4/24/2017

ET51 Processor Instruction Set Architecture

CECS 440: Computer Architecture

MW: 11:00 AM – 1:00 PM

Instructor: R. W. Allison

Tuan Nguyen, Eduardo Marquez
CAL STATE UNIVERSITY: LONG BEACH

Abstract

We come with an idea that combines our names to name our processor. ET51 processor is a MIPS based processor which is a load-store architecture which only performs arithmetic and logic operations between CPU registers and requires load and store instruction to access the memory. The ET51 processor also has 32 general purpose registers, but some of them are reserved for the system. We can use the rest registers and temporary registers for the Arithmetic/Logic Unit (ALU) operations. In this Instruction Set Architecture (ISA), my team will design a customized MIPS processor for around 50 operations. By using this ISA, users will have better ideas to understand processor register set, data types, addressing modes, instruction set, instruction formats, and help them to visualize how the system was built.

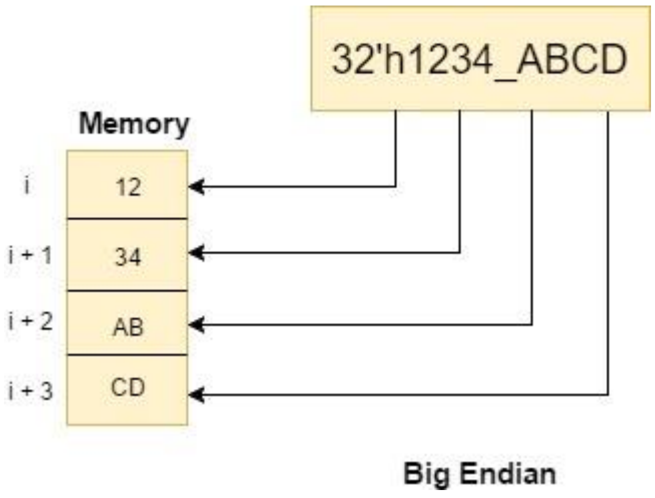
Table of contents

Contents **Page**

Instruction Set Architecture

Harvard Memory Architecture and Organization

A Harvard Memory Architecture is used; the Instruction Memory is separate from the Data and IO Memory. Both memory locations are byte addressable, each memory address will hold one byte. The Instruction Memory is 8-bits wide and 4096 addresses deep and is organized to function as a 32-bit wide and 1024 address deep memory. The other memory location contains both the Data and IO Memories in one module. Both of which are structured the same way as the Instruction Memory. Each memory locations holds an 8-bit word so 4 consecutive memory locations will be used to create a 32-bit memory operand. Both memory locations are in big endian format, the most significant 8-bits in the 32-bit operand are stored in the lower memory address and the least significant 8-bits are stored in the higher memory address.



Register File

The Register File is an array of registers and is 32 bits wide and 32 addresses deep. The Register File reads the operands from the IR register during the Decode state and provides the R address, T address, D address, 16-bit immediate values, the shift amount, and the opcode that will be executed in the next state. The addressing type will determine which of these operand sections will be read. The zero register in this module will always contain the value of zero. Register 29 is reserved for the stack pointer and register 31 is reserved for the return address.

Name	Register Number	Usage	Preserve on call?
$zero	0	The constant 0	N/A
$at	1	Temporary for assembler	No
$v0 - $v1	2-3	Returned values	No
$a0 - $a3	4-7	Arguments	Yes
$t0 - $t7	8-15	Temporaries	No
$s0 - $s7	16-23	Saved values	Yes
$t8 - $t9	24-25	Temporaries	No
$gp	28	Global pointer	Yes
$sp	29	Stack pointer	Yes
$fp	30	Frame pointer	Yes
$ra	31	Return address	Yes

Machine Register Set

All of the registers used in this design are 32-bits wide.

Data Register

Data registers are 32-bit registers used to hold data which will be needed by other modules. They are mainly used as an intermediate point to hold the results of any operation and are strategically placed between the Register File, ALU, and Memory modules. Some of these registers contain a load function where a select bit or bits will need to be asserted to load into the register.

PC register

The PC register is a 32-bit register that holds the Program Counter. The Program Counter will provide the memory address of the next instruction to be fetched. This register has a load function and an increment function. The increment function adds 4 to the PC, it adds 4 since it will need to pass 4 memory addresses to get to the next 32-bit instruction memory operand.

IR register

The IR register is a 32-bit register that holds the instruction fetched by the Instruction Memory at the address provided by the Program Counter. It also has a load enable.

Flags Registers

These registers hold the values of the flags IE, C, V, N, and Z.

IE - interrupt enable, enables an interrupt to occur if both this flag and the intr bit are set high.

C - carry, is set if a carry bit is set during an ALU operation. That is if the most significant bit from the resulting value of the ALU operation either borrows or carries a bit.

V – overflow, this flag is set when the signed result of an operation doesn’t fit in the number of bits provided without changing the values sign resulting in a wrong sign. For example, if either two positive integers are added and result in a negative or two negative integers are added and result in a positive.

N – negative, this flag shows that the result of an operation was negative.

Z – zero, this flag is indicates that the result of an operation was zero.

Data Types

32-bit signed integer

32-bit unsigned integer

Decimal

Integer

Addressing Modes

Register addressing:

The address is assigned by an operand as a register address.

Ex. rs(0x09) -> $r9

Immediate addressing:

A numeric value resulting from the instruction is the operand.

Ex. $rt <- $rs() + imm16()

Base addressing:

The effective address is calculated by adding a base value provided by a register and a 16-bit immediate offset value.

Ex. 4($rs) -> 4($rs(0x00000008) -> 0x00000008 + 0x00000004 = 0x0000000C

PC—relative addressing:

The PC value of the next instruction is added to a 16-bit immediate value that is left shifted by 2 and sign extended. The immediate value will specify how many instructions will be “jumped”, the shift will multiply the immediate value by 4.

Ex. PC=0x4, imm16=0x1: (PC+4) + {sign-extended(imm16)<<2} -> (0x00000004 + 4) + (0x00000004) = 0x0000000B

Pseudo—direct addressing:

The address is calculated by concatenating the 4 most significant bits of the PC with 26-bits provided by the instruction which will be shifted left 2 times.

Ex. {PC[31:28],26-bit operand,2’b00}

Register Indirect:

The value of the effective address is located in a register.

Ex. rs(0x1F) -> $31(0x000003FC) -> 0x000003FC

Instruction Format

R-Type:

31 26	25 21	20 16	15 11	10 6	5 0
0x00	rs	rt	rd	shtamt	function

I-Type:

31 26	25 21	20 16	15 0
opcode	rs	rt	16-bit immediate

J-Type:

31 26	25 0
opcode	26-bit jump address

R – Type Instructions:

SLL Shift Word Left Logical

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

0x00	0x00	rt	rd	shamt	0x00
6	5	5	5	5	6

Format:

SLL rd, rt, shamt

Purpose:

To shift a word to the left by a certain number of bits.

Description:

The value that register $rt holds is shifted to the left as many times as shamt specifies. Zeros are inserted into the least significant bit as shifting occurs to replace empty bits. The result is placed in register $rd.

Restrictions:

None

Operation:

Fetch: PC <- PC +4; IR <- M[PC];

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

SLL: ALU_OUT <- RT << shamt;

WB_alu: Reg[IR[15-11]] <- ALU_OUT(RT << shamt);

Exceptions:

None

Programming Notes:

None

Example:

```
3C 01 00 00     // lui $01, 0x0000
34 21 FF FF     // ori $01, 0xFFFF       # LI   R01,   0x0000FFFF
00 01 0C 00     // sll  $r1, $r1, 16     R01 = 0xFFFF0000
```

SRL Shift Word Right Logical

31 26	25 21	20 16	15 11	10 6	5 0
0x00	0x00	rt	rd	shamt	0x02
6	5	5	5	5	6

Format:

SRL rd, rt, shamt

Purpose:

To logical right shift a word by a fixed number of bits.

Description:

The value that register $rt holds is shifted to the right as many times as shamt specifies. Zeros are inserted into the most significant bit as shifting occurs to replace empty bits. The result is placed in register $rd.

Restrictions:

None

Operation:

Fetch: PC <- PC +4; IR <- M[PC];

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

SRL: ALU_OUT <- RT >> shamt;

WB_alu: Reg[IR[15-11]] <- ALU_OUT(RT >> shamt);

Exceptions:

None

Programming Notes:

None

Example:

```
3C 01 FF FF     // lui $01, 0xFFFF
34 21 00 00     // ori $01, 0x0000      # LI   R01,   0xFFFF0000
00 01 0C 02     // srl  $r1, $r1, 16    R1 = 0x0000FFFF
```

SRA Shift Right Word Arithmetic

31 26	25 21	20 16	15 11	10 6	5 0
0x00	0x00	rt	rd	shamt	0x03
6	5	5	5	5	6

Format:

SRA rd, rt, shamt

Purpose:

To arithmetic right shift a word by a fixed number of bits.

Description:

The value that register $rt holds is shifted to the right as many times as shamt specifies. Bits with the value of the sign bit (bit 31) are inserted into the most significant bit as shifting occurs to replace empty bits. The result is placed in register $rd.

Restrictions:

None

Operation:

Fetch: PC <- PC +4; IR <- M[PC];

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

SRL: ALU_OUT <- RT >> shamt;

WB_alu: Reg[IR[15-11]] <- ALU_OUT(RT >> shamt);

Exceptions:

None

Programming Notes:

None

Example:

```
3C 01 F6 FA     // lui $01, 0xF6FA
34 21 24 CE     // ori $01, 0x24CE        # LI   R01,   0xF6FA24CE
                // R1 = 1111 0110 1111 1010 0010 0100 1100 1110
00 01 10 83     // sra  $r2, $r1, 2       # R2 = 0xFDBE8933
                // R2 = 1111 1101 1011 1110 1000 1001 0011 0011
```

JR Jump Register

31 26	25 21	20 6	5 0
0x00	rs	26 bit Jump Address	0x08
6	5	26	6

Format:

JR rs

Purpose:

To jump to an address stored in a register.

Description:

Jumps to the effective address as specified in register $rs.

Execute the instruction following the jump, in the branch delay slot, before jumping.

Restrictions:

If either of the two least-significant bits are not zero, then an address error exception occurs which will violate the branch address when the branch target is subsequently fetched as an instruction.

Operation:

Fetch: PC <- PC +4; IR <- M[PC];

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

JR: ALU_OUT <- RS;

JR2: PC <- ALU_OUT($rs);

Exceptions:

None

Programming Notes:

None

Example:

```
3C 01 12 34    // lui $01, 0x1234
34 21 AB CD    // ori $01, 0xABCD      # LI   R01,   0x1234ABCD
00 20 00 08    // jr $r1               # Jump to address 0x1234_ABCD
```

MFHI Move From Hi Register

31 26	25 21	20 16	15 11	10 6	5 0
0x00	0x00	0x00	rd	0x00	0x10
6	5	5	5	5	6

Format:

MFHI rd

Purpose:

To copy the special purpose HI register to a general purpose register $rd.

Description:

The contents of special register HI are loaded into register $rd.

Restrictions:

MFHI or MFLO will not allow you to do a multiply or a divide instruction within two instructions after it executed. The reason is it violates how the MIPS pipeline works.

Operation:

*We did a multiply or a divide instruction ahead.

MULT: ALU_OUT <- RS * RT or DIV: ALU_OUT <- RS / RT

*Now we load Hi's content to register $rd

Fetch: PC <- PC +4; IR <- M[PC];

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

MFHI: Reg[IR[15-11]] <- ALU_OUT[31-16]

Exceptions: None

Programming Notes: None

Example:

```
3C 01 00 00     // lui $01, 0x0000
34 21 04 D2     // ori $01, 0x04D2      # LI   R01,   0x000004D2 (Decimal: 1234)
3C 01 00 00     // lui $02, 0x0000
34 21 16 2E     // ori $02, 0x162E      # LI   R02,   0x0000162E (Decimal: 5678)
00 21 00 18     // mult $r1, $r2        # R1*R2 = 0x006AE9BC (Decimal: 7,006,652)
00 00 18 10     // mfhi $r3             # R3 = 0x006A
```

MFLO Move From Lo Register

31 26	25 21	20 16	15 11	10 6	5 0
0x00	0x00	0x00	rd	0x00	0x12
6	5	5	5	5	6

Format:

MFLO rd

Purpose:

To copy the special purpose LO register to general purpose register $rd.

Description:

The contents of special register LO are loaded into register $rd.

Restrictions:

MFHI or MFLO will not allow you to do a multiply or a divide instruction within two instructions after it executed. The reason is it violates how the MIPS pipeline works.

Operation:

*We did a multiply or a divide instruction ahead.

MULT: ALU_OUT <- RS * RT or DIV: ALU_OUT <- RS / RT

*Now we load Hi’s content to register $rd

Fetch: PC <- PC +4; IR <- M[PC];

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

MFLO: Reg[IR[15-11]] <- ALU_OUT[15-0]

Exceptions: None

Programming Notes: None

Example:

```
3C 01 00 00     // lui $01, 0x0000
34 21 04 D2     // ori $01, 0x04D2      # LI   R01,   0x000004D2 (Decimal: 1234)
3C 01 00 00     // lui $02, 0x0000
34 21 16 2E     // ori $02, 0x162E      # LI   R02,   0x0000162E (Decimal: 5678)
00 21 00 18     // mult $r1, $r2        # R1*R2 = 0x006AE9BC (Decimal: 7,006,652)
00 00 20 12     // mflo $r4             # R4 = 0xE9BC
```

MULT Multiply Word

31 26	25 21	20 16	15 11	10 6	5 0
0x00	rs	rt	0x00	0x00	0x18
6	5	5	5	5	6

Format:

MULT rs, rt

Purpose:

To multiply 32-bit signed integers.

Description:

The 32-bit values in register $rt is multiplied by the 32-bit value in register $rs, treating both operands as signed values, to produce a 64-bit result. The low-order 32-bit word of the result is placed into special register LO, and the high-order 32-bit word is placed into special register HI.

Restrictions:

32-bit values in register $rt and $rs must be signed.

MFHI or MFLO will not allow you to do a multiply or a divide instruction within two instructions after it executed. The reason is it violates how the MIPS pipeline works.

Operation:

Fetch: PC <- PC +4; IR <- M[PC];

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

MULT: ALU_OUT <- RS * RT;

Exceptions:

None

Programming Notes:

None

Example:

```
3c 01 00 00    // lui $01, 0x0000
34 21 04 D2    // ori $01, 0x04D2     # LI  R01,  0x000004D2 (Decimal: 1234)
3c 01 00 00    // lui $02, 0x0000
34 21 16 2E    // ori $02, 0x162E     # LI  R02,  0x0000162E (Decimal: 5678)
00 21 00 18    // mult $r1, $r2       # R1*R2 = 0x006AE9BC (Decimal: 7,006,652)
```

DIV Divide Word

31 26	25 21	20 16	15 11	10 6	5 0
0x00	rs	rt	0x00	0x00	0x1A
6	5	5	5	5	6

Format:

DIV rs, rt

Purpose:

To divide 32-bit signed integers.

Description:

The 32-bit values in register $rt is divided by the 32-bit value in register $rs, treating both operands as signed values. The 32-bit quotient is placed into special register LO and the 32-bit remainder is placed into special register HI.

Restrictions:

32-bit values in register $rt and $rs must be signed.

MFHI or MFLO will not allow you to do a multiply or a divide instruction within two instructions after it executed. The reason is it violates how the MIPS pipeline works.

If the divisor in $rt is zero, the arithmetic result value is undefined.

Operation:

Fetch: PC <- PC +4; IR <- M[PC];

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

DIV: ALU_OUT <- RS / RT

Exceptions: None

Programming Notes: None

Example:

```
3C 01 00 00     // lui $01, 0x0000
34 21 DB 18     // ori $01, 0xDB18          # LI   R01,   0x0000DB18 (Decimal: 56088)
3C 01 00 00     // lui $02, 0x0000
34 21 01 C8     // ori $02, 0x01C8          # LI   R02,   0x000001C8 (Decimal: 456)
00 21 00 1A     // div $r1, $r2             # R1 / R2 = 0x0000007B (Decimal: 123)
```

ADD Add Word

31 26	25 21	20 16	15 11	10 6	5 0
0x00	rs	rt	rd	0x00	0x20
6	5	5	5	5	6

Format:

ADD rd, rs, rt

Purpose:

To add 32-bit integers.

Description: rd <- rs + rt

The 32-bit value in register $rt is added to the 32-bit value in register $rs to produce a 32-bit result. The 32-bit result is placed into register $rd.

They will be overflowed if we add two positive numbers and get a negative number or add two negative numbers and get a positive number.

Restrictions:

Values in register $rs and $rt must be signed.

Operation:

Fetch: PC <- PC +4; IR <- M[PC];

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

ADD: ALU_OUT <- RS + RT;

WB_ALU: Reg[IR[15-11]] <- ALU_OUT($rs + $rt)

Exceptions:

Integer Overflow

Programming Notes:

None

Example:

```
# No Overflow case:
3C 01 12 34       // lui $01, 0x1234
34 21 AB CD       // ori $01, 0xABCD                # LI   R01,   0x1234ABCD
3C 01 56 78       // lui $02, 0x5678
34 21 DC BA       // ori $02, 0xDCBA                # LI   R02,   0x5678DCBA
00 22 18 20       // add $r3, $r1, $r2              # R3 = 0x68AD8887

# Overflow case:
3C 01 7F FF       // lui $01, 0x7FFF
34 21 FF FF       // ori $01, 0xFFFF                # LI   R01,   0x7FFFFFFF
3C 01 00 00       // lui $02, 0x0000
34 21 00 02       // ori $02, 0x0002                # LI   R02,   0x00000002
00 22 18 20       // add $r3, $r1, $r2              # R3 = 0x80000001
```

ADDU Add Unsigned Word

31 26	25 21	20 16	15 11	10 6	5 0
0x00	rs	rt	rd	0x00	0x21
6	5	5	5	5	6

Format:

ADDU rd, rs, rt

Purpose:

To add 32-bit integers.

Description: rd <- rs + rt

The 32-bit value in register $rt is added to the 32-bit value in register $rs to produce a 32-bit result. The 32-bit result is placed into register $rd.

No Integer Overflow exception occurs under any circumstances.

Restrictions:

Values in register $rs and $rt must be unsigned**.**

Operation:

Fetch: PC <- PC +4; IR <- M[PC];

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

ADDU: ALU_OUT <- RS + RT;

WB_ALU: Reg[IR[15-11]] <- ALU_OUT($rs + $rt)

Exceptions:

None

Programming Notes:

None

Example:

```
3C 01 FF FF     // lui $01, 0xFFFF
34 21 FF 85     // ori $01, 0xFF85          # LI   R01,   0xFFFFFF85
3C 01 00 00     // lui $02, 0x0000
34 21 00 80     // ori $02, 0x0080          # LI   R02,   0x00000080
00 22 18 21     // Add $r3, $r1, $r2        # R3 = 0x00000005
```

SUB — Subtract Word

31 26	25 21	20 16	15 11	10 6	5 0
0x00	rs	rt	rd	0x00	0x22
6	5	5	5	5	6

Format:

SUB rd, rs, rt

Purpose:

To subtract 32-bit signed integers.

Description: rd ← rs – rt

The 32-bit value in register $rt is subtracted to the 32-bit value in register $rs to produce a 32-bit result. The 32-bit difference is placed into register $rd.

They will be overflowed if we subtract two different signed numbers and get a positive number or subtract two different signed numbers and get a negative number. For example, we take a negative number to subtract a positive number and we get a positive number. We get overflow.

Restrictions:

Values in register $rs and $rt must be signed.

Operation:

Fetch: PC <- PC +4; IR <- M[PC];

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

SUB: ALU_OUT <- RS - RT;

WB_ALU: Reg[IR[15-11]] <- ALU_OUT($rs - $rt)

Exceptions:

Integer overflow.

