Memory [0ce] = xx      Memory [0eb] = xx
Memory [0cf] = xx      Memory [0ec] = xx
Memory [0d0] = xx      Memory [0ed] = xx
Memory [0d1] = xx      Memory [0ee] = xx
Memory [0d2] = xx      Memory [0ef] = xx
Memory [0d3] = xx      Memory [0f0] = xx
Memory [0d4] = xx      Memory [0f1] = xx
Memory [0d5] = xx      Memory [0f2] = xx
Memory [0d6] = xx      Memory [0f3] = xx
Memory [0d7] = xx      Memory [0f4] = xx
Memory [0d8] = xx      Memory [0f5] = xx
Memory [0d9] = xx      Memory [0f6] = xx
Memory [0da] = xx      Memory [0f7] = xx
Memory [0db] = xx      Memory [0f8] = xx
Memory [0dc] = xx      Memory [0f9] = xx
```

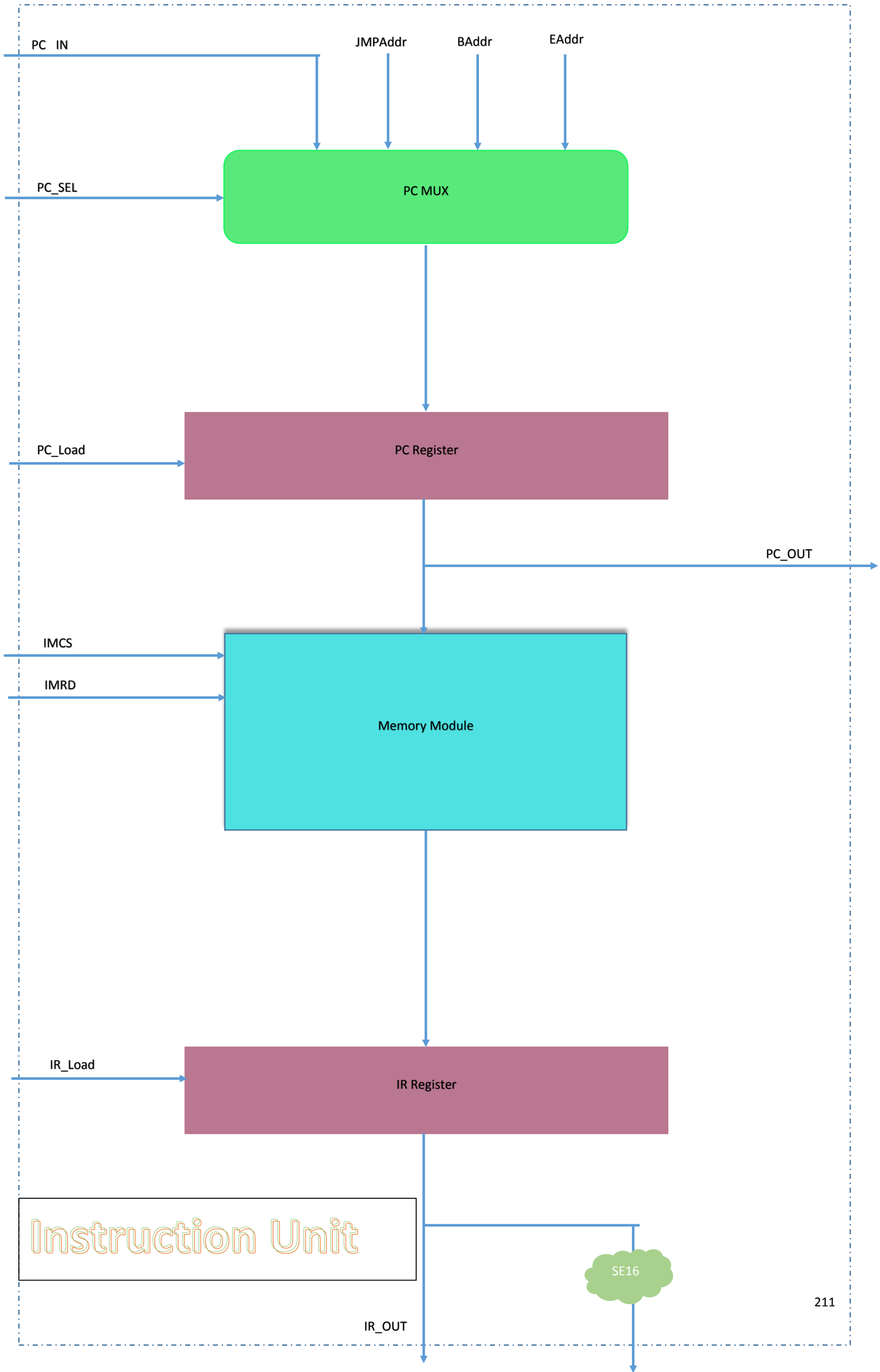
IV Hardware Implementation .