Programming Notes:

None

Example:

```
# No Overflow case
3C 01 00 00     // lui $01, 0x0000
34 21 04 D2     // ori $01, 0x04D2              # LI   R01,   0x000004D2
3C 01 FF FF     // lui $02, 0xFFFF
34 21 FC 18     // ori $02, 0xFC18              # LI   R02,   0xFFFFFC18
00 22 18 22     // sub $r3, $r1, $r2            # R3 = 0x000008BA

# Overflow case
3C 01 FF FF     // lui $01, 0xFFFF
34 21 EF 1F     // ori $01, 0xEF1F              # LI   R01,   0xFFFFEF1F (Decimal: -4321)
3C 01 00 00     // lui $02, 0x0000
34 21 04 D2     // ori $02, 0x04D2              # LI   R02,   0x000004D2 (Decimal: 1234)
00 22 18 22     // sub $r3, $r1, $r2            # R3 = 0x00000C0F (Decimal: 3087)
```

SUBU Subtract Unsigned Word

31 26	25 21	20 16	15 11	10 6	5 0
0x00	rs	rt	rd	0x00	0x23
6	5	5	5	5	6

Format:

SUBU rd, rs, rt

Purpose:

To subtract 32-bit unsigned integers.

Description: rd ← rs – rt

The 32-bit value in register $rt is subtracted to the 32-bit value in register $rs to produce a 32-bit result. The 32-bit difference is placed into register $rd.

They will be overflowed if we subtract two different signed numbers and get a positive number or subtract two different signed numbers and get a negative number.

Restrictions:

Values in register $rs and $rt must be unsigned.

Operation:

Fetch: PC <- PC +4; IR <- M[PC];

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

SUBU: ALU_OUT <- RS - RT;

WB_ALU: Reg[IR[15-11]] <- ALU_OUT($rs - $rt)

Exceptions:

None

Programming Notes:

None

Example:

```
3C 01 00 00    // lui $01, 0x0000
34 21 04 D2    // ori $01, 0x04D2          # LI   R01,   0x000004D2
3C 01 00 00    // lui $02, 0x0000
34 21 00 EA    // ori $02, 0x00EA          # LI   R02,   0x000000EA
00 22 18 23    // sub $r3, $r1, $r2        # R3 = 0x000003E8
```

AND Logic And

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

0x00	rs	rt	rd	0x00	0x24
6	5	5	5	5	6

Format:

AND rd, rs, rt

Purpose:

To do a bitwise logical AND.

Description: rd ← rs AND rt

The 32-bit value in register $rs are combined with the 32-bit value in register $rt in a bitwise logical AND operation. The result is placed into register $rd.

Restrictions:

None

Operation:

Fetch: PC <- PC +4; IR <- M[PC];

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

AND: ALU_OUT <- RS and RT;

WB_ALU: Reg[IR[15-11]] <- ALU_OUT($rs and $rt)

Exceptions:

None

Programming Notes:

None

Example:

```
3C 01 00 00     // lui $01, 0x0000
34 21 AB CD     // ori $01, 0xABCD          # LI   R01,   0x0000ABCD
3C 01 00 00     // lui $02, 0x0000
34 21 12 34     // ori $02, 0x1234          # LI   R02,   0x00001234
00 22 18 24     // and $r3, $r1, $r2        # R3 = 0x00000204
```

OR Logic Or

31 26	25 21	20 16	15 11	10 6	5 0
0x00	rs	rt	rd	0x00	0x25
6	5	5	5	5	6

Format:

OR rd, rs, rt

Purpose:

To do bitwise logical OR.

Description: rd ← rs OR rt

The 32-bit value in register $rs are combined with the 32-bit value in register $rt in a bitwise logical OR operation. The result is placed into register $rd.

Restrictions:

None

Operation:

Fetch: PC <- PC +4; IR <- M[PC];

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

OR: ALU_OUT <- RS or RT;

WB_ALU: Reg[IR[15-11]] <- ALU_OUT($rs or $rt)

Exceptions:

None

Programming Notes:

None

Example:

```
3C 01 00 00     // lui $01, 0x0000
34 21 AB CD     // ori $01, 0xABCD          # LI   R01,   0x0000ABCD
3C 01 00 00     // lui $02, 0x0000
34 21 12 34     // ori $02, 0x1234          # LI   R02,   0x00001234
00 22 18 24     // or $r3, $r1, $r2         # R3 = 0x0000BBFD
```

XOR Logic Exclusive Or

31 26	25 21	20 16	15 11	10 6	5 0
0x00	rs	rt	rd	0x00	0x26
6	5	5	5	5	6

Format:

XOR rd, rs, rt

Purpose:

To do bitwise logical Exclusive OR.

Description: rd ← rs XOR rt

The 32-bit value in register $rs are combined with the 32-bit value in register $rt in a bitwise logical Exclusive OR operation. The result is placed into register $rd.

Restrictions:

None

Operation:

Fetch: PC <- PC +4; IR <- M[PC];

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

XOR: ALU_OUT <- RS xor RT;

WB_ALU: Reg[IR[15-11]] <- ALU_OUT($rs xor $rt)

Exceptions:

None

Programming Notes:

None

Example:

```
3C 01 00 00     // lui $01, 0x0000
34 21 AB CD     // ori $01, 0xABCD          # LI   R01,   0x0000ABCD
3C 01 00 00     // lui $02, 0x0000
34 21 12 34     // ori $02, 0x1234          # LI   R02,   0x00001234
00 22 18 26     // xor $r3, $r1, $r2        # R3 = 0x0000B9F9
```

NOR Logic Not Or

31 26	25 21	20 16	15 11	10 6	5 0
0x00	rs	rt	rd	0x00	0x27
6	5	5	5	5	6

Format:

NOR rd, rs, rt

Purpose:

To do a bitwise logical NOT OR.

Description: rd ← rs NOR rt

The 32-bit value in register $rs are combined with the 32-bit value in register $rt in a bitwise logical NOT OR operation. The result is placed into register $rd.

Restrictions:

None

Operation:

Fetch: PC <- PC +4; IR <- M[PC];

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

NOR: ALU_OUT <- RS nor RT;

WB_ALU: Reg[IR[15-11]] <- ALU_OUT($rs nor $rt)

Exceptions:

None

Programming Notes:

None

Example:

```
3C 01 00 00     // lui $01, 0x0000
34 21 AB CD     // ori $01, 0xABCD          # LI   R01,   0x0000ABCD
3C 01 00 00     // lui $02, 0x0000
34 21 12 34     // ori $02, 0x1234          # LI   R02,   0x00001234
00 22 18 27     // nor $r3, $r1, $r2        # R3 = 0xFFFF4402
```

SLT Set Less Than

31 26	25 21	20 16	15 11	10 6	5 0
0x00	rs	rt	rd	0x00	0x2A
6	5	5	5	5	6

Format:

SLT rd, rs, rt

Purpose:

To record the result of a less-than comparison.

Description: rd ← (rs < rt)

Compare the contents of register $rs and register $rt as signed integers and record the Boolean result of the comparison in register $rd. if $rs is less than $rt the result is true, otherwise the result is false.

Restrictions:

The 32-bit values of register $rs and $rt must be singed.

Operation:

Fetch: PC <- PC +4; IR <- M[PC];

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

SLT: ALU_OUT <- (RS < RT) ? 1 : 0;

WB_ALU: Reg[IR[15-11]] <- ALU_OUT($rs < $rt) ? 1 : 0)

Exceptions:

None

Programming Notes:

None

Example:

```
# TRUE
3C 01 FF FF     // lui $01, 0xFFFF
34 21 FF D1     // ori $01, 0xFFD1              # LI   R01,   0xFFFFFFD1 (Decimal: -47)
3C 01 00 00     // lui $02, 0x0000
34 21 00 45     // ori $02, 0x0045              # LI   R02,   0x00000045 (Decimal: 69)
00 22 18 2A     // slt $r3, $r1, $r2            # R3 = 1

# FALSE
3C 01 FF FF     // lui $01, 0xFFFF
34 21 FF 85     // ori $01, 0xFF85              # LI   R01,   0xFFFFFF85 (Decimal: -123)
3C 01 FF FF     // lui $02, 0xFFFF
34 21 FC EB     // ori $02, 0xFCEB              # LI   R02,   0xFFFFFCEB (Decimal: -789)
00 22 18 2A     // slt $r3, $r1, $r2            # R3 = 0
```

SLTU Set Less Than Unsigned

31 26	25 21	20 16	15 11	10 6	5 0
0x00	rs	rt	rd	0x00	0x2B
6	5	5	5	5	6

Format:

SLTU rd, rs, rt

Purpose:

To record the result of an unsigned less-than comparison.

Description: rd ← (rs < rt)

Compare the contents of register $rs and register $rt as unsigned integers and record the Boolean result of the comparison in register $rd. if $rs is less than $rt the result is true, otherwise the result is false.

Restrictions:

The 32-bit values of register $rs and $rt must be unsinged**.**

Operation:

Fetch: PC <- PC +4; IR <- M[PC];

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

SLT: ALU_OUT <- (RS < RT) ? 1 : 0;

WB_ALU: Reg[IR[15-11]] <- ALU_OUT($rs < $rt) ? 1 : 0)

Exceptions:

None

Programming Notes:

None

Example:

```
# TRUE
3C 01 00 00      // lui $01, 0x0000
34 21 00 23      // ori $01, 0x0023              # LI   R01,   0x00000023 (Decimal: 35)
3C 01 00 00      // lui $02, 0x0000
34 21 00 45      // ori $02, 0x0044              # LI   R02,   0x00000044 (Decimal: 68)
00 22 18 2B      // slt $r3, $r1, $r2            # R3 = 1

# FALSE
3C 01 00 00      // lui $01, 0x0000
34 21 00 63      // ori $01, 0x0063              # LI   R01,   0x00000063 (Decimal: 99)
3C 01 00 00      // lui $02, 0x0000
34 21 00 2F      // ori $02, 0x002F              # LI   R02,   0x0000002F (Decimal: 47)
00 22 18 2B      // slt $r3, $r1, $r2            # R3 = 0
```

BREAK

31 26	25 21	20 16	15 11	10 6	5 0
0x00	0x00	0x00	0x00	0x00	0x0D
6	5	5	5	5	6

Format:

BREAK

Purpose:

To stop the instruction.

Description:

It will display “Break Instruction Fetched” with the current time. Then, lets us know that we get a safe break. Finally, it will do the Dump_Registers and Mem_Dump taskes, which will display the 32 bit MIPS registers and MIPS Data Memory.

Restrictions: None

Operation:

Fetch: PC <- PC +4; IR <- M[PC];

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

NS <- BREAK;

Exceptions: None

Programming Notes: None

Example:

```
0C 10 00 15    //              jal  mem2mem
3C 0F FF FF    //              lui  $15, 0xFFFF
35 EF FF FF    //              ori  $15, 0xFFFF     # LI   R15,  0xFFFFFFFF  "pass flag"
00 00 00 0D    //              break
8D F1 00 00    // mem2mem:     lw   $17, 00($15)              # do mem to
AD D1 00 00    //              sw   $17, 00($14)              # mem transfer
21 EF 00 04    //              addi $15, $15, 04              # bump both source
21 CE 00 04    //              addi $14, $14, 04              # and dest pointers
21 AD FF FF    //              addi $13, $13, -1              # dec the loop counter
15 A0 FF FA    //              bne  $13, $00, mem2mem         # and continue till done
03 E0 00 08    //              jr   $31                       # return to calling code
00 00 00 0D    //              break                          # safety net
```

SETIE

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

0x00	0x00	0x00	0x00	0x00	0x1F
6	5	5	5	5	6

Format:

SETIE

Purpose:

To set interrupt enable flag.

Description:

We will update our present interrupt enable flag state to 1’b1 and pass it to our next interrupt enable flag state. Then, we go to FETCH.

Restrictions: None

Operation:

```
Fetch: PC <- PC +4; IR <- M[PC];

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

        NS < SETIE;

SETIE: NS <- FETCH;
```

Exceptions: None

Programming Notes: None

Example:

```
@0
00 00 00 1F      // main: setie
@200
3C 10 10 01      // isr:   lui   $16, 0x1001       # load destination IO address
36 10 00 C0      //        ori   $16, 0x00C0       # 0x100100C0 into r16
3C 11 80 00      //        lui   $17, 0x8000       # initialize the pattern of
36 31 FF FF      //        ori   $17, 0xFFFF       # 0x8000FFFF into r17
20 12 00 10      //        addi  $18, $0, 0x10     # loop counter set to 16
```

I – Type Instructions

BEQ Branch Equal

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

0x04	rs	rt	16 – bit immediate
6	5	5	16

Format:

BEQ $rs, $rt, branch address

Purpose:

Branch if the values of two registers $rs and $rs are equal.

Description:

We compare the 32-bit value of register $rs with the 32-bit value of register $rt. If they are equal, we will jump to the calculated branch address. To calculate the branch address, we will shift left the 16-bit immediate by 2. After that, we will take the current PC to add with result and load it to the PC to jump to that address.

Branch address = {PC + {Signed-Extend 16-bit Immediate[29-0],2'b00}}.

Restrictions:

None

Operation:

```
Fetch: PC <- PC +4; IR <- M[PC];
Decode: RS <- Reg[IR[25-21]]; RT <- Reg[IR[20-16]];
        ALU_OUT <- PC + (sign-extend(IR[15-0]) << 2);
BEQ: ALU_OUT <- RS – RT;
BEQ2: if(zero) PC <- ALU_OUT(PC + (sign-extend(IR[15-0]) << 2));
```

Exceptions:

None

Programming Notes:

None

Example:

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

10 22 00 01     //           beq  $01, $02, no_eq    # should not branch
10 23 00 03     //           beq  $01, $03, yes_eq   # should branch
3C 0E FF FF     // no_eq:    lui  $14, 0xFFFF
35 CE FF FF     //           ori  $14, 0xFFFF        # LI   R14,  0xFFFFFFFF  "fail flag"
00 00 00 0D     //           break
00 00 70 20     // yes_eq:   add  $14, $0, $0        # CLR  R14  "pass flag"
```

BNE Branch Not Equal

31 26	25 21	20 16	15 11 10 6 5 0
0x05	rs	rt	16 – bit immediate
6	5	5	16

Format:

BNE $rs, $rt, branch address

Purpose:

Branch if the values of two registers $rs and $rs are not equal.

Description:

We compare the 32-bit value of register $rs with the 32-bit value of register $rt. If they are not equal, we will jump to the calculated branch address. To calculate the branch address, we will shift left the 16-bit immediate by 2. After that, we will take the current PC to add with result and load it to the PC to jump to that address.

Branch address = {PC + {Signed-Extend 16-bit Immediate[29-0],2'b00}}.

Restrictions:

None

Operation:

```
Fetch: PC <- PC +4; IR <- M[PC];
Decode: RS <- Reg[IR[25-21]]; RT <- Reg[IR[20-16]];
        ALU_OUT <- PC + (sign-extend(IR[15-0]) << 2);
BNE: ALU_OUT <- RS – RT;
BNE2: if(~zero) PC <- ALU_OUT(PC + (sign-extend(IR[15-0]) << 2));
```

Exceptions:

None

Programming Notes:

None

Example:

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

14 23 00 01    //           bne  $01, $03, no_ne     # should not branch
14 22 00 03    //           bne  $01, $02, yes_ne    # should branch
3C 0F FF FF    // no_ne:    lui  $15, 0xFFFF
35 EF FF FF    //           ori  $15, 0xFFFF         # LI   R15,  0xFFFFFFFF  "fail flag"
00 00 00 0D    //           break
00 00 78 20    // yes_ne:   add  $15, $0, $0         # CLR  R15  "pass flag"
```

BLEZ Branch Less Than or Equal Zero

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

0x06	rs	rt	16 – bit immediate
6	5	5	16

Format:

BLEZ $rs, $rt, branch address

Purpose:

Branch if the value of register $rt are less than or equal ZERO.

Description:

We input 0 into register $rs and compare with the 32-bit value of register $rt. If the flags are negative or zero, we will jump to the calculated branch address. To calculate the branch address, we will shift left the 16-bit immediate by 2. After that, we will take the current PC to add with result and load it to the PC to jump to that address.

Branch address = {PC + {Signed-Extend 16-bit Immediate[29-0],2'b00}}.

Restrictions:

None

Operation:

```
Fetch: PC <- PC +4; IR <- M[PC];
Decode: RS <- Reg[IR[25-21]]; RT <- Reg[IR[20-16]];
        ALU_OUT <- PC + (sign-extend(IR[15-0]) << 2);
BLEZ: ALU_OUT <- RT <= RS;
BLEZ2: if(zero || negative) PC <- ALU_OUT(PC + (sign-extend(IR[15-0]) << 2));
```

Exceptions:

None

Programming Notes:

None

Example:

```
20 03 00 C0     // blez_p1:     addi $03, $00, 0xC0     # pass flag1 M[C0] <-- C0
AD E3 00 00     //              sw   $03, 0x00($15)
18 40 00 03     //              blez $02, blez_f2       # this should not branch
20 04 00 C4     //              addi $04, $00, 0xC4     # pass flag2 M[C4] <-- C4
AD E4 00 04     //              sw   $04, 0x04($15)
08 10 00 13     //              j    blez_p2
20 0E FF FE     // blez_f2:     addi $14, $00, -2       # fail flag2 r14 <-- FFFF_FFFE
00 00 00 0D     //              break
18 00 00 02     // blez_p2:     blez $0, blez_p3        # this should branch
20 0E FF FD     //              addi $14, $00, -3       # fail flag3 r14 <-- FFFF_FFFD
00 00 00 0D     //              break
20 05 00 C8     // blez_p3:     addi $05, $00, 0xC8     # pass flag3 M[C8] <-- C8
AD E5 00 08     //              sw   $05, 0x08($15)
```

BGTZ Branch Greater Than or Equal Zero

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

31 26	25 21	20 16	15 11 10 6 5 0
0x07	rs	rt	16 – bit immediate
6	5	5	16

Format:

BGTZ $rs, $rt, branch address

Purpose:

Branch if the value of register $rt are greater than or equal ZERO.

Description:

We input 0 into register $rs and compare with the 32-bit value of register $rt. If the flags are non-negative or non-zero, we will jump to the calculated branch address. To calculate the branch address, we will shift left the 16-bit immediate by 2. After that, we will take the current PC to add with result and load it to the PC to jump to that address.

Branch address = {PC + {Signed-Extend 16-bit Immediate[29-0],2'b00}}.

Restrictions:

None

Operation:

```
Fetch: PC <- PC +4; IR <- M[PC];
Decode: RS <- Reg[IR[25-21]]; RT <- Reg[IR[20-16]];
        ALU_OUT <- PC + (sign-extend(IR[15-0]) << 2);
BGTZ: ALU_OUT <- RT >= RS;
BGTZ2: if(~zero || ~negative) PC <- ALU_OUT(PC + (sign-extend(IR[15-0]) << 2));
```

Exceptions:

None

Programming Notes:

None

Example:

```
1C 40 00 02     //              bgtz $02, bgtz_p1       # this should pass
20 0E FF FC     //              addi $14, $00, -4       # fail flag3 r14 <-- FFFF_FFFC
00 00 00 0D     //              break
20 06 00 CC     // bgtz_p1:     addi $06, $00, 0xCC     # pass flag4 M[C0] <-- CC
AD E6 00 0C     //              sw   $06, 0x0C($15)
1C 20 00 03     //              bgtz $01, bgtz_f2       # this should not branch
20 07 00 D0     //              addi $07, $00, 0xD0     # pass flag5 M[D0] <-- D0
AD E7 00 10     //              sw   $07, 0x10($15)
08 10 00 23     //              j    bgtz_p2
20 0E FF FB     // bgtz_f2:     addi $14, $00, -5       # fail flag5 r14 <-- FFFF_FFFB
00 00 00 0D     //              break
1C 20 00 03     // bgtz_p2:     bgtz $01, bgtz_f3       # this should not branch
20 08 00 D4     //              addi $08, $00, 0xD4     # pass flag6 M[D0] <-- D4
AD E8 00 14     //              sw   $08, 0x14($15)
08 10 00 29     //              j    bgtz_p3
20 0E FF FA     // bgtz_f3:     addi $14, $00, -6       # fail flag6 r14 <-- FFFF_FFFA
00 00 00 0D     //              break
20 0E 00 00     // bgtz_p3:     addi $14, $00, 0        # set $r14 to 0000_0000
```

ADDI Add Immediate

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

0x08	rs	rt	16 – bit immediate
6	5	5	16

Format:

ADDI rt, rs, immediate.

Purpose:

To add 32-bit integers with a constant.

Description: rt <- rs + immediate

The sign-extend 16-bit immediate value is added to the 32-bit value in register $rs to produce a 32-bit result. The 32-bit result is placed into register $rt.

They will be overflowed if we add two positive numbers and get a negative number or add two negative numbers and get a positive number.

Restrictions:

Values in register $rs and immediate must be signed.

Operation:

Fetch: PC <- PC +4; IR <- M[PC];

Decode: RS <- Reg[IR[25-21]]; RT <- sign-extend(IR[15-0]);

ADDI: ALU_OUT <- RS + immediate;

WB_IMM: Reg[IR[20-16]] <- ALU_OUT($rs + immediate);

Exceptions:

Integer Overflow

Programming Notes:

None

Example:

```
20 02 00 10     // addi $02, $00, 0x10          # LI   R02,  0x10
20 42 FF FF     // addi $02, $02, -1            # Decrement R02 by 1
```

SLTI Set on Less Than Immediate

31 26	25 21	20 16	15 11 10 6 5 0
0x0A	rs	rt	16 – bit immediate
6	5	5	16

Format:

SLTI rt, rs, immediate.

Purpose:

To record the result of a less-than comparison with a constant.

Description: rt ← (rs < immediate)

Compare the contents of register $rs and the sign-extend 16-bit immediate value as signed integers and record the Boolean result of the comparison in register $rt. if $rs is less than immediate the result is true, otherwise the result is false.

Restrictions:

The 32-bit values of register $rs and immediate must be singed.

Operation:

Fetch: PC <- PC +4; IR <- M[PC];

Decode: RS <- Reg[IR[25-21]]; RT <- sign-extend(IR[15-0]);

SLT: ALU_OUT <- (RS < RT) ? 1 : 0;

WB_IMM: Reg[IR[20-16]] <- ALU_OUT(($rs < $rt) ? 1 : 0);

Exceptions:

None

Programming Notes:

None

Example:

```
20 02 FF F0     // addi $02, $00, -16     # LI   R02,  -16
20 42 00 01     // addi $02, $02, 1       # increment the loop counter
28 43 00 00     // slti $03, $02, 0       # r3 <--1 if r2 < 0
```

SLTIU Set on Less Than Immediate Unsigned

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

0x0B	rs	rt	16 – bit immediate
6	5	5	16

Format:

SLTIU rt, rs, immediate

Purpose:

To record the result of a less-than comparison with a constant.

Description: rt ← (rs < immediate)

Compare the contents of register $rs and the sign-extend 16-bit immediate value as unsigned integers and record the Boolean result of the comparison in register $rt. if $rs is less than immediate the result is true, otherwise the result is false.

Restrictions:

The 32-bit values of register $rs and immediate must be unsinged.

Operation:

```
Fetch: PC <- PC +4; IR <- M[PC];

Decode: RS <- Reg[IR[25-21]]; RT <- sign-extend(IR[15-0]);

SLT: ALU_OUT <- (RS < RT) ? 1 : 0;

WB_IMM: Reg[IR[20-16]] <- ALU_OUT(($rs < $rt) ? 1 : 0);
```

Exceptions: None

Programming Notes: None

Example:

```
20 02 FF F0     // addi $02, $00, 0         # LI   R02, 0
20 42 00 01     // addi $02, $02, 1         # increment the loop counter
2C 43 00 64     // sltiu $03, $02, 100      # r3 <-- 1 if r2 < 100
```

ANDI And Immediate

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

0x0C	rs	rt	16 – bit immediate
6	5	5	16

Format:

ANDI rt, rs, immediate

Purpose:

To do a bitwise logical AND with a constant.

Description: rt ← rs AND immediate

The 32-bit value in register $rs are combined with the sign-extend 16-bit immediate in a bitwise logical AND operation. The result is placed into register $rt.

Restrictions:

None

Operation:

Fetch: PC <- PC +4; IR <- M[PC];

Decode: RS <- Reg[IR[25-21]]; RT <- sign-extend(IR[15-0]);

AND: ALU_OUT <- RS and immediate;

WB_IMM: Reg[IR[20-16]]<- ALU_OUT($rs and immediate)

Exceptions:

None

Programming Notes:

None

Example:

```
20 01 FF 00     // addi $01, $00, 65280        # LI   R01, 0xFF00
30 22 F0 00     // andi $01, $02, 61440        # R2 <- 0xF000
andi:
R1:      1111_1111_0000_0000
IMM:     1111_0000_0000_0000
R2:      1111_0000_0000_0000
```

ORI   Or Immediate

31 26	25 21	20 16	15 11 10 6 5 0
0x0D	rs	rt	16 – bit immediate
6	5	5	16

Format:

ORI rt, rs, immediate

Purpose:

To do a bitwise logical OR with a constant.

Description: rt ← rs OR immediate

The 32-bit value in register $rs are combined with the sign-extend 16-bit immediate in a bitwise logical OR operation. The result is placed into register $rt.

Restrictions:

None

Operation:

Fetch: PC <- PC +4; IR <- M[PC];

Decode: RS <- Reg[IR[25-21]]; RT <- sign-extend(IR[15-0]);

AND: ALU_OUT <- RS or immediate;

WB_IMM: Reg[IR[20-16]]<- ALU_OUT($rs or immediate)

Exceptions:

None

Programming Notes:

None

Example:

```
20 01 FF 00      // addi  $01, $00, 65280          # LI   R01, 0xFF00
34 23 F0 00      // ori     $01, $03, 61440          # R3 <- 0xFF00
ori:
R1:       1111_1111_0000_0000
IMM:     1111_0000_0000_0000
R3:       1111_1111_0000_0000
```

XORI Exclusive OR Immediate

31 26	25 21	20 16	15 11 10 6 5 0
0x0E	rs	rt	16 – bit immediate
6	5	5	16

Format:

XORI rt, rs, immediate

Purpose:

To do a bitwise logical Exclusive OR with a constant.

Description: rt ← rs XOR immediate

The 32-bit value in register $rs are combined with the sign-extend 16-bit immediate in a bitwise logical Exclusive OR operation. The result is placed into register $rt.

Restrictions:

None

Operation:

Fetch: PC <- PC +4; IR <- M[PC];

Decode: RS <- Reg[IR[25-21]]; RT <- sign-extend(IR[15-0]);

AND: ALU_OUT <- RS xor immediate;

WB_IMM: Reg[IR[20-16]]<- ALU_OUT($rs xor immediate)

Exceptions:

None

Programming Notes:

None

Example:

```
20 01 FF 00     // addi  $01, $00, 65280        # LI   R01, 0xFF00
38 24 F0 00     // xori  $01, $04, 61440        # R3 <- 0x0F00
```

<u>xori:</u>

```
R1:     1111_1111_0000_0000
IMM:    1111_0000_0000_0000
R3:     0000_1111_0000_0000
```

LUI Load Upper Immediate

31 26	25 21	20 16	15 11 10 6 5 0
0x0F	rs	rt	16 – bit immediate
6	5	5	16

Format:

LUI rt, immediate

Purpose:

To load a constant into the upper half of a word.

Description: rt ← immediate || 0^{16}

The 16-bit immediate is shifted left 16 bits and concatenated with 16 bits of low-order zeros. The 32-bit result is placed into register rt.

Restrictions:

None

Operation:

Fetch: PC <- PC +4; IR <- M[PC];

Decode: RS <- Reg[IR[25-21]]; RT <- sign-extend(IR[15-0]);

LUI: ALU_OUT <- (RT[15-0], 16’h0);

WB_IMM: Reg[IR[20-16]]← ALU_OUT(RT[15-0], 16’h0);

Exceptions:

None

Programming Notes:

None

Example:

```
3C 01 FF FF     // lui  $01, 0xFFFF
34 21 FF FF     // ori  $01, 0xFFFF     # LI   R01,  0xFFFFFFF
3C 02 87 65     // lui  $02, 0x8765
34 42 43 21     // ori  $02, 0x4321     # LI   R02,  0x87654321
3C 0D 10 01     // lui  $13, 0x1001
35 AD 00 C0     // ori  $13, 0x00C0     # LI   R13,  0x100100C0
```

LW Load Word

31 26	25 21	20 16	15 11 10 6 5 0
0x23	rs	rt	16 – bit immediate
6	5	5	16

Format:

LW rt, offset(base)

Purpose:

To load a word from memory.

Description: rt ← memory[base + offset]

The 16-bit signed offset is added to the contents of register base usually from register $rs to form the effective address. The 32-bit word from that effective address will be loaded and placed into register $rt.

Restrictions:

None

Operation:

Fetch: PC <- PC +4; IR <- M[PC];

Decode: RS <- Reg[IR[25-21]]; RT <- sign-extend(IR[15-0]);

LW1: ALU_OUT <- RS + RT;

LW2: dMem[Addr] <- ALU_OUT($rs +$rt)

LW3: Reg[IR[20-16]] <- dMem[$rs+$rt];

Exceptions:

None

Programming Notes:

None

Example:

```
3C 01 FF FF      // lui  $01, 0xFFFF
34 21 FF FF      // ori  $01, 0xFFFF      # LI   R01,  0xFFFFFFF
AD E1 00 00      // lw   $01, 0($15)      # ST  [R15], R01
```

SW Store Word

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

0x2B	rs	rt	16 – bit immediate
6	5	5	16

Format:

SW rt, offset(base)

Purpose:

To store a word to memory.

Description: memory[base + offset] ← rt

The 16-bit signed offset is added to the contents of register base usually from register $rs to form the effective address. The 32-bit word from that effective address will be loaded and placed into register $rt.

Restrictions:

None

Operation:

Fetch: PC <- PC +4; IR <- M[PC];

Decode: RS <- Reg[IR[25-21]]; RT <- sign-extend(IR[15-0]);

SW1: ALU_OUT <- RS + RT;

SW2: dMem[Addr] <- ALU_OUT($rs +$rt);

Exceptions:

None

Programming Notes:

None

Example:

```
3C 0F 10 01     // lui  $15, 0x1001
35 EF 00 00     // ori  $15, 0x0000     # $r15 <-- 0x10010000  (source pointer)
8D E1 00 00     // lw   $01, 00($15)    # $r01 <-- 264465 (from memory)
```

J – Type Instructions:

J Jump

31 26	25 21 20 16 15 11 10 6 5 0
0x02	26 – bit Jump Address
6	26

Format:

J target

Purpose:

To jump to a target address.

Description:

The j instruction loads an immediate value into the PC register. This immediate value is either a numeric offset or a label (and the assembler converts the label into an offset).

Restrictions:

None

Operation:

Fetch: PC <- PC +4; IR <- M[PC];

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

J: PC <- {PC[31-28], IR[25-0], 2'b00};

Exceptions:

None

Programming Notes:

None

Example:

```
08 10 00 0C    // j exit
               // exit:
3C 0E 01 01    // lui $14, 0x0101
35 CE 10 10    // ori $14, 0x1010        # LI   R14, 0x01011010
00 00 00 0D    // break
```

JAL Jump and Link

31 26	25 21 20 16 15 11 10 6 5 0
0x03	26 – bit Jump Address
6	26

Format:

JAL target

Purpose:

To jump to a subroutine, execute, and return.

Description:

Like the j instruction, except that the return address is loaded into the $ra register. This allows a subroutine to return to the main body routine after completion.

Restrictions:

None

Operation:

Fetch: PC <- PC +4; IR <- M[PC];

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

JAL: PC <- {PC[31-28], IR[25-0], 2'b00}; $31 <- PC + 8;

Exceptions:

None

Programming Notes:

None

Example:

```
0C 10 00 1A     // jal  sltiu_tests
                // sltiu_tests:
2C 23 FF 8B     // sltiu  $03, $01, -117        # for unsigned# r01 < se(0xFF8B)
14 60 00 02     // bne    $03, $00, slt1_p1     # thus, we should branch
20 0E FF FF     // addi   $14, $00, -1          # fail flag1 r14 <-- FFFF_FFFF
00 00 00 0D     // break
```

Enhanced Instructions:

SLA Shift Left Arithmetic

31 26	25 21	20 16	15 11	10 6	5 0
0x00	0x00	rt	rd	shamt	0x35
6	5	5	5	5	6

Format:

SLA rd, rt, shamt

Purpose:

To arithmetic left shift a word by a fixed number of bits.

Description:

The value that register $rt holds is shifted to the left as many times as shamt specifies. Bits with the value of the sign bit(bit 31) are inserted into the least significant bit as shifting occurs to replace empty bits. The result is placed in register $rd.

Restrictions: None

Operation:

Fetch: PC <- PC +4; IR <- M[PC];

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

SRL: ALU_OUT <- RT << shamt;

WB_alu: Reg[IR[15-11]] <- ALU_OUT(RT << shamt);

Exceptions: None

Programming Notes: None

Example:

```
3C 01 FF FF     // lui  $01, 0xFFFF
34 21 80 00     // ori  $01, 0x8000        # LI   R01,  0xFFFF8000
                // R01 = 1111_1111_1111_1111_1000_0000_0000_0000
00 01 10 B5     // sla  $01, $02, 2        # shift left arithmetic by 2
                // R02 = 1111_1111_1111_1110_0000_0000_0000_0000
```

ROL Rotation Left without Carry

31 26	25 21	20 16	15 11	10 6	5 0
0x00	0x00	rt	rd	shamt	0x36
6	5	5	5	5	6

Format:

ROL rd, rt, shamt

Purpose:

To rotate a word to the left by a certain number of bits.

Description:

The value that register $rt holds is rotated to the left as many times as shamt specifies. The most significant bit and the least significant bit will be wrapped around. The result is placed in register $rd.

Restrictions: None

Operation:

Fetch: PC <- PC +4; IR <- M[PC];

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

SLL: ALU_OUT <- RT << shamt;

WB_alu: Reg[IR[15-11]] <- ALU_OUT(RT << shamt);

Exceptions: None

Programming Notes: None

Example:

```
3C 01 FF FF     // lui  $01, 0xFFFF
34 21 80 00     // ori  $01, 0x8000       # LI   R01,  0xFFFF8000
                // R01 = 1111_1111_1111_1111_1000_0000_0000_0000
00 01 19 76     // rol $01, $03, 5        # rotate left R01 by 5. R03 <- 0xFFC0001F
                // R03 = 1111_1111_1100_0000_0000_0000_0001_1111
```

ROR Rotation Right without Carry

31 26	25 21	20 16	15 11	10 6	5 0
0x00	0x00	rt	rd	shamt	0x37
6	5	5	5	5	6

Format:

ROR rd, rt, shamt

Purpose:

To rotate a word to the right by a certain number of bits.

Description:

The value that register $rt holds is rotated to the right as many times as shamt specifies. The most significant bit and the least significant bit will be wrapped around. The result is placed in register $rd.

Restrictions: None

Operation:

Fetch: PC <- PC +4; IR <- M[PC];

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

SLL: ALU_OUT <- RT >> shamt;

WB_alu: Reg[IR[15-11]] <- ALU_OUT(RT >> shamt);

Exceptions: None

Programming Notes: None

Example:

```
3C 01 FF FF    // lui  $01, 0xFFFF
34 21 80 00    // ori  $01, 0x8000       # LI   R01,  0xFFFF8000
               // R01 = 1111_1111_1111_1111_1000_0000_0000_0000
00 01 19 76    // rol $01, $03, 5        # rotate left R01 by 5. R03 <- 0xFFC0001F
               // R03 = 1111_1111_1100_0000_0000_0000_0001_1111
00 03 21 77    // ror $03, $04, 5        # rotate right R03 by 5. R04 <- 0xFFFF8000
               // R04 = 1111_1111_1111_1111_1000_0000_0000_0000
```

INPUT

31 26	25 21	20 16	15 11 10 6 5 0
0x1C	rs	rt	16 – bit immediate
6	5	5	16

Format:

INPUT rt offset(rs)

Purpose:

To receive an input from IO and load it into a GPR

Description: rt <- ioM[rs + offset]

The 32-bit value in register $rs is added to the sign-extended 16-bit immediate value to produce a memory address to read from in the IO memory. The 32-bit value that was read is then loaded into register $rt.

Restrictions:

Operation:

Fetch: PC <- PC +4; IR <- M[PC];

Decode: RS <- Reg[IR[25-21]]; RT <- IR[15:0]

INPUT: ALU_OUT <- RS($rs) + RT(IR[15:0])

INPUT2: D_in <- ioM[ALU_OUT($rs+imm)]

INPUT3: $rt <- D_in(ioM[($rs+imm)]

Exceptions:

Programming Notes:

Example:

```
addi $3, $0, 03          # $3 <- 0x00000003
input  $5,  0($3)        # $5 <- ioM[0x00000003]
```

OUTPUT

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

0x1D	rs	rt	16 – bit immediate
6	5	5	16

Format:

OUTPUT rt imm16(rs)

Purpose:

To store a word from a GPR into IO Memory.

Description: ioM[rs + imm] <- rt

The 32-bit value in $rs will be added to the sign-extended 16-bit immediate value to produce a memory location to write to in IO memory. The 32-bit value in $rt will then be stored into that memory location.

Restrictions:

Operation:

```
Fetch: PC <- PC +4; IR <- M[PC];
Decode: RS <- Reg[IR[25-21]]; RT <- IR[15:0]
OUTPUT: ALU_OUT <- RS($rs) + RT(imm)
OUTPUT2: ioM[ALU_OUT($rs+imm)] <- RT($rt)
```

Exceptions:

Programming Notes:

Example:

```
addi $3, $0, 03         # $3 <- 0x00000003
output $3, 0($16)       # ioM[$16] <- 0x00000003
```

RETI Return from Interrupt

31 26	25 21	20 16	15 11 10 6 5 0
0x1E	rs	rt	16 – bit immediate
6	5	5	16

Format: RETI

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

Description: PC <- dM[$sp]

Passes in the value at $sp and at the memory location pointed at to by the SP, POP's the top of stack twice to get the status flags and the return PC. It then updates an saves the new SP value into $sp.

Restrictions:

Operation:

Fetch: PC <- PC +4; IR <- M[PC];

Decode: RS <- Reg[IR[25-21]]; RT <- IR[15:0]

RETI: ALU_OUT <- RS($sp)

RET2I: D_in <- dMem[ALU_OUT($sp)]

RETI3: Flags <- D_in, ALU_OUT <- ALU_OUT($sp) + 4

RETI4: D_in <- dMem[ALU_OUT($sp+4)], ALU_OUT <- ALU_OUT($sp+4) + 4

RETI5: PC <- D_in(PC), ALU_OUT <- ALU_OUT($sp+8)

RETI6: RF[$sp] <- ALU_OUT($sp+8)

Exceptions:

Programming Notes:

Example:

```
@0
//interrupt here
//return here
@200
reti
```

DJNZ Decrement Jump Not Zero

31 26	25 21	20 16	15 11 10 6 5 0
0x30	rs	rt	16 – bit immediate
6	5	5	16

Format: DJNZ $rs, branch offset

Purpose: To branch to a location if the GPR specified isn’t zero.

Description: if(RS != 0); PC <- branch address

Decrements a 32-bit value from a GPR and branches to a branch address if that value isn’t zero. If the value is zero, nothing is done.

Restrictions: rs and rt must be the same value

Operation:

```
Fetch: PC <- PC +4; IR <- M[PC];
Decode: RS <- Reg[IR[25-21]]; RT <- IR[15:0]
DJNZ: ALU_OUT <- RS($rs) – 1
DJNZ2: $rt <- ALU_OUT($rs - 1)
if zero flag is set, PC <- branch address
else, pass through
```

Exceptions:

Programming Notes:

Example:

```
                addi $5, $0, 05   # $5 <- 0x00000005
                addi $13, $0, 05# $13 <- 0x00000005
//loop          addi $5, $5, 05   # $5 <- 0x00000005     # $5 <- 0x0000000A, F, 14, 19, 1E
                djnz $13, loop   # loops 5 times
                break
```

BLT — Branch Less Than

31 — 26 25	21 20	16 15	11 10 — 6 5 — 0
0x32	rs	rt	16 – bit immediate
6	5	5	16

Format: BLT rs, rt, branch offset

Purpose: To branch to a location if RS < RT

Description: if (RS < RT); PC <- branch address

$rs and $rt and subtracted and if a negative flag is set high, the branch address is calculated and branched to.