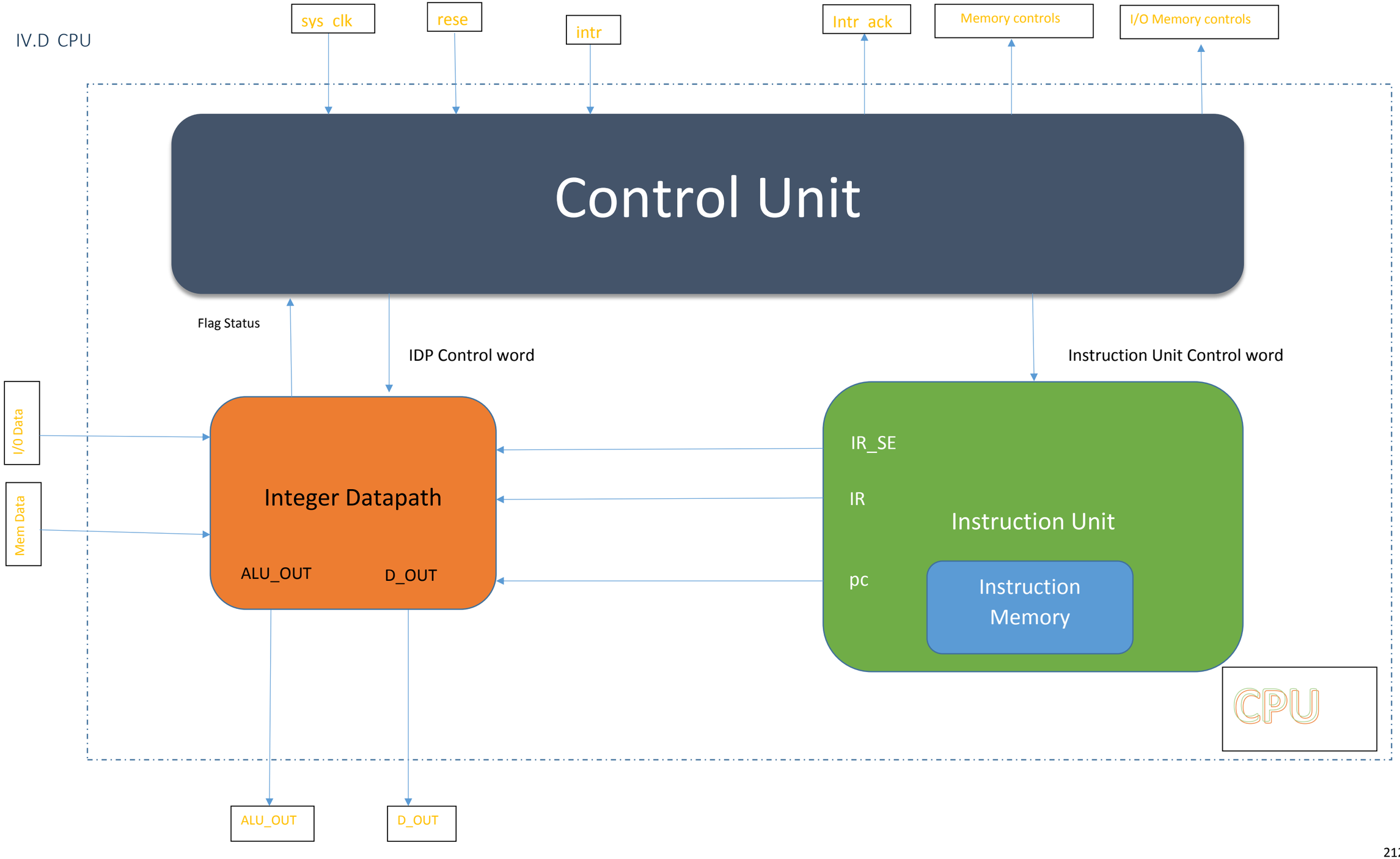
IV.A CPU Test Module




IV.B IDP
S_Sel
29
31
T_Addr
D_Addr
Destination Mux
29
S_Addr
S_Addr Mux
Sys_clk
Reset
Integer Register File
D_Addr
S_Addr
S
T
T_Addr
IDP
T_Addr
DT
T_sel
T Mux
RS
RT
FS
Shamt
ALU
C, V, N, Z
DY
I/O
Din_Sel
D_in Mux
D_In
Hi
Lo
ALU_out
PC
RT
Status Register
YSEL
ALU_OUT mux
D_OUT mux
DOUTSEL

IV.C Instruction Unit




sys clk
rese
intr
Intr ack
Memory controls
I/O Memory controls
Control Unit
Flag Status
IDP Control word
Instruction Unit Control word
I/0 Data
Mem Data
Integer Datapath
ALU_OUT
D_OUT
IR_SE
IR
pc
Instruction Unit
Instruction Memory
CPU
ALU_OUT
D_OUT