Restrictions:

Operation:

```
Fetch: PC <- PC +4; IR <- M[PC];
Decode: RS <- Reg[IR[25-21]]; RT <- Reg[IR[20-16]];
BLT: ALU_OUT <- RS($rs) - RT($rt)
BLT2: if(N == 1); PC <- branch address
```

Exceptions:

Programming Notes:

Example:

```
addi $3, $0, 03   # $3 <- 0x00000003
addi $5, $0, 05   # $5 <- 0x00000005
blt $3, $5, 3     # ($3 < $5), branches to branch address.
```

BGE Branch Greater or Equal

31 26	25 21	20 16	15 11 10 6 5 0
0x33	rs	rt	16 – bit immediate
6	5	5	16

Format: BGE rs, rt, branch offset

Purpose: To branch if RS >= RT

Description: if (RS >= RT); PC <- branch address

$rs and $rt are added and if a zero flag is high or if the negative flag is not high, then we will branch to a branch address calculated using the immediate offset.

Restrictions:

Operation:

Fetch: PC <- PC +4; IR <- M[PC];

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

BGE: ALU_OUT <- RS($rs) - RT($rt)

BGE2: if(Z || ~N); PC <- branch address

Exceptions:

Programming Notes:

Example:

```
addi $3, $0, 03   # $3 <- 0x00000003

addi $5, $0, 05   # $5 <- 0x00000005

bge $5, $3, 8     # ($5 > $3), branches to branch address.
```

PUSH

31 26	25 21	20 16	15 11	10 6	5 0
0x00	rs	rt	rd	shtamt	0x30
6	5	5	5	5	6

Format: PUSH rt

Purpose: To push a 32-bit value located in a GPR into the stack.

Description: dM[$sp] <- $rt

Pushes $rt into the stack. Pre-decrement, updates the $sp before pushing into stack.

Restrictions:

Operation:

```
Fetch: PC <- PC +4; IR <- M[PC];
Decode: RS <- Reg[IR[25-21]]; RT <- Reg[IR[20-16]];
PUSH: RS <- RF[$sp], RT <- $rt
PUSH2: ALU_OUT <- RS($sp) - 4
PUSH3: dM[ALU_OUT($sp-4)] <- RT($rt)
PUSH4: $sp <- ALU_OUT($sp)
```

Exceptions:

Programming Notes:

Example:

```
lui $1, 0x1234
ori $1, 0x5678   # $1 <- 0x12345678
push $1          # dM[3F8] <- 0x12345678
```

POP

31 26	25 21	20 16	15 11	10 6	5 0
0x00	rs	rt	rd	shtamt	0x31
6	5	5	5	5	6

Format: POP rt

Purpose: To pop a 32-bit value from top of stack into a GPR.

Description: $rt <- dM[$sp]

Pops the top of stack into a GPR specified by rt. Post-increment, updates the $sp after popping.

Restrictions:

Operation:

```
Fetch: PC <- PC +4; IR <- M[PC];

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

POP: RS <- RF[$sp]

POP2: ALU_OUT <- RS($sp)

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

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

POP5: $sp <- ALU_OUT
```

Exceptions:

Programming Notes:

Example:

```
lui $1, 0x1234

ori $1, 0x5678   # $1 <- 0x12345678

push $1          # dM[3F8] <- 0x12345678

pop $16          # $16 <- 0x12345678
```

NOP No Operation

31 26	25 21	20 16	15 11	10 6	5 0
0x00	rs	rt	rd	shtamt	0x32
6	5	5	5	5	6

Format: NOP

Purpose: To provide one clock tick where no operations are performed.

Description:

The value in the ALU_OUT register is passed back onto ALU_OUT so that the value isn't changed, while taking a clock tick to do it.

Restrictions:

Operation:

Fetch: PC <- PC +4; IR <- M[PC];

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

NOP: ALU_OUT <- ALU_OUT

Exceptions:

Programming Notes:

Example:

```
nop
nop
nop
```

CLR Clear

31 26	25 21	20 16	15 11	10 6	5 0
0x00	rs	rt	rd	shtamt	0x33
6	5	5	5	5	6

Format: CLR rt

Purpose: To clear a GPR.

Description: $rt <- 32'h0

Loads all zeros into a GPR.

Restrictions:

Operation:

Fetch: PC <- PC +4; IR <- M[PC];

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

CLR: ALU_OUT <- 0x0;

CLR2: $rt <- ALU_OUT(0x0)

Exceptions:

Programming Notes:

Example:

```
addi $6, $0, 16  # $6 <- 0x00000010
clr $6           # $6 <- 0x00000000
```

MOV Move

31 26	25 21	20 16	15 11	10 6	5 0
0x00	rs	rt	rd	shtamt	0x34
6	5	5	5	5	6

Format: MOV rs rt

Purpose: To move a 32-bit value from one register into another.

Description: $rt <- $rs

Moves $rs into $rt

Restrictions:

Operation:

Fetch: PC <- PC +4; IR <- M[PC];

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

MOV: ALU_OUT <- RS($rs)

MOV2: $rt <- ALU_OUT($rs)

Exceptions:

Programming Notes:

Example:

```
lui $4 0xFFFF
ori $4 0xFFFF          # $4 <- 0xFFFFFFFF
mov $10, $4            # $10 <- 0xFFFFFFFF
```

Verilog Implementation / Design / Verification

MIPS 32-bit Processor Top Level Module

```
`timescale 1ns / 1ps

/****************************** C E C S 4 4 0 ******************************
 *
 * File Name: MIPS_CU_TB.v
 * Project: Senior Design Project
 * Designer: Eduardo Marquez
 *                       Tuan Nguyen
 * Email: marquez.edu@outlook.com
 *                  tuannd.ryan2410@gmail.com
 * Rev. No.: Version 1.0
 * Rev. Date: 4/23/17
 *
 * Purpose: This module will structurally connect the MCU, Instruction_Unit,
 *                      Integer_Datapath, and Memory modules.
 *
 * Notes:
 *

****************************************************************************/

module MIPS_CU_TB;

      // Inputs
      reg sys_clk;
      reg reset;
      reg intr;
      reg [31:0] IR;

      // Outputs
      wire int_ack;
      wire [1:0] pc_sel;
      wire pc_ld;
      wire pc_inc;
      wire ir_ld;
      wire im_cs;
      wire im_rd;
      wire im_wr;
      wire D_En;
      wire [1:0] DA_sel;
      wire HILO_ld;
      wire [2:0] Y_sel;
      wire dm_cs;
      wire dm_rd;
      wire dm_wr;
      wire [4:0] FS;
      wire io_cs;
      wire io_rd;
      wire io_wr;
      wire S_sel;
      wire SP_sel;
      wire [1:0] T_sel;
      wire inI, inC, inV, inN, inZ;
      wire outI, outC, outV, outN, outZ;

      wire c;
      wire n;
      wire z;
      wire v;
      wire [31:0] IR_out,
                  pc_out,
                  se_16,
                  Addr,
                  d_in,
                  d_out;

//*****************************************************************************
// Instantiate MCU, Instruction Unit, Integer Datapath, and Memory
//*****************************************************************************

//    sys_clk, reset, intr, c, n, z, v, IR,      int_ack, pc_sel, pc_ld,
MCU mcu (sys_clk, reset, intr, c, n, z, v, IR_out, int_ack, pc_sel, pc_ld,

//    pc_inc, ir_ld, im_cs, im_rd, im_wr, D_En, DA_sel, T_sel, S_sel,
      pc_inc, ir_ld, im_cs, im_rd, im_wr, D_En, DA_sel, T_sel, S_sel,

//    HILO_ld, Y_sel, dm_cs, dm_rd, dm_wr, FS, io_cs, io_rd, io_wr, SP_sel,
      HILO_ld, Y_sel, dm_cs, dm_rd, dm_wr, FS, io_cs, io_rd, io_wr, SP_sel,

//          input from IDP          output to IDP
      inI, inC, inV, inN, inZ, outI, outC, outV, outN, outZ);

//                clk, reset, pc_sel, pc_ld, pc_inc, im_cs, im_wr, im_rd,
Instruction_Unit iu (sys_clk, reset, pc_sel, pc_ld, pc_inc, im_cs, im_wr,
im_rd,

//                ir_ld, pc_in, pc_out, ir_out, se_16);
                  ir_ld, Addr,  pc_out, IR_out, se_16);

//                      clk, reset, D_En, DA_sel,     D_Addr,
Integer_Datapath id (sys_clk, reset, D_En, DA_sel, IR_out[15:11],
//                      S_Addr,
                        IR_out[25:21],

//          T_Addr,          shamt,            DT,     T_Sel, FS, C, N, Z, V,
      IR_out[20:16], IR_out[10:6], se_16, T_sel, FS, c, n, z, v,

//    HILO_ld, DY,     PC_in,   Y_Sel, ALU_OUT, D_OUT, S_sel, SP_sel
      HILO_ld, d_out, pc_out, Y_sel, Addr,    d_in, S_sel, SP_sel,

//          input from MCU          output to MCU
      outI, outC, outV, outN, outZ, inI, inC, inV, inN, inZ);
```

```
//                  clk, dm_cs, dm_wr, dm_rd, io_cs, io_wr, io_rd,
      Memory Mem (sys_clk, dm_cs, dm_wr, dm_rd, io_cs, io_wr, io_rd,

//                              Addr,          D_In, D_Out
                        {20'h0,Addr[11:0]}, d_in, d_out);

// Instantiate the Unit Under Test (UUT)
      MCU uut (
            .sys_clk(sys_clk),
            .reset(reset),
            .intr(intr),
            .c(c),
            .n(n),
            .z(z),
            .v(v),
            .IR(IR_out),
            .int_ack(int_ack),
            .pc_sel(pc_sel),
            .pc_ld(pc_ld),
            .pc_inc(pc_inc),
            .ir_ld(ir_ld),
            .im_cs(im_cs),
            .im_rd(im_rd),
            .im_wr(im_wr),
            .D_En(D_En),
            .DA_sel(DA_sel),
            .T_sel(T_sel),
            .HILO_ld(HILO_ld),
            .Y_sel(Y_sel),
            .dm_cs(dm_cs),
            .dm_rd(dm_rd),
            .dm_wr(dm_wr),
            .FS(FS)
      );

      always #5 sys_clk = ~sys_clk;

      initial begin
            $timeformat(-9, 1, " ns", 9);
            sys_clk = 0;
            @(negedge sys_clk)
            reset = 1;
            @(negedge sys_clk)
            reset = 0;

            //*******************************************************
            // Initialize Instruction Memory
            //*******************************************************
            @(negedge sys_clk)
            $readmemh("iMem01_Sp17.dat",iu.iMem.irMem);
//          $readmemh("iMem02_Sp17.dat",iu.iMem.irMem);
//          $readmemh("iMem03_Sp17.dat",iu.iMem.irMem);
//          $readmemh("iMem04_Sp17.dat",iu.iMem.irMem);
//          $readmemh("iMem05_Sp17.dat",iu.iMem.irMem);
//          $readmemh("iMem06_Sp17.dat",iu.iMem.irMem);
//          $readmemh("iMem07_Sp17.dat",iu.iMem.irMem);
//          $readmemh("iMem08_Sp17.dat",iu.iMem.irMem);
//          $readmemh("iMem09_Sp17.dat",iu.iMem.irMem);
//          $readmemh("iMem10_Sp17.dat",iu.iMem.irMem);
//          $readmemh("iMem11_Sp17.dat",iu.iMem.irMem);
//          $readmemh("iMem12_Sp17.dat",iu.iMem.irMem);
//          $readmemh("iMem13_Sp17_w_isr.dat",iu.iMem.irMem);
//          $readmemh("iMem14_Sp17_w_isr.dat",iu.iMem.irMem);
            $readmemh("iMemEnhanced_Sp17.dat",iu.iMem.irMem);

            //*****************************************************
            // Initialize Data Memory
            //*****************************************************
            @(negedge sys_clk)
//          $readmemh("dMem01_Sp17.dat",Mem.dM);
//          $readmemh("dMem02_Sp17.dat",Mem.dM);
//          $readmemh("dMem03_Sp17.dat",Mem.dM);
//          $readmemh("dMem04_Sp17.dat",Mem.dM);
//          $readmemh("dMem05_Sp17.dat",Mem.dM);
//          $readmemh("dMem06_Sp17.dat",Mem.dM);
//          $readmemh("dMem07_Sp17.dat",Mem.dM);
//          $readmemh("dMem08_Sp17.dat",Mem.dM);
//          $readmemh("dMem09_Sp17.dat",Mem.dM);
//          $readmemh("dMem10_Sp17.dat",Mem.dM);
//          $readmemh("dMem11_Sp17.dat",Mem.dM);
//          $readmemh("dMem12_Sp17.dat",Mem.dM);
//          $readmemh("dMem13_Sp17.dat",Mem.dM);
//          $readmemh("dMem14_Sp17.dat",Mem.dM);
            $readmemh("dMemEnhanced_Sp17.dat",Mem.dM);


            //#400 //about 10 instructions
            #680 //about 17 instructions
            intr = 1;
            #100
            intr = 0;

            //$stop;
      end

endmodule
```

MIPS Control Unit

```
`timescale 1ns / 1ps
/****************************** C E C S 4 4 0 ******************************
 *
 * File Name: MCU.v
 * Project: Senior Design Project
 * Designer: Eduardo Marquez
 *                         Tuan Nguyen
 * Email: marquez.edu@outlook.com
 *                   tuannd.ryan2410@gmail.com
 * Rev. No.: Version 1.0
 * Rev. Date: 4/24/17
 *
 * Purpose: A state machine implementing the MIPS Control Unit (MCU) for the
 *    major cycles of fetch, execute and some MIPS instructions from memory,
 *    including checking for interrupts.
 *
 * Notes:
 *

****************************************************************************/
/****************************************************************************
------------------------------------------------------------------------
* MCU C O N T R O L W O R D
*----------------------------------------------------------------------------
*
*     int_ack=0;                    FS=5'h0;
*     {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
*     {im_cs, im_rd, im_wr} = 3'b0_0_0;
*     {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
*     11'b0_00_0_00_0_0_000;
*     {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
*     {io_cs, io_rd, io_wr} = 3'b0_0_0;
*     {nsi, nsc, nsv, nsn, nsz} = 5'b0;
*
****************************************************************************/

//***************************************************************************
module MCU (sys_clk, reset, intr,         // system inputs
            c, n, z, v,                   // ALU status inputs
            IR,                           // Instruction Register input
            int_ack,                      // output to I/O subsystem
            pc_sel, pc_ld, pc_inc, ir_ld, // rest of control word fields
            im_cs, im_rd, im_wr,
            D_En, DA_sel, T_sel, S_sel, HILO_ld, Y_sel,
            dm_cs, dm_rd, dm_wr,
            FS, io_cs, io_rd, io_wr, SP_sel,
            inI, inC, inV, inN, inZ,
            nsi, nsc, nsv, nsn, nsz);
//***************************************************************************

input       sys_clk, reset, intr;   //system clock, reset, interrupt request
input       c, n, z, v;             // Integer ALU status inputs
input       inI, inC, inV, inN, inZ;// flags from idp
input [31:0] IR;                    // Instruction Register input from IU
output      int_ack;                // interrupt acknowledge
output      pc_ld, pc_inc, ir_ld;   // ALL OF THE REMAINING
output      im_cs, im_rd, im_wr;    // CONTROL WORD OUTPUTS
output      D_En, HILO_ld;          // for the IU, DP and Data Memory
output      dm_cs, dm_rd, dm_wr;
output [1:0] pc_sel;
output [1:0] DA_sel;
output [1:0] T_sel;
output [2:0] Y_sel;
output [4:0] FS;
output      io_cs, io_rd, io_wr;
output       S_sel;
output      SP_sel;
output      nsi, nsc, nsv, nsn, nsz; //output to IDP
reg         int_ack;                // interrupt acknowledge
reg         pc_ld, pc_inc, ir_ld;   // to contain all of the bits
reg         im_cs, im_rd, im_wr;    // for all the control word fields
reg         D_En, HILO_ld;          // needed by the IU, DP and Data Memory
reg         dm_cs, dm_rd, dm_wr;
reg  [1:0]   pc_sel;
reg  [1:0]   DA_sel;
reg  [1:0]   T_sel;
reg  [2:0]   Y_sel;
reg  [4:0]   FS;
reg         io_cs, io_rd, io_wr;
reg         psi, psc, psv, psn, psz;      //flag registers
reg         nsi, nsc, nsv, nsn, nsz;      //flag registers
reg         S_sel;
reg         SP_sel;

// Declare variables
integer i, X, Y;

//****************************
// internal data structures
//****************************

// state assignments
parameter
RESET        = 00,  FETCH     = 01,  DECODE = 02,
//R-TYPE
SLL = 10, SRL = 11, SRA  = 12, JR   = 13, MFHI  = 14, MFLO  = 15, MULT = 16,
DIV = 17, ADD = 18, ADDU = 19, SUB  = 20, SUBU  = 21, AND   = 22, OR   = 23,
XOR = 24, NOR = 25, SLT  = 26, SLTU = 27, SETIE = 29,
JR2 = 40,

//I-TYPE
BEQ  = 50, BNE = 51, BLEZ = 52, BGTZ = 53, ADDI = 54, SLTI = 55, SLTIU = 56,
ANDI = 57, ORI = 58, XORI = 59, LUI  = 60, LW   = 61, SW   = 62,
BEQ2 = 70, BNE2 = 71,
LW2 = 80, LW3 = 81, BLEZ2 = 82, BGTZ2 = 83,

//J-TYPE
J = 90, JAL = 91,

//Enhanced shift operations
SLA = 100, ROL = 101, ROR = 102,

// START AT FOR MORE 130 ENHANCED INSTRUCTIONS
INPUT = 130, OUTPUT = 131, RETI = 132, E_KEY = 133,
INPUT2 = 140, INPUT3 = 141, OUTPUT2 = 142, RETI2 = 143, RETI3 = 144,
RETI4 = 145, RETI5 = 146, RETI6 = 147,

// ENHANCEMENTS, HAVE A VARIETY OF INSTRUCTION TYPES
PUSH = 170, PUSH2 = 171, PUSH3 = 172, POP = 173, POP2 = 174,
POP3 = 175, POP4 = 176, DJNZ = 177, DJNZ2 = 178, NOP = 180,
CLR = 181, CLR2 = 182, MOV = 183, MOV2 = 184, BLT = 185, BLT2 = 186,
BGE = 187, BGE2 = 188, POP5 = 189, PUSH4 = 190,

WB_alu       = 30, WB_imm = 31,  WB_Din = 32,   WB_hi = 33,  WB_lo = 34,
WB_mem       = 35, INTR_1 = 501, INTR_2 = 502, INTR_3 = 503, INTR_4 = 504,
INTR_5 = 505, INTR_6 = 506,
BREAK        = 510,
ILLEGAL_OP= 511;
//state register (up to 512 states)
reg [8:0] state;

/**************************************************
* 440 MIPS CONTROL UNIT (Finite State Machine) *
**************************************************/

// AND gate, only interrupts if the ie flag and intr are both high
assign intrpt = (nsi && intr) ? 1'b1 : 1'b0;

// Flags Register
always@(posedge sys_clk, posedge reset)
      if (reset)
            {psi, psc, psv, psn, psz} = 5'b0;
      else
            {psi, psc, psv, psn, psz} = {nsi, nsc, nsv, nsn, nsz};

always @(posedge sys_clk, posedge reset)
      if (reset)
      begin
      // control word assignments for ALU_OUT <-- 32'h3FC
            @(negedge sys_clk)
            int_ack=0;                    FS=5'h15;
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0;
            {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
            11'b0_00_0_00_0_0_000;
            {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
            {io_cs, io_rd, io_wr} = 3'b0_0_0;
            {nsi, nsc, nsv, nsn, nsz} = 5'b0;
            state = RESET;
            end
        else
            case (state)
            FETCH:
            if (int_ack==0 & intrpt==1)
            //if (int_ack==0 & intr==1)
            begin //*** new interrupt pending; prepare for ISR ***
            // control word assignments for "deasserting" everything
                  @(negedge sys_clk)
                  int_ack=0;                    FS=5'h0;
                  {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                  {im_cs, im_rd, im_wr} = 3'b0_0_0;
                  {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                  11'b0_00_0_00_0_0_000;
                  {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                  {io_cs, io_rd, io_wr} = 3'b0_0_0;
                  {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                  state = INTR_1;
            end
            else
            begin //*** no new interrupt pending; fetch and instruction
                  if (int_ack==1 & intrpt==0) int_ack=1'b0;
                  //if (int_ack==1 & intr==0) int_ack=1'b0;
                  // control word assignments for IR <- iM[PC]; PC <- PC+4
                  @(negedge sys_clk)
                  int_ack=0;                      FS=5'h0;
                  {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_1_1;
                  {im_cs, im_rd, im_wr} = 3'b1_1_0;
                  {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                  11'b0_00_0_00_0_0_000;
                  {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                  {io_cs, io_rd, io_wr} = 3'b0_0_0;
                  {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                  state = DECODE;
            end

RESET:
            begin
            // control word assignments for $sp <-- ALU_Out(32'h3FC)
                  @(negedge sys_clk)
                  int_ack=0;                      FS=5'h0;
                  {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                  {im_cs, im_rd, im_wr} = 3'b0_0_0;
                  {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                  11'b1_11_0_00_0_0_000;
                  {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                  {io_cs, io_rd, io_wr} = 3'b0_0_0;
                  {nsi, nsc, nsv, nsn, nsz} = 5'b0;
                  state = FETCH;
            end

DECODE:
            begin
                  @(negedge sys_clk) // check for MIPS format
                  if (MIPS_CU_TB.iu.ir_out[31:26] == 6'h0)
                  begin
            // it is an R-type format
            // control word assignments: RS <-- $rs RT <-- $rt (default)
                  int_ack=0;                      FS=5'h0;
                  {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                  {im_cs, im_rd, im_wr} = 3'b0_0_0;
                  {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                  11'b0_00_0_00_0_0_000;
                  {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                  {io_cs, io_rd, io_wr} = 3'b0_0_0;
                  {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                  case (MIPS_CU_TB.iu.ir_out[5:0])
                        6'h00 :  state = SLL;
                        6'h02 :  state = SRL;
                        6'h03 :  state = SRA;
                        6'h35 :  state = SLA;
                        6'h36 :  state = ROL;
                        6'h37 :  state = ROR;
                        6'h08 :  state = JR;
                        6'h10 :  state = MFHI;
                        6'h12 :  state = MFLO;
                        6'h18 :  state = MULT;
                        6'h1A :  state = DIV;
                        6'h20 :  state = ADD;
                        6'h21 :  state = ADDU;
                        6'h22 :  state = SUB;
                        6'h23 :  state = SUBU;
```

```
                6'h24 :  state = AND;
                6'h25 :  state = OR;
                6'h26 :  state = XOR;
                6'h27 :  state = NOR;
                6'h2A :  state = SLT;
                6'h2B :  state = SLTU;
                6'h0D :  state = BREAK;
                6'h1F :  state = SETIE;
                6'h30 :  state = PUSH;
                6'h31 :  state = POP;
                6'h32 :  state = NOP;
                6'h33 :  state = CLR;
                6'h34 :  state = MOV;
                default: state = ILLEGAL_OP;
          endcase
    end // end of if for R-type Format
    else
    begin // it is an I-type or J-type format
          // control word assignments: RS <-- $rs RT <-- DT(se_16)
    if(MIPS_CU_TB.iu.ir_out[31:26] != 6'h04 ||
    MIPS_CU_TB.iu.ir_out[31:26] != 6'h05 ||
    MIPS_CU_TB.iu.ir_out[31:26] != 6'h06 ||
    MIPS_CU_TB.iu.ir_out[31:26] != 6'h07 ||
    MIPS_CU_TB.iu.ir_out[31:26] != 6'h32 ||
    MIPS_CU_TB.iu.ir_out[31:26] != 6'h33) begin
    int_ack=0;                    FS=5'h0;
    {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
    {im_cs, im_rd, im_wr} = 3'b0_0_0;
    {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
    11'b0_00_0_01_0_0_000;
    {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
    {io_cs, io_rd, io_wr} = 3'b0_0_0;
    {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz}; end
    else
    begin // for branch instruction, RS <-- $rs RT <-- $rt
    int_ack=0;                    FS=5'h0;
    {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
    {im_cs, im_rd, im_wr} = 3'b0_0_0;
    {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
    11'b0_00_0_00_0_0_000;
    {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
    {io_cs, io_rd, io_wr} = 3'b0_0_0;
    {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz}; end
    case (MIPS_CU_TB.iu.ir_out[31:26])
                6'h02 :  state = J;
                6'h03 :  state = JAL;
                6'h04 :  state = BEQ;
                6'h05 :  state = BNE;
                6'h06 :  state = BLEZ;
                6'h07 :  state = BGTZ;
                6'h08 :  state = ADDI;
                6'h0A :  state = SLTI;
                6'h0B :  state = SLTIU;
                6'h0C :  state = ANDI;
                6'h0D :  state = ORI;
                6'h0E :  state = XORI;
                6'h0F :  state = LUI;
```

```
                        6'h23 :  state = LW;
                        6'h2B :  state = SW;
                        6'h1C :  state = INPUT;
                        6'h1D :  state = OUTPUT;
                        6'h1E :  state = RETI;
                        6'h1F :  state = E_KEY;
                        6'h30 :  state = DJNZ;
                        6'h32 :  state = BLT;
                        6'h33 :  state = BGE;
                        default: state = ILLEGAL_OP;
                endcase
                end // end of else for I-type or J-type formats
                end // end of DECODE


/********************R-TYPE********************/

SLL:
                begin // control word assignments:
                      @(negedge sys_clk)
                      int_ack=0;                    FS=5'h0C;
                      {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                      {im_cs, im_rd, im_wr} = 3'b0_0_0;
                      {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                      11'b0_00_0_00_0_0_000;
                      {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                      {io_cs, io_rd, io_wr} = 3'b0_0_0;
                      {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                      state = WB_alu; end

SRL:
                begin // control word assignments:
                      @(negedge sys_clk)
                      int_ack=0;                    FS=5'h0D;
                      {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                      {im_cs, im_rd, im_wr} = 3'b0_0_0;
                      {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                      11'b0_00_0_00_0_0_000;
                      {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                      {io_cs, io_rd, io_wr} = 3'b0_0_0;
                      {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                      state = WB_alu; end

SRA:
                begin // control word assignments:
                      @(negedge sys_clk)
                      int_ack=0;                    FS=5'h0E;
                      {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                      {im_cs, im_rd, im_wr} = 3'b0_0_0;
                      {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                      11'b0_00_0_00_0_0_000;
                      {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                      {io_cs, io_rd, io_wr} = 3'b0_0_0;
                      {nsi, nsc, nsv, nsn, nsz} = {psi, c, v, n, z};
                      state = WB_alu; end

SLA:
```

```
            begin// control word assignments:
                  @(negedge sys_clk)
                  int_ack=0;                        FS=5'h1A;
                  {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                  {im_cs, im_rd, im_wr} = 3'b0_0_0;
                  {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                  11'b0_00_0_00_0_0_000;
                  {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                  {io_cs, io_rd, io_wr} = 3'b0_0_0;
                  {nsi, nsc, nsv, nsn, nsz} = {psi, c, v, n, z};
                  state = WB_alu; end

ROL:
            begin// control word assignments:
                  @(negedge sys_clk)
                  int_ack=0;                        FS=5'h1B;
                  {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                  {im_cs, im_rd, im_wr} = 3'b0_0_0;
                  {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                  11'b0_00_0_00_0_0_000;
                  {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                  {io_cs, io_rd, io_wr} = 3'b0_0_0;
                  {nsi, nsc, nsv, nsn, nsz} = {psi, c, v, n, z};
                  state = WB_alu; end

ROR:
            begin// control word assignments:
                  @(negedge sys_clk)
                  int_ack=0;                        FS=5'h1C;
                  {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                  {im_cs, im_rd, im_wr} = 3'b0_0_0;
                  {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                  11'b0_00_0_00_0_0_000;
                  {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                  {io_cs, io_rd, io_wr} = 3'b0_0_0;
                  {nsi, nsc, nsv, nsn, nsz} = {psi, c, v, n, z};
                  state = WB_alu; end

JR:
            begin// control word assignments:
                  @(negedge sys_clk)
                  int_ack=0;                        FS=5'h0;
                  {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                  {im_cs, im_rd, im_wr} = 3'b0_0_0;
                  {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                  11'b0_00_0_00_0_0_000;
                  {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                  {io_cs, io_rd, io_wr} = 3'b0_0_0;
                  {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                  state = JR2; end

JR2:
            begin// control word assignments:
                  @(negedge sys_clk)
                  int_ack=0;                        FS=5'h0;
                  {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_1_0_0;
                  {im_cs, im_rd, im_wr} = 3'b0_0_0;
```

```
                {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                11'b0_00_0_00_0_0_000;
                {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                {io_cs, io_rd, io_wr} = 3'b0_0_0;
                {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                state = FETCH; end

MFHI:
          begin// control word assignments:
                @(negedge sys_clk)
                int_ack=0;                     FS=5'h0;
                {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                {im_cs, im_rd, im_wr} = 3'b0_0_0;
                {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                11'b1_00_0_00_0_0_011;
                {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                {io_cs, io_rd, io_wr} = 3'b0_0_0;
                {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                state = FETCH; end

MFLO:
          begin// control word assignments:
                @(negedge sys_clk)
                int_ack=0;                     FS=5'h0;
                {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                {im_cs, im_rd, im_wr} = 3'b0_0_0;
                {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                11'b1_00_0_00_0_0_100;
                {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                {io_cs, io_rd, io_wr} = 3'b0_0_0;
                {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                state = FETCH; end

MULT:
          begin// control word assignments:
                @(negedge sys_clk)
                int_ack=0;                     FS=5'h1E;
                {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                {im_cs, im_rd, im_wr} = 3'b0_0_0;
                {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                11'b0_00_0_00_0_1_000;
                {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                {io_cs, io_rd, io_wr} = 3'b0_0_0;
                {nsi, nsc, nsv, nsn, nsz} = {psi, c, v, n, z};
                state = FETCH; end

DIV:
          begin// control word assignments:
                @(negedge sys_clk)
                int_ack=0;                     FS=5'h1F;
                {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                {im_cs, im_rd, im_wr} = 3'b0_0_0;
                {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                11'b0_00_0_00_0_1_000;
                {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                {io_cs, io_rd, io_wr} = 3'b0_0_0;
                {nsi, nsc, nsv, nsn, nsz} = {psi, c, v, n, z};
```

```
                    state = WB_alu; end

ADD:
            begin// control word assignments: ALU_Out <-- $rs + $rt
                    @(negedge sys_clk)
                    int_ack=0;                    FS=5'h02;
                    {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                    {im_cs, im_rd, im_wr} = 3'b0_0_0;
                    {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                    11'b0_00_0_00_0_0_000;
                    {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                    {io_cs, io_rd, io_wr} = 3'b0_0_0;
                    {nsi, nsc, nsv, nsn, nsz} = {psi, c, v, n, z};
                    state = WB_alu; end

ADDU:
            begin// control word assignments:
                    @(negedge sys_clk)
                    int_ack=0;                    FS=5'h03;
                    {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                    {im_cs, im_rd, im_wr} = 3'b0_0_0;
                    {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                    11'b0_00_0_00_0_0_000;
                    {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                    {io_cs, io_rd, io_wr} = 3'b0_0_0;
                    {nsi, nsc, nsv, nsn, nsz} = {psi, c, v, n, z};
                    state = WB_alu; end

SUB:
            begin// control word assignments:
                    @(negedge sys_clk)
                    int_ack=0;                    FS=5'h04;
                    {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                    {im_cs, im_rd, im_wr} = 3'b0_0_0;
                    {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                    11'b0_00_0_00_0_0_000;
                    {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                    {io_cs, io_rd, io_wr} = 3'b0_0_0;
                    {nsi, nsc, nsv, nsn, nsz} = {psi, c, v, n, z};
                    state = WB_alu; end

SUBU:
            begin // control word assignments:
                    @(negedge sys_clk)
                    int_ack=0;                    FS=5'h05;
                    {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                    {im_cs, im_rd, im_wr} = 3'b0_0_0;
                    {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                    11'b0_00_0_00_0_0_000;
                    {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                    {io_cs, io_rd, io_wr} = 3'b0_0_0;
                    {nsi, nsc, nsv, nsn, nsz} = {psi, c, v, n, z};
                    state = WB_alu; end

AND:
            begin// control word assignments:
                    @(negedge sys_clk)
```

```
                int_ack=0;                    FS=5'h08;
                {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                {im_cs, im_rd, im_wr} = 3'b0_0_0;
                {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                11'b0_00_0_00_0_0_000;
                {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                {io_cs, io_rd, io_wr} = 3'b0_0_0;
                {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                state = WB_alu; end

OR:
            begin// control word assignments:
                @(negedge sys_clk)
                int_ack=0;                    FS=5'h09;
                {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                {im_cs, im_rd, im_wr} = 3'b0_0_0;
                {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                11'b0_00_0_00_0_0_000;
                {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                {io_cs, io_rd, io_wr} = 3'b0_0_0;
                {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                state = WB_alu; end

XOR:
            begin// control word assignments:
                @(negedge sys_clk)
                int_ack=0;                    FS=5'h0A;
                {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                {im_cs, im_rd, im_wr} = 3'b0_0_0;
                {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                11'b0_00_0_00_0_0_000;
                {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                {io_cs, io_rd, io_wr} = 3'b0_0_0;
                {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                state = WB_alu; end

NOR:
            begin// control word assignments:
                @(negedge sys_clk)
                int_ack=0;                    FS=5'h0B;
                {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                {im_cs, im_rd, im_wr} = 3'b0_0_0;
                {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                11'b0_00_0_00_0_0_000;
                {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                {io_cs, io_rd, io_wr} = 3'b0_0_0;
                {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                state = WB_alu; end

SLT:
            begin// control word assignments:
                @(negedge sys_clk)
                int_ack=0;                    FS=5'h06;
                {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                {im_cs, im_rd, im_wr} = 3'b0_0_0;
                {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                11'b0_00_0_00_0_0_000;
```

```
                  {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                  {io_cs, io_rd, io_wr} = 3'b0_0_0;
                  {nsi, nsc, nsv, nsn, nsz} = {psi, c, v, n, z};
                  state = WB_alu; end

SLTU:
            begin// control word assignments:
                  @(negedge sys_clk)
                  int_ack=0;                    FS=5'h07;
                  {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                  {im_cs, im_rd, im_wr} = 3'b0_0_0;
                  {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                  11'b0_00_0_00_0_0_000;
                  {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                  {io_cs, io_rd, io_wr} = 3'b0_0_0;
                  {nsi, nsc, nsv, nsn, nsz} = {psi, c, v, n, z};
                  state = WB_alu; end

//BREAK

SETIE:
            begin// control word assignments:
                  @(negedge sys_clk)
                  int_ack=0;                    FS=5'h0;
                  {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                  {im_cs, im_rd, im_wr} = 3'b0_0_0;
                  {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                  11'b0_00_0_00_0_0_000;
                  {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                  {io_cs, io_rd, io_wr} = 3'b0_0_0;
                  {nsi, nsc, nsv, nsn, nsz} = {1'b1, psc, psv, psn, psz};
                  state = FETCH; end


/********************I-TYPE********************/

BEQ:
            begin// control word assignments: ALU_Out <-- $rs - $rt
                  @(negedge sys_clk)
                  int_ack=0;                    FS=5'h04;
                  {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b10_0_0_0;
                  {im_cs, im_rd, im_wr} = 3'b0_0_0;
                  {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                  11'b0_00_0_00_0_0_000;
                  {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                  {io_cs, io_rd, io_wr} = 3'b0_0_0;
                  {nsi, nsc, nsv, nsn, nsz} = {psi, c, v, n, z};
                  state = BEQ2; end

BEQ2:
            begin// control word assignments:
                  @(negedge sys_clk)
                  int_ack=0;                    FS=5'h0;
                  {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b10_0_0_0;
                  {im_cs, im_rd, im_wr} = 3'b0_0_0;
                  {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                  11'b0_00_0_00_0_0_000;
                  {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                  {io_cs, io_rd, io_wr} = 3'b0_0_0;
                  {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                  pc_ld = z;
                  state = FETCH; end


BNE:
            begin// control word assignments:
                  @(negedge sys_clk)
                  int_ack=0;                     FS=5'h04;
                  {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b10_0_0_0;
                  {im_cs, im_rd, im_wr} = 3'b0_0_0;
                  {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                  11'b0_00_0_00_0_0_000;
                  {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                  {io_cs, io_rd, io_wr} = 3'b0_0_0;
                  {nsi, nsc, nsv, nsn, nsz} = {psi, c, v, n, z};
                  state = BNE2; end

BNE2:
            begin// control word assignments:
                  @(negedge sys_clk)
                  int_ack=0;                     FS=5'h0;
                  {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b10_0_0_0;
                  {im_cs, im_rd, im_wr} = 3'b0_0_0;
                  {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                  11'b0_00_0_00_0_0_000;
                  {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                  {io_cs, io_rd, io_wr} = 3'b0_0_0;
                  {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                  pc_ld = !z;
                  state = FETCH;end

BLEZ:
            begin// control word assignments: R[rs] <= 0
                  @(negedge sys_clk)
                  int_ack=0;                     FS=5'h0;
                  {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b10_0_0_0;
                  {im_cs, im_rd, im_wr} = 3'b0_0_0;
                  {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                  11'b0_00_0_00_0_0_000;
                  {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                  {io_cs, io_rd, io_wr} = 3'b0_0_0;
                  {nsi, nsc, nsv, nsn, nsz} = {psi, c, v, n, z};
                  state = BLEZ2; end

BLEZ2:
            begin// control word assignments:
                  @(negedge sys_clk)
                  int_ack=0;                     FS=5'h0;
                  {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b10_0_0_0;
                  {im_cs, im_rd, im_wr} = 3'b0_0_0;
                  {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                  11'b0_00_0_00_0_0_000;
                  {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                  {io_cs, io_rd, io_wr} = 3'b0_0_0;
```

```
                    {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                    pc_ld = (n || z);
                    state = FETCH; end

BGTZ:
              begin// control word assignments:
                    @(negedge sys_clk)
                    int_ack=0;                    FS=5'h0;
                    {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b10_0_0_0;
                    {im_cs, im_rd, im_wr} = 3'b0_0_0;
                    {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                    11'b0_00_0_00_0_0_000;
                    {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                    {io_cs, io_rd, io_wr} = 3'b0_0_0;
                    {nsi, nsc, nsv, nsn, nsz} = {psi, c, v, n, z};
                    state = BGTZ2; end

BGTZ2:
              begin// control word assignments:
                    @(negedge sys_clk)
                    int_ack=0;                    FS=5'h0;
                    {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b10_0_0_0;
                    {im_cs, im_rd, im_wr} = 3'b0_0_0;
                    {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                    11'b0_00_0_00_0_0_000;
                    {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                    {io_cs, io_rd, io_wr} = 3'b0_0_0;
                    {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                    pc_ld = (~n && ~z);
                    state = FETCH; end

ADDI:
              begin// control word assignments:
                    @(negedge sys_clk)
                    int_ack=0;                    FS=5'h02;
                    {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                    {im_cs, im_rd, im_wr} = 3'b0_0_0;
                    {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                    11'b0_00_0_01_0_0_000;
                    {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                    {io_cs, io_rd, io_wr} = 3'b0_0_0;
                    {nsi, nsc, nsv, nsn, nsz} = {psi, c, v, n, z};
                    state = WB_imm; end

SLTI:
              begin// control word assignments:
                    @(negedge sys_clk)
                    int_ack=0;                    FS=5'h06;
                    {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                    {im_cs, im_rd, im_wr} = 3'b0_0_0;
                    {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                    11'b0_00_0_01_0_0_000;
                    {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                    {io_cs, io_rd, io_wr} = 3'b0_0_0;
                    {nsi, nsc, nsv, nsn, nsz} = {psi, c, v, n, z};
                    state = WB_imm; end
```

```
SLTIU:
            begin// control word assignments:
                  @(negedge sys_clk)
                  int_ack=0;                    FS=5'h07;
                  {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                  {im_cs, im_rd, im_wr} = 3'b0_0_0;
                  {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                  11'b0_00_0_01_0_0_000;
                  {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                  {io_cs, io_rd, io_wr} = 3'b0_0_0;
                  {nsi, nsc, nsv, nsn, nsz} = {psi, c, v, n, z};
                  state = WB_imm; end

ANDI:
            begin// control word assignments:
                  @(negedge sys_clk)
                  int_ack=0;                    FS=5'h16;
                  {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                  {im_cs, im_rd, im_wr} = 3'b0_0_0;
                  {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                  11'b0_00_0_01_0_0_000;
                  {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                  {io_cs, io_rd, io_wr} = 3'b0_0_0;
                  {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                  state = WB_imm; end

ORI:
            begin
      // ctrl word assignments for ALU_Out <-- $rs | {16'h0, RT[15:0]}
                  @(negedge sys_clk)
                  int_ack=0;                    FS=5'h17;
                  {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                  {im_cs, im_rd, im_wr} = 3'b0_0_0;
                  {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                  11'b0_00_0_01_0_0_000;
                  {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                  {io_cs, io_rd, io_wr} = 3'b0_0_0;
                  {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                  state = WB_imm; end

XORI:
            begin// control word assignments:
                  @(negedge sys_clk)
                  int_ack=0;                    FS=5'h18;
                  {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                  {im_cs, im_rd, im_wr} = 3'b0_0_0;
                  {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                  11'b0_00_0_01_0_0_000;
                  {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                  {io_cs, io_rd, io_wr} = 3'b0_0_0;
                  {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                  state = WB_imm; end

LUI:
            begin
      // control word assignments for ALU_Out <-- { RT[15:0], 16'h0}
                  @(negedge sys_clk)
```

```
                        int_ack=0;                    FS=5'h19;
                        {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                        {im_cs, im_rd, im_wr} = 3'b0_0_0;
                        {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                        11'b0_00_0_01_0_0_000;
                        {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                        {io_cs, io_rd, io_wr} = 3'b0_0_0;
                        {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                        state = WB_imm; end

LW:
                  begin// control word assignments:
                        @(negedge sys_clk)
                        int_ack=0;                    FS=5'h02;
                        {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                        {im_cs, im_rd, im_wr} = 3'b0_0_0;
                        {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                        11'b0_00_0_01_0_0_000;
                        {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                        {io_cs, io_rd, io_wr} = 3'b0_0_0;
                        {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                        state = LW2; end

LW2:
                  begin// control word assignments:
                        @(negedge sys_clk)
                        int_ack=0;                    FS=5'h0;
                        {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                        {im_cs, im_rd, im_wr} = 3'b0_0_0;
                        {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                        11'b0_00_0_00_0_0_000;
                        {dm_cs, dm_rd, dm_wr} = 3'b1_1_0;
                        {io_cs, io_rd, io_wr} = 3'b0_0_0;
                        {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                        state = LW3; end

LW3:
                  begin// control word assignments:
                        @(negedge sys_clk)
                        int_ack=0;                    FS=5'h0;
                        {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                        {im_cs, im_rd, im_wr} = 3'b0_0_0;
                        {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                        11'b1_01_0_00_0_0_001;
                        {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                        {io_cs, io_rd, io_wr} = 3'b0_0_0;
                        {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                        state = FETCH; end

SW:
                  begin
            // control word assignments for ALU_Out <-- $rs + $rt(se_16) "EA calc"
                        @(negedge sys_clk)
                        int_ack=0;                    FS=5'h02;
                        {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                        {im_cs, im_rd, im_wr} = 3'b0_0_0;
                        {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                        11'b0_01_0_00_0_0_000;
                        {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                        {io_cs, io_rd, io_wr} = 3'b0_0_0;
                        {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                        state = WB_mem; end


      /*******************J-TYPE*******************/

      J:
                  begin// control word assignments:
                        @(negedge sys_clk)
                        int_ack=0;                    FS=5'h0;
                        {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b01_1_0_0;
                        {im_cs, im_rd, im_wr} = 3'b0_0_0;
                        {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                        11'b0_00_0_00_0_0_000;
                        {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                        {io_cs, io_rd, io_wr} = 3'b0_0_0;
                        {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                        state = FETCH; end

      JAL:
                  begin// control word assignments:
                        @(negedge sys_clk)
                        int_ack=0;                    FS=5'h0;
                        {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b01_1_0_0;
                        {im_cs, im_rd, im_wr} = 3'b0_0_0;
                        {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                        11'b1_10_0_00_0_0_010;
                        {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                        {io_cs, io_rd, io_wr} = 3'b0_0_0;
                        {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                        state = FETCH; end


      /*******************E-TYPE*******************/

      OUTPUT:
                  begin// control word assignments:
                        @(negedge sys_clk)
                        int_ack=0;                    FS=5'h02;
                        {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                        {im_cs, im_rd, im_wr} = 3'b0_0_0;
                        {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                        11'b0_00_0_00_0_0_000;
                        {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                        {io_cs, io_rd, io_wr} = 3'b0_0_0;
                        {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                        state = OUTPUT2; end

      OUTPUT2:
                  begin// control word assignments:
                        @(negedge sys_clk)
                        int_ack=0;                    FS=5'h0;
                        {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                        {im_cs, im_rd, im_wr} = 3'b0_0_0;
```

```
                {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                11'b0_00_0_00_0_0_000;
                {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                {io_cs, io_rd, io_wr} = 3'b1_0_1;
                {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                state = FETCH; end

INPUT:
          begin// control word assignments:
                @(negedge sys_clk)
                int_ack=0;                      FS=5'h02;
                {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                {im_cs, im_rd, im_wr} = 3'b0_0_0;
                {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                11'b0_00_0_00_0_0_000;
                {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                {io_cs, io_rd, io_wr} = 3'b0_0_0;
                {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                state = INPUT2; end

INPUT2:
          begin// control word assignments:
                @(negedge sys_clk)
                int_ack=0;                      FS=5'h0;
                {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                {im_cs, im_rd, im_wr} = 3'b0_0_0;
                {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                11'b0_00_0_00_0_0_000;
                {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                {io_cs, io_rd, io_wr} = 3'b1_1_0;
                {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                state = INPUT3; end

INPUT3:
          begin// control word assignments:
                @(negedge sys_clk)
                int_ack=0;                      FS=5'h0;
                {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                {im_cs, im_rd, im_wr} = 3'b0_0_0;
                {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                11'b1_01_0_00_0_0_001;
                {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                {io_cs, io_rd, io_wr} = 3'b0_0_0;
                {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                state = FETCH; end

RETI:
          begin
          // Pass RS(RF[0x1D]) into ALU_OUT, RF[0x1D] = $sp
          // Would need SP_sel = 1'b1 if 0x1D hadn't been provided by RETI
          // ALU_OUT <- RS($sp)
                @(negedge sys_clk)
                int_ack=0;                      FS=5'h0;
                {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                {im_cs, im_rd, im_wr} = 3'b0_0_0;
                {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                11'b0_00_0_00_0_0_000;
                {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                {io_cs, io_rd, io_wr} = 3'b0_0_0;
                {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                state = RETI2; end

RETI2:
            begin
            // Post-increment POP the status flags from the stack into D_in
            // D_in is holding the status flags, will bus them
            // here(MCU) in next clock tick
            // Flags will arrive as inI, inC, inV, inN, inZ inputs
            // D_in <- dMem[ALU_OUT($sp)]
                @(negedge sys_clk)
                int_ack=0;                    FS=5'h0;
                {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                {im_cs, im_rd, im_wr} = 3'b0_0_0;
                {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                11'b0_00_0_00_0_0_000;
                {dm_cs, dm_rd, dm_wr} = 3'b1_1_0;
                {io_cs, io_rd, io_wr} = 3'b0_0_0;
                {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                state = RETI3; end

RETI3:
            begin
            // Send flags from D_in to MCU and assign values to NS flags
            // Complete post-increment POP by incrementing the $sp
            // Flags <- D_in, ALU_OUT <- ALU_OUT($sp) + 4
                @(negedge sys_clk)
                int_ack=0;                    FS=5'h11;
                {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                {im_cs, im_rd, im_wr} = 3'b0_0_0;
                {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                11'b0_00_0_00_1_0_000;
                {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                {io_cs, io_rd, io_wr} = 3'b0_0_0;
                {nsi, nsc, nsv, nsn, nsz} = {inI, inC, inV, inN, inZ};
                state = RETI4; end

RETI4:
            begin
            // Post-increment POP the return PC into D_in
            // D_in <- dMem[ALU_OUT($sp+4)], ALU_OUT <- ALU_OUT($sp+4) + 4
                @(negedge sys_clk)
                int_ack=0;                    FS=5'h11;
                {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                {im_cs, im_rd, im_wr} = 3'b0_0_0;
                {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                11'b0_00_0_00_1_0_000;
                {dm_cs, dm_rd, dm_wr} = 3'b1_1_0;
                {io_cs, io_rd, io_wr} = 3'b0_0_0;
                {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                state = RETI5; end

RETI5:
            begin
            // load the return PC into the PC register,
            // hold the $sp to be loaded into RF in next state
            // PC <- D_in(PC), ALU_OUT <- ALU_OUT($sp+8)
                @(negedge sys_clk)
                int_ack=0;                        FS=5'h0;
                {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_1_0_0;
                {im_cs, im_rd, im_wr} = 3'b0_0_0;
                {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                11'b0_00_0_00_1_0_001;
                {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                {io_cs, io_rd, io_wr} = 3'b0_0_0;
                {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                state = RETI6; end

RETI6:
            begin
            // load the $sp into RF[$29], $29 = $sp location
            // iMem should be fetching return PC now
            // RF[$sp] <- ALU_OUT($sp+8)
                @(negedge sys_clk)
                int_ack=0;                        FS=5'h0;
                {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                {im_cs, im_rd, im_wr} = 3'b0_0_0;
                {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                11'b1_11_0_00_0_0_000;
                {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                {io_cs, io_rd, io_wr} = 3'b0_0_0;
                {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                state = FETCH; end

E_KEY:
            begin// control word assignments:
                @(negedge sys_clk)
                int_ack=0;                        FS=5'h0;
                {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                {im_cs, im_rd, im_wr} = 3'b0_0_0;
                {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                11'b0_00_0_00_0_0_000;
                {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                {io_cs, io_rd, io_wr} = 3'b0_0_0;
                {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                state = FETCH; end


/*********************OTHER********************/

WB_alu:
            begin// control word assignments for R[rd] <-- ALU_Out
                @(negedge sys_clk)
                int_ack=0;                        FS=5'h0;
                {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                {im_cs, im_rd, im_wr} = 3'b0_0_0;
                {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                11'b1_00_0_00_0_0_000;
                {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                {io_cs, io_rd, io_wr} = 3'b0_0_0;
                {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                state = FETCH; end
```

```
WB_imm:
            begin// control word assignments for R[rt] <-- ALU_Out
                  @(negedge sys_clk)
                  int_ack=0;                    FS=5'h0;
                  {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                  {im_cs, im_rd, im_wr} = 3'b0_0_0;
                  {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                  11'b1_01_0_00_0_0_000;
                  {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                  {io_cs, io_rd, io_wr} = 3'b0_0_0;
                  {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                  state = FETCH; end

WB_mem:
            begin
            // control word assignments for M[ ALU_Out(rs+se_16)] <-- RT(rt)
                  @(negedge sys_clk)
                  int_ack=0;                    FS=5'h0;
                  {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                  {im_cs, im_rd, im_wr} = 3'b0_0_0;
                  {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                  11'b0_00_0_00_0_0_000;
                  {dm_cs, dm_rd, dm_wr} = 3'b1_0_1;
                  {io_cs, io_rd, io_wr} = 3'b0_0_0;
                  {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                  state = FETCH; end

BREAK:
            begin
                  $display("BREAK INSTRUCTION FETCHED %t",$time);
                  // control word assignments for "deasserting" everything
                  @(negedge sys_clk)
                  int_ack=0;                    FS=5'h0;
                  {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                  {im_cs, im_rd, im_wr} = 3'b0_0_0;
                  {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                  11'b0_00_0_00_0_0_000;
                  {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                  {io_cs, io_rd, io_wr} = 3'b0_0_0;
                  {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                  $display(" R E G I S T E R ' S A F T E R B R E A K");
                  $display(" ");
                  Dump_Registers; // task to output MIPS RegFile
                  $display(" ");
                  $display("time=%t M[3F0]=%h", $time,
                        {MIPS_CU_TB.Mem.dM[12'h3F0],
                        MIPS_CU_TB.Mem.dM[12'h3F1],
                        MIPS_CU_TB.Mem.dM[12'h3F2],
                        MIPS_CU_TB.Mem.dM[12'h3F3]});
                  $display(" ");
                  Mem_Dump;
                  $finish; end

ILLEGAL_OP:
            begin
                  $display("ILLEGAL OPCODE FETCHED %t",$time);
                  // control word assignments for "deasserting" everything
                  @(negedge sys_clk)
                  int_ack=0;                    FS=5'h0;
                  {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                  {im_cs, im_rd, im_wr} = 3'b0_0_0;
                  {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                  11'b0_00_0_00_0_0_000;
                  {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                  {io_cs, io_rd, io_wr} = 3'b0_0_0;
                  {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                  $display(" ");
                  $display("Memory:");
                  $display(" ");
                  Dump_Registers;
                  $display(" ");
                  Dump_PC_and_IR;
                  $finish; end

//INTR_1-INTR_3 for iMem01-iMem13
// start of iMem01-iMem13 INTR's
INTR_1:
            begin
            // PC gets address of interrupt vector; Save PC in $ra
            // control word assignments for ALU_Out <- 0x3FC, R[$ra] <- PC
                  @(negedge sys_clk)
                  int_ack=0;                    FS=5'h15;
                  {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                  {im_cs, im_rd, im_wr} = 3'b0_0_0;
                  {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                  11'b1_10_0_00_0_0_010;
                  {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                  {io_cs, io_rd, io_wr} = 3'b0_0_0;
                  {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                  state = INTR_2; end

INTR_2:
            begin
            // Read address of ISR into D_in;
            // control word assignments for D_in <- dM[ALU_Out(0x3FC]
                  @(negedge sys_clk)
                  int_ack=0;                    FS=5'h0;
                  {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                  {im_cs, im_rd, im_wr} = 3'b0_0_0;
                  {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                  11'b0_00_0_00_0_0_000;
                  {dm_cs, dm_rd, dm_wr} = 3'b1_1_0;
                  {io_cs, io_rd, io_wr} = 3'b0_0_0;
                  {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                  state = INTR_3; end

INTR_3:
            begin
            // Reload PC with address of ISR; ack the intr; goto FETCH
            // control word assignments for PC <- D_in( dM[0x3FC] ),
            // int_ack <- 1
                  @(negedge sys_clk)
                  int_ack=1;                    FS=5'h0;
                  {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_1_0_0;
                  {im_cs, im_rd, im_wr} = 3'b0_0_0;
                  {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                  11'b0_00_0_00_0_0_001;
                  {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                  {io_cs, io_rd, io_wr} = 3'b0_0_0;
                  {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                  state = FETCH; end
// end of iMem01-iMem13 INTR's


// INTR_1-INTR_6 for iMem14
// start of iMem14 INTR's
INTR_1:
            begin
            // Save PC in $ra, read $sp into RS
            // RF[$ra] <- PC, RS <- RF[$sp]
                  @(negedge sys_clk)
                  int_ack=0;                    FS=5'h0;
                  {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                  {im_cs, im_rd, im_wr} = 3'b0_0_0;
                  {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                  11'b1_10_1_00_0_0_010;
                  {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                  {io_cs, io_rd, io_wr} = 3'b0_0_0;
                  {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                  state = INTR_2; end

INTR_2:
            begin
            // Pass RS into ALU_OUT
            // ALU_OUT <- RS($sp)
                  @(negedge sys_clk)
                  int_ack=0;                    FS=5'h0;
                  {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                  {im_cs, im_rd, im_wr} = 3'b0_0_0;
                  {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                  11'b0_00_0_00_0_0_000;
                  {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                  {io_cs, io_rd, io_wr} = 3'b0_0_0;
                  {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                  state = INTR_3; end

INTR_3:
            begin
            // read memory(stack) address being pointed to by $sp into D_in
            // D_in <- dM[ALU_OUT($sp)]
                  @(negedge sys_clk)
                  int_ack=0;                    FS=5'h0;
                  {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                  {im_cs, im_rd, im_wr} = 3'b0_0_0;
                  {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                  11'b0_00_0_00_0_0_000;
                  {dm_cs, dm_rd, dm_wr} = 3'b1_1_0;
                  {io_cs, io_rd, io_wr} = 3'b0_0_0;
                  {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                  state = INTR_4; end
```

```
INTR_4:
            begin
      // load the value contained in the address pointed to by the $sp,
      // into PC,
      // stage the registers to pre-decrement PUSH the PC into the stack
      // PC <- D_in(dM[$sp]), ALU_OUT <- ALU_OUT - 4, RT <- PC
                  @(negedge sys_clk)
                  int_ack=0;                    FS=5'h12;
                  {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_1_0_0;
                  {im_cs, im_rd, im_wr} = 3'b0_0_0;
                  {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                  11'b0_00_0_10_1_0_001;
                  {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                  {io_cs, io_rd, io_wr} = 3'b0_0_0;
                  {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                  state = INTR_5; end

INTR_5:
            begin
            // PUSH PC onto stack
            // stage the registers to pre-decrement PUSH the status flags
            // into the stack
            // dMem[ALU_OUT($sp-4)] <- RT(PC), ALU_OUT <- ALU_OUT - 4,
            // RT <- status flags
                  @(negedge sys_clk)
                  int_ack=0;                    FS=5'h12;
                  {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                  {im_cs, im_rd, im_wr} = 3'b0_0_0;
                  {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                  11'b0_00_0_11_1_0_000;
                  {dm_cs, dm_rd, dm_wr} = 3'b1_0_1;
                  {io_cs, io_rd, io_wr} = 3'b0_0_0;
                  {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                  state = INTR_6; end

INTR_6:
            begin
      // PUSH status flags onto stack,
      // save current $sp value into the RF at address 29($sp), ack the intr
      // dMem[ALU_OUT($sp-8)] <- RT(status flags),
      // RF[$sp] <- ALU_OUT($sp-8), int_ack <- 1
                  @(negedge sys_clk)
                  int_ack=1'b1;                       FS=5'h0;
                  {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                  {im_cs, im_rd, im_wr} = 3'b0_0_0;
                  {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                  11'b1_11_0_00_0_0_000;
                  {dm_cs, dm_rd, dm_wr} = 3'b1_0_1;
                  {io_cs, io_rd, io_wr} = 3'b0_0_0;
                  {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                  state = FETCH; end
// end of iMem14 INTR's


/****************ENHANCEMENTS****************/
```

```
PUSH:
            begin
            // PUSH is pre-decrement, R-TYPE
            // pushes $rt into stack
            // RS <- RF[$sp], RT <- $rt
            // control word assignments:
                  @(negedge sys_clk)
                  int_ack=0;                    FS=5'h0;
                  {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                  {im_cs, im_rd, im_wr} = 3'b0_0_0;
                  {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                  11'b0_00_1_00_0_0_000;
                  {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                  {io_cs, io_rd, io_wr} = 3'b0_0_0;
                  {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                  state = PUSH2; end

PUSH2:
            begin
            // ALU_OUT <- RS($sp) - 4
            // control word assignments:
                  @(negedge sys_clk)
                  int_ack=0;                    FS=5'h12;
                  {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                  {im_cs, im_rd, im_wr} = 3'b0_0_0;
                  {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                  11'b0_00_0_00_0_0_000;
                  {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                  {io_cs, io_rd, io_wr} = 3'b0_0_0;
                  {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                  state = PUSH3; end

PUSH3:
            begin
            // dM[ALU_OUT($sp-4)] <- RT($rt)
            // control word assignments:
                  @(negedge sys_clk)
                  int_ack=0;                    FS=5'h0;
                  {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                  {im_cs, im_rd, im_wr} = 3'b0_0_0;
                  {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                  11'b0_00_0_00_1_0_000;
                  {dm_cs, dm_rd, dm_wr} = 3'b1_0_1;
                  {io_cs, io_rd, io_wr} = 3'b0_0_0;
                  {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                  state = PUSH4; end

PUSH4:
            begin
            // save new sp to $sp $sp <- ALU_OUT($sp)
            // control word assignments:
                  @(negedge sys_clk)
                  int_ack=0;                    FS=5'h0;
                  {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                  {im_cs, im_rd, im_wr} = 3'b0_0_0;
                  {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                  11'b1_11_0_00_0_0_000;
```

```
                {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                {io_cs, io_rd, io_wr} = 3'b0_0_0;
                {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                state = FETCH; end

POP:
            begin
            // POP is post-increment, I-TYPE
            // POP's top of stack into $rt
            // RS <- RF[$sp]
            // control word assignments:
                @(negedge sys_clk)
                int_ack=0;                   FS=5'h0;
                {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                {im_cs, im_rd, im_wr} = 3'b0_0_0;
                {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                11'b0_00_1_00_0_0_000;
                {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                {io_cs, io_rd, io_wr} = 3'b0_0_0;
                {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                state = POP2; end

POP2:
            begin
            // ALU_OUT <- RS($sp)
            // control word assignments:
                @(negedge sys_clk)
                int_ack=0;                   FS=5'h0;
                {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                {im_cs, im_rd, im_wr} = 3'b0_0_0;
                {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                11'b0_00_0_00_0_0_000;
                {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                {io_cs, io_rd, io_wr} = 3'b0_0_0;
                {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                state = POP3;
            end

POP3:
            begin
            // D_in <- dM[ALU_OUT($sp)], keep $sp in ALU_OUT
            // control word assignments:
            @(negedge sys_clk)
            int_ack=0;                   FS=5'h0;
            {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
            {im_cs, im_rd, im_wr} = 3'b0_0_0;
            {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
            11'b0_00_0_00_1_0_000;
            {dm_cs, dm_rd, dm_wr} = 3'b1_1_0;
            {io_cs, io_rd, io_wr} = 3'b0_0_0;
            {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
            state = POP4; end

POP4:
            begin
            // $rt <- D_in, ALU_OUT <- ALU_OUT($sp) + 4
            // control word assignments:
```

```
                        @(negedge sys_clk)
                        int_ack=0;                    FS=5'h11;
                        {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                        {im_cs, im_rd, im_wr} = 3'b0_0_0;
                        {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                        11'b1_01_0_00_1_0_001;
                        {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                        {io_cs, io_rd, io_wr} = 3'b0_0_0;
                        {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                        state = POP5; end

POP5:
                  begin
                  // $sp <- ALU_OUT
                  // control word assignments:
                        @(negedge sys_clk)
                        int_ack=0;                    FS=5'h0;
                        {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                        {im_cs, im_rd, im_wr} = 3'b0_0_0;
                        {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                        11'b1_11_0_00_0_0_000;
                        {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                        {io_cs, io_rd, io_wr} = 3'b0_0_0;
                        {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                        state = FETCH; end

DJNZ:
                  begin
            // opcode, rs, rt, imm16, I-TYPE, decrement RF[rs] then jumps
            // amount of instructions specified in imm16
            // $rs will be decremented, imm16 will specify how many instructions
            // will be jumped
            // $rs and $rt must be the same
            // if $rs is not zero then jump(branch to new address using imm16)
            // control word assignments:
                        @(negedge sys_clk)
                        int_ack=0;                    FS=5'h10;
                        {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b10_0_0_0;
                        {im_cs, im_rd, im_wr} = 3'b0_0_0;
                        {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                        11'b0_00_0_00_0_0_000;
                        {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                        {io_cs, io_rd, io_wr} = 3'b0_0_0;
                        {nsi, nsc, nsv, nsn, nsz} = {psi, c, v, n, z};
                        state = DJNZ2; end

DJNZ2:
                  begin
                  //$rt <- ALU_OUT
                  // control word assignments:
                        @(negedge sys_clk)
                        int_ack=0;                    FS=5'h0;
                        {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b10_0_0_0;
                        {im_cs, im_rd, im_wr} = 3'b0_0_0;
                        {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                        11'b1_01_0_00_0_0_000;
                        {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
```

```
                {io_cs, io_rd, io_wr} = 3'b0_0_0;
                {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                pc_ld = ~z;
                state = FETCH; end

NOP:
            begin
            // ALL registers remain the same for 1 clock cycle, R-TYPE
            // ALU_OUT <- ALU_OUT
            // control word assignments:
                @(negedge sys_clk)
                int_ack=0;                      FS=5'h0;
                {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                {im_cs, im_rd, im_wr} = 3'b0_0_0;
                {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                11'b0_00_0_00_1_0_000;
                {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                {io_cs, io_rd, io_wr} = 3'b0_0_0;
                {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                state = FETCH; end

CLR:
            begin
            // Clears(all zeros) in $rt, R_TYPE
            // ALU_OUT <- 0x0;
            // control word assignments:
                @(negedge sys_clk)
                int_ack=0;                      FS=5'h13;
                {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                {im_cs, im_rd, im_wr} = 3'b0_0_0;
                {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                11'b0_00_0_00_0_0_000;
                {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                {io_cs, io_rd, io_wr} = 3'b0_0_0;
                {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                state = CLR2; end

CLR2:
            begin
            // $rt <- ALU_OUT(0x0)
            // control word assignments:
                @(negedge sys_clk)
                int_ack=0;                      FS=5'h0;
                {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                {im_cs, im_rd, im_wr} = 3'b0_0_0;
                {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                11'b1_01_0_00_0_0_000;
                {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                {io_cs, io_rd, io_wr} = 3'b0_0_0;
                {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                state = FETCH; end

MOV:
            begin
            // moves RF[rs] into RF[rt], R-TYPE
            // ALU_OUT <- RS($rs)
            // control word assignments:
                @(negedge sys_clk)
                int_ack=0;                    FS=5'h0;
                {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                {im_cs, im_rd, im_wr} = 3'b0_0_0;
                {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                11'b0_00_0_00_0_0_000;
                {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                {io_cs, io_rd, io_wr} = 3'b0_0_0;
                {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                state = MOV2; end

MOV2:
            begin
            // $rt <- ALU_OUT($rs)
            // control word assignments:
                @(negedge sys_clk)
                int_ack=0;                    FS=5'h0;
                {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                {im_cs, im_rd, im_wr} = 3'b0_0_0;
                {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                11'b1_01_0_00_0_0_000;
                {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                {io_cs, io_rd, io_wr} = 3'b0_0_0;
                {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                state = FETCH; end

BLT:
            begin
            // branch if R[rs] < R[rt], I-TYPE
            // ALU_OUT <- RS($rs) - RT($rt)
            // control word assignments:
                @(negedge sys_clk)
                int_ack=0;                    FS=5'h04;
                {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                {im_cs, im_rd, im_wr} = 3'b0_0_0;
                {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                11'b0_00_0_00_0_0_000;
                {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                {io_cs, io_rd, io_wr} = 3'b0_0_0;
                {nsi, nsc, nsv, nsn, nsz} = {psi, c, v, n, z};
                state = BLT2; end

BLT2:
            begin
            // if N flag is set, then branch using imm16
            // control word assignments:
                @(negedge sys_clk)
                int_ack=0;                    FS=5'h0;
                {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b10_0_0_0;
                {im_cs, im_rd, im_wr} = 3'b0_0_0;
                {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                11'b0_00_0_00_0_0_000;
                {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                {io_cs, io_rd, io_wr} = 3'b0_0_0;
                {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                pc_ld = n;
                state = FETCH; end
```

```
BGE:
            begin
            // branch if R[rs] >= R[rt], I-TYPE
            // ALU_OUT <- RS($rs) - RT($rt)
            // control word assignments:
                  @(negedge sys_clk)
                  int_ack=0;                        FS=5'h04;
                  {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b00_0_0_0;
                  {im_cs, im_rd, im_wr} = 3'b0_0_0;
                  {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                  11'b0_00_0_00_0_0_000;
                  {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                  {io_cs, io_rd, io_wr} = 3'b0_0_0;
                  {nsi, nsc, nsv, nsn, nsz} = {psi, c, v, n, z};
                  state = BGE2; end

BGE2:
            begin// control word assignments:
                  @(negedge sys_clk)
                  int_ack=0;                        FS=5'h0;
                  {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b10_0_0_0;
                  {im_cs, im_rd, im_wr} = 3'b0_0_0;
                  {D_En, DA_sel, SP_sel, T_sel, S_sel, HILO_ld, Y_sel} =
                  11'b0_00_0_00_0_0_000;
                  {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
                  {io_cs, io_rd, io_wr} = 3'b0_0_0;
                  {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};
                  pc_ld = (z || ~n);
                  state = FETCH; end

endcase // end of FSM logic

//*********************************************************
//Integer Register File Dump Task
//*********************************************************
task Dump_Registers;
      begin
            for(i = 0; i < 16; i = i + 1) begin
                  $display ("t=%t   $r%0d = %h   ||  t=%t   $r%0d = %h",
                  $time, i, MIPS_CU_TB.id.regfile.RegArray[i],
                  $time, i+16, MIPS_CU_TB.id.regfile.RegArray[i+16]);
            end
      end
endtask

//*********************************************************
//PC and IR Register Dump Task
//*********************************************************
task Dump_PC_and_IR;
      begin
            $display(" ");
            $display("PC Register:");
            $display(" ");
            $display("t=%t PC=%h", $time, MIPS_CU_TB.iu.mypc.pc_out);
            $display(" ");
            $display("IR Register:");
```

```
			$display("t=%t IR=%h", $time, MIPS_CU_TB.iu.myir.ir_out);
			$display(" ");
			$display(" ");
		end
endtask

//*********************************************************
//Data Memory and I/O Memory Dump Task
//*********************************************************
task Mem_Dump;
		begin
			$display("        DATA MEMORY           IO MEMORY");
			for(i = 9'h0C0; i < 9'h100; i = i + 4) begin
				X = {MIPS_CU_TB.Mem.dM[i], MIPS_CU_TB.Mem.dM[i+1],
				MIPS_CU_TB.Mem.dM[i+2], MIPS_CU_TB.Mem.dM[i+3]};
				Y = {MIPS_CU_TB.Mem.ioM[i], MIPS_CU_TB.Mem.ioM[i+1],
				MIPS_CU_TB.Mem.ioM[i+2], MIPS_CU_TB.Mem.ioM[i+3]};
			$display("t=%t   DM[%h] = %h  ||  t=%t   IOM[%h] = %h", $time, i,
					X, $time, i, Y);
			end
		end
endtask

endmodule
```

MIPS Instruction Unit

```
`timescale 1ns / 1ps
/****************************** C E C S 4 4 0 ******************************
 *
 * File Name: Instruction_Unit.v
 * Project: Senior Design Project
 * Designer: Eduardo Marquez
 *                     Tuan Nguyen
 * Email: marquez.edu@hotmail.com
 *                tuannd.ryan2410@gmail.com
 * Rev. No.: Version 1.0
 * Rev. Date: 4/24/17
 *
 * Purpose: This module structurally connects the PC, Instruction Memory,
 *                    the IR, and a mux for the PC.
 *
 * Notes:
 *

***************************************************************************/
module Instruction_Unit(clk, reset, pc_sel, pc_ld, pc_inc, im_cs, im_wr,
im_rd, ir_ld, pc_in, pc_out, ir_out, se_16);
		// Input setting
		input			clk, reset, im_cs, im_wr, im_rd,
					pc_ld, pc_inc, ir_ld;
		input	[1:0]		pc_sel;
		input	[31:0]		pc_in;

		// Output setting
	output [31:0]     pc_out, ir_out, se_16;

	// Wire setting
	wire   [31:0]     pc_out, d_out, pc_mux, ir_out;

	// PC_MUX for PC_in, Branch, and Jump
	assign pc_mux = (pc_sel == 2'b10) ? {pc_out + {se_16[29:0],2'b00}}:
		// Branch {{16{ir_out[15]}},ir_out[15:0]}
			(pc_sel == 2'b01) ? {20'b0,ir_out[9:0],2'b00}:
		// Jump {pc_out[31:28],ir_out[25:0],2'b00}
			(pc_sel == 2'b00) ? pc_in : pc_mux;

	// Initialize Module
	//                  pc_ld, pc_inc, pc_in, pc_out
	PC mypc (clk, reset, pc_ld, pc_inc, pc_mux, pc_out);
	//                        im,cs, im_wr, im_rd,     addr,    d_in, d_out
	Instruction_Mem iMem (clk, im_cs, im_wr, im_rd, pc_out, 32'h0, d_out);
	//                  ir_ld, ir_in, ir_out
	IR myir (clk, reset, ir_ld, d_out, ir_out);

	assign se_16 = {{16{ir_out[15]}},ir_out[15:0]};


endmodule

`timescale 1ns / 1ps
/****************************** C E C S  4 4 0 ******************************
 *
 * File Name: Instruction_Unit.v
 * Project: Senior Design Project
 * Designer: Eduardo Marquez
 *                           Tuan Nguyen
 * Email: marquez.edu@hotmail.com
 *                     tuannd.ryan2410@gmail.com
 * Rev. No.: Version 1.0
 * Rev. Date: 4/24/17
 *
 * Purpose: A register for Program Counter.
 *
 * Notes:
 *

*****************************************************************************/
module PC(clk, reset, pc_ld, pc_inc, pc_in, pc_out);

	input clk, reset, pc_ld, pc_inc;
	input [31:0] pc_in;
	output [31:0] pc_out;
	reg [31:0] pc_out;

	always@(posedge clk, posedge reset) begin
		if(reset)
			pc_out <= 32'b0;
		else
			case({pc_ld,pc_inc})
				2'b10: pc_out <= pc_in;
				2'b01: pc_out <= pc_out + 4;
```

```
                        default: pc_out <= pc_out;
                  endcase
      end // always block

endmodule

`timescale 1ns / 1ps
/****************************** C E C S 4 4 0 ******************************
 *
 * File Name: Instruction_Unit.v
 * Project: Senior Design Project
 * Designer: Eduardo Marquez
 *                      Tuan Nguyen
 * Email: marquez.edu@hotmail.com
 *                tuannd.ryan2410@gmail.com
 * Rev. No.: Version 1.0
 * Rev. Date: 4/24/17
 *
 * Purpose: A memory of Control Unit.
 *
 * Notes:
 *

****************************************************************************/
module Instruction_Mem(clk, im_cs, im_wr, im_rd, addr, d_in, d_out);

       // Inputs & Outputs setting
       input                clk, im_cs, im_wr, im_rd;
       input  [31:0] addr, d_in;
       output [31:0] d_out;

       // 4096x8 Byte Addressable Memory
       reg    [7:0] irMem [0:4095];

       // Tri state D_out that will read when im_rd is true or
       // high impedance when im_wr is true.
       assign d_out = (im_cs && im_rd && !im_wr) ?
            {irMem[addr],irMem[addr+1],irMem[addr+2],irMem[addr+3]} : 32'hz;

       always @(posedge clk) begin
       if(im_cs && im_wr)
       {irMem[addr],irMem[addr+1],irMem[addr+2],irMem[addr+3]}<=
                                {d_in[31:24],d_in[23:16],d_in[15:8],d_in[7:0]};
       else
       {irMem[addr],irMem[addr+1],irMem[addr+2],irMem[addr+3]}<=
                     {irMem[addr],irMem[addr+1],irMem[addr+2],irMem[addr+3]};
       end


endmodule

`timescale 1ns / 1ps
/****************************** C E C S 4 4 0 ******************************
 *
 * File Name: Instruction_Unit.v
 * Project: Senior Design Project
 * Designer: Eduardo Marquez
 *                        Tuan Nguyen
 * Email: marquez.edu@hotmail.com
 *                  tuannd.ryan2410@gmail.com
 * Rev. No.: Version 1.0
 * Rev. Date: 4/24/17
 *
 * Purpose: A register of Instruction Register.
 *
 * Notes:
 *

****************************************************************************/
module IR(clk, reset, ir_ld, ir_in, ir_out);

      input clk, reset, ir_ld;
      input [31:0] ir_in;
      output [31:0] ir_out;
      reg [31:0] ir_out;

      always@(posedge clk, posedge reset) begin
            if(reset)
                  ir_out <= 32'b0;
            else
                  if(ir_ld)
                        ir_out <= ir_in;
                  else
                        ir_out <= ir_out;
      end   // end always block

endmodule
```

MIPS Integer Datapath

```
`timescale 1ns / 1ps
/****************************** C E C S 4 4 0 ******************************
 *
 * File Name: Integer_Datapath.v
 * Project: Senior Design Project
 * Designer: Eduardo Marquez, Tuan Nguyen
 * Email: marquez.edu@hotmail.com, tuannd.ryan2410@gmail.com
 * Rev. No.: Version 1.0
 * Rev. Date: 4/24/17
 *
 * Purpose: This file structurally connects the register file module with the
 * ALU module. A 2-1 mux and a 5-1 mux decide which path to output will be
 * taken. Two 32 bit registers hold data from the ALU.
 *
 * Notes:
 *

****************************************************************************/
module Integer_Datapath(clk, reset, D_En, DA_sel, D_Addr, S_Addr, T_Addr,
shamt, DT, T_Sel, FS, C, N, Z, V, HILO_ld, DY, PC_in, Y_Sel, ALU_OUT, D_OUT,
S_sel, SP_sel, inI, inC, inV, inN, inZ, outI, outC, outV, outN, outZ);

      input       clk, reset, D_En, HILO_ld, S_sel;
      input [1:0] DA_sel;
      input [4:0] D_Addr, S_Addr, T_Addr, FS, shamt;
      input [31:0] DT, DY, PC_in;
      input [2:0] Y_Sel;
      input       SP_sel;
      input [1:0] T_Sel;
      input       inI, inC, inV, inN, inZ; //flags coming in from MCU

      output      C, N, Z, V;
      output [31:0] ALU_OUT, D_OUT;
      output      outI, outC, outV, outN, outZ; //flags going out to MCU

      wire [31:0] S, Sout, T, Tout, Y_hi, Y_lo, hiOut, loOut, aluin, din;
      wire        C, N, Z, V;
      wire [31:0] ALU_OUT, D_OUT;
      wire [4:0]  D_MUX;
      wire [31:0] Smux;
      wire [4:0]  SP_MUX;
      wire        inI, inC, inV, inN, inZ, outI, outC, outV, outN, outZ;

      //**************** D - M U X ******************************
      //DA_sel = 0, D_MUX = D_Addr,
      //DA_sel = 1, D_MUX = T_Addr,
      //DA_sel = 2, D_MUX = 31 bit reserved data,
      //DA_sel = 3, D_MUX = 29 bit reserved data.
      assign D_MUX = (DA_sel == 2'b00) ? D_Addr :
                  (DA_sel == 2'b01) ? T_Addr :
                  (DA_sel == 2'b10) ? 32'h1F :  // $ra addr
                  (DA_sel == 2'b11) ? 32'h1D :  // $sp addr
                                    D_Addr;     // default

      // mux to S_Addr of regfile, SP_sel = 1, inputs 29, for $sp
      assign SP_MUX = (SP_sel == 1'b0) ? S_Addr : 32'h1D;

      //instantiate regfile
      //                  clk, reset, D_en, D_Addr, S_Addr, T_Addr,
      regfile32 regfile (clk, reset, D_En, D_MUX, SP_MUX, T_Addr,
      //                   D,      S,T
                        ALU_OUT, S, T);

      //T_MUX, receives either RF T output, imm16, PC, or flags from MCU
      assign Tout = (T_Sel == 2'b00)? T     :
                  (T_Sel == 2'b01) ? DT    :
                  (T_Sel == 2'b10) ? PC_in :
                  (T_Sel == 2'b11) ? {27'b0, inI, inC, inV, inN, inZ} : T;

      // mux from ALU_OUT to S input of ALU, used to decrement $sp for INTR
      // and RETI takes ALU_OUT or output of RS
      assign Smux = (S_sel == 1'b0) ? Sout : aluin;

      //instantiate alu
      //          S,    T,      FS, Y_hi, Y_lo, shamt, C, V, N, Z
      ALU_32 alu (Smux, D_OUT, FS, Y_hi, Y_lo, shamt, C, V, N, Z);

      //HILO registers
      //          clk, reset, ld,      D,    Q
      reg32 HI (clk, reset, HILO_ld, Y_hi, hiOut),
             LO (clk, reset, HILO_ld, Y_lo, loOut);

       //pipelining registers
       //                 clk, reset, D, Q
       reg32NoLD    RS     (clk, reset, S, Sout),
                    RT     (clk, reset, Tout, D_OUT),
                    ALU_Out (clk, reset, Y_lo, aluin),
                    D_in   (clk, reset, DY, din);

       // send flags out to MCU
       assign {outI, outC, outV, outN, outZ} = din[4:0];

       //Y_MUX
       assign ALU_OUT = (Y_Sel == 3'b000) ? (aluin) :
                                         (Y_Sel == 3'b001) ? (din)    :
                                         (Y_Sel == 3'b010) ? (PC_in)  :
                                         (Y_Sel == 3'b011) ? (hiOut)  :
                                         (Y_Sel == 3'b100) ? (loOut)  :
                                                               (32'b0);

Endmodule

`timescale 1ns / 1ps
/****************************** C E C S 4 4 0 ******************************
 *
 * File Name: regfile32.v
 * Project: Senior Design Project
 * Designer: Eduardo Marquez
 * Email: marquez.edu@hotmail.com
 * Rev. No.: Version 1.0
 * Rev. Date: 4/24/17
 *
 * Purpose: This is a 32x32 register file. This will be used to hold data
 *                      for other modules.
 *
 * Notes:
 *

****************************************************************************/
module regfile32(clk, reset, D_en, D_Addr, S_Addr, T_Addr, D, S, T);

       input                  clk, reset, D_en;
       input   [4:0]          D_Addr, S_Addr, T_Addr;
       input   [31:0]         D;
       output wire [31:0]     S, T;

       reg [31:0] RegArray [31:0]; //31 is first in list, 0 is last in list

       //read
       assign S = RegArray[S_Addr];
       assign T = RegArray[T_Addr];

       //write
       always @ (posedge clk, posedge reset)
            //for reset, only $r0 is set to 0
            if(reset)
                        RegArray[0] <= 32'h0;
            else
                  if(D_en)
                        //$r0 is read only, dont write to it
                        if(D_Addr != 0)
                              RegArray[D_Addr] <= D;
                        else
                              RegArray[D_Addr] <= RegArray[D_Addr];
                  else
                        RegArray[D_Addr] <= RegArray[D_Addr];

Endmodule

`timescale 1ns / 1ps
/****************************** C E C S 4 4 0 ******************************
 *
 * File Name: ALU_32.v
 * Project: Senior Design Project
 * Designer: Eduardo Marquez
 * Email: marquez.edu@hotmail.com
 * Rev. No.: Version 1.0
 * Rev. Date: 4/24/17
 *
 * Purpose: This module is a wrapper for the MIPS_32, MUL_32, and DIV_32
 *    modules. It structurally connects the modules together to form a 32 bit
 *    ALU with a variety of operations.
 *
 * Notes:
 *

****************************************************************************/
module ALU_32(S, T, FS, Y_hi, Y_lo, shamt, C, V, N, Z);

       input       [31:0]      S, T;
       input       [4:0]       FS;
       input       [4:0]       shamt;
       output wire[31:0]       Y_hi, Y_lo;
       output wire             C, V, N, Z;

       wire        [31:0]      Y, rem, quo, Y_shift;
       wire        [63:0]      mul;
       wire                    mipsC,mipsV,mipsN,mipsZ,mpyZ,divZ;
       wire                    bsN,bsZ;
       wire        [2:0]       sType;


       //instantiate
       MIPS_32 MIPS  (S,T,FS,Y,mipsC,mipsV);
       MPY_32  MUL   (S,T,mul);
       DIV_32  DIV   (S,T,rem,quo);
       BShift32bit BSHIFT (T,shamt,sType,Y_shift,bsN,bsZ);

       //Shift types
        assign sType = (FS == 5'h0C) ? 3'b000: // Shift left logic
                        (FS == 5'h0D) ? 3'b001: // Shift right logic
                        (FS == 5'h0E) ? 3'b011: // Shift right arithmetic
                        (FS == 5'h1A) ? 3'b010: // Shift left arithmetic
                        (FS == 5'h1B) ? 3'b100: // Rotate left without carry
```

```
                              (FS == 5'h1C) ? 3'b101: // Rotate right without carry
                                              3'b111; // No Shift

         //mips negative flag, 0 for unsigned
         assign mipsN = (FS == 5'h03) ? 1'b0 :
                        (FS == 5'h05) ? 1'b0 :
                        (FS == 5'h07) ? 1'b0 :
                                     Y[31];

         //zero flags
         assign mipsZ = (Y == 32'b0)   ? 1'b1 : 1'b0;
         assign mpyZ =  (mul == 64'b0) ? 1'b1 : 1'b0;
         assign divZ =  (quo == 32'b0) ? 1'b1 : 1'b0;


//if FS==IE, then output MUL, if FS==1F, then DIV, if neither, then output
MIPS
assign{Y_hi,Y_lo,C,V,N,Z} =
(FS == 5'h1E) ? {mul[63:32],mul[31:0], 1'bx,1'bx,mul[63],mpyZ} :  //MUL
(FS == 5'h1F) ? {rem,quo,1'bx,1'bx,quo[31],divZ} :                //DIV
(FS == 5'h0C || FS == 5'h0D || FS == 5'h0E ||
FS == 5'h1A || FS == 5'h1B || FS == 5'h1C) ?
                              {32'b0,Y_shift,1'bx,1'bx,bsN,bsZ}:  //BShift
                              {32'b0,Y,mipsC,mipsV,mipsN,mipsZ};  //MIPS


Endmodule

`timescale 1ns / 1ps
/****************************** C E C S 4 4 0 ******************************
 *
 * File Name: MIPS_32.v
 * Project: Senior Design Project
 * Designer: Eduardo Marquez
 * Email: marquez.edu@hotmail.com
 * Rev. No.: Version 1.0
 * Rev. Date: 4/24/17
 *
 * Purpose: This module performs most of the operations for the ALU. The FS
 * input will determine which operation will be executed. It outputs a 32 bit
 * hex value along with the flags for the respective operations.
 *
 * Notes:
 *
****************************************************************************/
module MIPS_32(S, T, FS, Y, C, V);

      input        [31:0]      S, T;
      input        [4:0]       FS;
      output reg   [31:0]      Y;
      output reg                           C, V;

      reg          [1:0]       temp;

      //cast the inputs as integers, makes them signed
      integer int_s, int_t;
```

```
always @ (S or T or FS) begin
case(FS)
//ARITHMETIC
5'h00: {V,C,Y} = {1'bx,1'bx,S}; //PASS_S
5'h01: {V,C,Y} = {1'bx,1'bx,T}; //PASS_T
5'h02: begin //ADD
//convert to signed integers, needed for signed arithmetic
      int_s = S;
      int_t = T;
      {C,Y} = S + T;

//if 2 equal signs are added together and result in an
//opposite sign, then overflow is HIGH
      V = ((int_s[31] & int_t[31] & ~Y[31]) |
            (~int_s[31] & ~int_t[31] & Y[31])); end
5'h03: begin //ADDU
      {C,Y} = S + T; //overflow will be determined by the carry bit
      V = C; end
5'h04: begin //SUB
      int_s = S;
      int_t = T;
      {C,Y} = S - T;

      //if 2 equal signs are added together and result in an
      //opposite sign, then overflow is HIGH
      V = ((~int_s[31] & int_t[31] & Y[31]) |
            (int_s[31] & ~int_t[31] & ~Y[31])); end
5'h05: begin //SUBU
      {C,Y} = S - T;
      V = C; end
5'h06: begin //SLT
      {V,C} = {1'bx,1'bx};
      int_s = S;
      int_t = T;
      if(int_s < int_t) Y = 1'b1;
      else Y = 1'b0; end
5'h07: begin //SLTU
      {V,C} = {1'bx,1'bx};
      if(S < T) Y = 1'b1;
      else Y = 1'b0; end

//LOGIC
5'h08: {V,C,Y} = {1'bx,1'bx,(S & T)}; //AND
5'h09: {V,C,Y} = {1'bx,1'bx,(S | T)}; //OR
5'h0A: {V,C,Y} = {1'bx,1'bx,(S ^ T)}; //XOR
5'h0B: {V,C,Y} = {1'bx,1'bx,(~(S | T))};//NOR
5'h0F: begin //INC
      //using temp[1] to hold sign bit of S
      temp[1] = S[31];

      //increment by 1
      int_s = S;
      {C,Y} = S + 1;

      //temp[0] holds new incremented S sign bit
      temp[0] = S[31];
```

```
            //if a positive plus a positive equals
            //a negative, then oveflow is HIGH
            V = (~temp[1] & temp[0]); end
      5'h10: begin//DEC
            temp[1] = S[31];
            int_s = S;
            {C,Y} = S - 1;
            temp[0] = S[31];
            //if a negative plus a negative equals
            //a positive, then overflow is HIGH
            V = (temp[1] & ~temp[0]); end
      5'h11: begin      //INC4
            temp[1] = S[31];
            int_s = S;
            {C,Y} = S + 4;
            temp[0] = S[31];
            //if a positive plus a positive equals
            //a negative, then oveflow is HIGH
            V = (~temp[1] & temp[0]); end
      5'h12: begin      //DEC4
            temp[1] = S[31];
            int_s = S;
            {C,Y} = S - 4;
            temp[0] = S[31];
            //if a negative plus a negative equals a
            //positive, then overflow is HIGH
            V = (temp[1] & ~temp[0]); end
      5'h13: {V,C,Y} = {1'b0,1'b0,32'h0}; // ZEROS
      5'h14: {V,C,Y} = {1'b0,1'b0,32'hFFFFFFFF};//ONES
      5'h15: {V,C,Y} = {1'b0,1'b0,32'h3FC};//SP_INIT

      //LOGIC IMMEDIATE
      5'h16: {V,C,Y} = {1'bx,1'bx,(S&{16'h0,T[15:0]})};     //ANDI
      5'h17: {V,C,Y} = {1'bx,1'bx,(S|{16'h0,T[15:0]})};     //ORI
      5'h18: {V,C,Y} = {1'bx,1'bx,(S^{16'h0,T[15:0]})};     //XORI
      5'h19: {V,C,Y} = {1'bx,1'bx,{T[15:0],16'h0}};         //LUI
      default: {V,C,Y} = {1'b0,1'b0,32'h0};
      endcase
      end

endmodule

`timescale 1ns / 1ps
/****************************** C E C S 4 4 0 ******************************
 *
 * File Name: MPY_32.v
 * Project: Senior Design Project
 * Designer: Eduardo Marquez
 * Email: marquez.edu@hotmail.com
 * Rev. No.: Version 1.0
 * Rev. Date: 4/24/17
 *
 * Purpose: This module performs the multiplication operation for the ALU.
 *
 * Notes:
 *
```

```
*****************************************************************************/
module MPY_32(a, b, mul);

      input             [31:0] a, b;
      output reg        [63:0] mul;

      //cast the inputs as integers, makes them signed
      integer int_a, int_b;

      always @(a or b) begin
            //convert to signed integers
            int_a = a;
            int_b = b;
            mul = int_a * int_b;
      end

endmodule

`timescale 1ns / 1ps
/****************************** C E C S 4 4 0 ******************************
 *
 * File Name: DIV_32.v
 * Project: Senior Design Project
 * Designer: Eduardo Marquez
 * Email: marquez.edu@hotmail.com
 * Rev. No.: Version 1.0
 * Rev. Date: 4/24/17
 *
 * Purpose: This module performs the divide operation for the ALU.
 *
 * Notes:
 *

*****************************************************************************/
module DIV_32(a, b, rem, quo);

      input                   [31:0] a, b;
      output reg        [31:0] rem, quo;

      //cast the inputs as integers, makes them signed
      integer int_a, int_b;

      always @(a or b) begin
            //convert to signed integers
            int_a = a;
            int_b = b;
            quo = int_a / int_b;
            rem = int_a % int_b;
      end

endmodule

`timescale 1ns / 1ps
/****************************** C E C S 4 4 0 ******************************
 *
 * File Name: BShift32bit.v
 * Project: Senior Design Project
 * Designer: Eduardo Marquez
 *                        Tuan Nguyen
 * Email: marquez.edu@hotmail.com
 *                  tuannd.ryan2410@gmail.com
 * Rev. No.: Version 1.0
 * Rev. Date: 4/24/17
 *
 * Purpose: 32 bit Barrel Shifter:
 *                       + shift left logical
 *                       + shift right logical
 *                       + shift left arithmetic
 *                       + shift right arithmetic
 *                       + rotation left
 *                       + rotation right
 *
 *

*****************************************************************************/
module BShift32bit(T,shamt,sType,Y,N,Z);

      input [31:0] T;
      input [4:0] shamt;
      input [2:0] sType;
      output reg [31:0] Y;
      output wire N,Z;

      always@(*) begin
            case({sType,shamt})

                  // left shift

                  8'b000_00001: Y = {T[30:0],1'b0};

                  8'b000_00010: Y = {T[29:0],2'b0};

                  8'b000_00011: Y = {T[28:0],3'b0};

                  8'b000_00100: Y = {T[27:0],4'b0};

                  8'b000_00101: Y = {T[26:0],5'b0};

                  8'b000_00110: Y = {T[25:0],6'b0};

                  8'b000_00111: Y = {T[24:0],7'b0};

                  8'b000_01000: Y = {T[23:0],8'b0};

                  8'b000_01001: Y = {T[22:0],9'b0};

                  8'b000_01010: Y = {T[21:0],10'b0};

                  8'b000_01011: Y = {T[20:0],11'b0};

                  8'b000_01100: Y = {T[19:0],12'b0};

                  8'b000_01101: Y = {T[18:0],13'b0};
```

```
8'b000_01110: Y = {T[17:0],14'b0};

8'b000_01111: Y = {T[16:0],15'b0};

8'b000_10000: Y = {T[15:0],16'b0};

8'b000_10001: Y = {T[14:0],17'b0};

8'b000_10010: Y = {T[13:0],18'b0};

8'b000_10011: Y = {T[12:0],19'b0};

8'b000_10100: Y = {T[11:0],20'b0};

8'b000_10101: Y = {T[10:0],21'b0};

8'b000_10110: Y = {T[9:0],22'b0};

8'b000_10111: Y = {T[8:0],23'b0};

8'b000_11000: Y = {T[7:0],24'b0};

8'b000_11001: Y = {T[6:0],25'b0};

8'b000_11010: Y = {T[5:0],26'b0};

8'b000_11011: Y = {T[4:0],27'b0};

8'b000_11100: Y = {T[3:0],28'b0};

8'b000_11101: Y = {T[2:0],29'b0};

8'b000_11110: Y = {T[1:0],30'b0};

8'b000_11111: Y = {T[0],31'b0};

// shift right

8'b001_00001: Y = {1'b0,T[31:1]};

8'b001_00010: Y = {2'b0,T[31:2]};

8'b001_00011: Y = {3'b0,T[31:3]};

8'b001_00100: Y = {4'b0,T[31:4]};

8'b001_00101: Y = {5'b0,T[31:5]};

8'b001_00110: Y = {6'b0,T[31:6]};

8'b001_00111: Y = {7'b0,T[31:7]};

8'b001_01000: Y = {8'b0,T[31:8]};

8'b001_01001: Y = {9'b0,T[31:9]};

8'b001_01010: Y = {10'b0,T[31:10]};
```

```
8'b001_01011: Y = {11'b0,T[31:11]};

8'b001_01100: Y = {12'b0,T[31:12]};

8'b001_01101: Y = {13'b0,T[31:13]};

8'b001_01110: Y = {14'b0,T[31:14]};

8'b001_01111: Y = {15'b0,T[31:15]};

8'b001_10000: Y = {16'b0,T[31:16]};

8'b001_10001: Y = {17'b0,T[31:17]};

8'b001_10010: Y = {18'b0,T[31:18]};

8'b001_10011: Y = {19'b0,T[31:19]};

8'b001_10100: Y = {20'b0,T[31:20]};

8'b001_10101: Y = {21'b0,T[31:21]};

8'b001_10110: Y = {22'b0,T[31:22]};

8'b001_10111: Y = {23'b0,T[31:23]};

8'b001_11000: Y = {24'b0,T[31:24]};

8'b001_11001: Y = {25'b0,T[31:25]};

8'b001_11010: Y = {26'b0,T[31:26]};

8'b001_11011: Y = {27'b0,T[31:27]};

8'b001_11100: Y = {28'b0,T[31:28]};

8'b001_11101: Y = {29'b0,T[31:29]};

8'b001_11110: Y = {30'b0,T[31:30]};

8'b001_11111: Y = {31'b0,T[31]};

// shift left arithmetic

8'b010_00001: Y = {T[30:1],{2{T[0]}}};

8'b010_00010: Y = {T[29:1],{3{T[0]}}};

8'b010_00011: Y = {T[28:1],{4{T[0]}}};

8'b010_00100: Y = {T[27:1],{5{T[0]}}};

8'b010_00101: Y = {T[26:1],{6{T[0]}}};

8'b010_00110: Y = {T[25:1],{7{T[0]}}};
```

```
8'b010_00111: Y = {T[24:1],{8{T[0]}}};
8'b010_01000: Y = {T[23:1],{9{T[0]}}};
8'b010_01001: Y = {T[22:1],{10{T[0]}}};
8'b010_01010: Y = {T[21:1],{11{T[0]}}};
8'b010_01011: Y = {T[20:1],{12{T[0]}}};
8'b010_01100: Y = {T[19:1],{13{T[0]}}};
8'b010_01101: Y = {T[18:1],{14{T[0]}}};
8'b010_01110: Y = {T[17:1],{15{T[0]}}};
8'b010_01111: Y = {T[16:1],{16{T[0]}}};
8'b010_10000: Y = {T[15:1],{17{T[0]}}};
8'b010_10001: Y = {T[14:1],{18{T[0]}}};
8'b010_10010: Y = {T[13:1],{19{T[0]}}};
8'b010_10011: Y = {T[12:1],{20{T[0]}}};
8'b010_10100: Y = {T[11:1],{21{T[0]}}};
8'b010_10101: Y = {T[10:1],{22{T[0]}}};
8'b010_10110: Y = {T[9:1],{23{T[0]}}};
8'b010_10111: Y = {T[8:1],{24{T[0]}}};
8'b010_11000: Y = {T[7:1],{25{T[0]}}};
8'b010_11001: Y = {T[6:1],{26{T[0]}}};
8'b010_11010: Y = {T[5:1],{27{T[0]}}};
8'b010_11011: Y = {T[4:1],{28{T[0]}}};
8'b010_11101: Y = {T[3:1],{29{T[0]}}};
8'b010_11110: Y = {T[2:1],{30{T[0]}}};
8'b010_11111: Y = {T[1],{31{T[0]}}};
// shift right arithmetic
8'b011_00001: Y = {{2{T[31]}},T[30:1]};
8'b011_00010: Y = {{3{T[31]}},T[30:2]};
8'b011_00011: Y = {{4{T[31]}},T[30:3]};
8'b011_00100: Y = {{5{T[31]}},T[30:4]};
```

```
8'b011_00101: Y = {{6{T[31]}},T[30:5]};

8'b011_00110: Y = {{7{T[31]}},T[30:6]};

8'b011_00111: Y = {{8{T[31]}},T[30:7]};

8'b011_01000: Y = {{9{T[31]}},T[30:8]};

8'b011_01001: Y = {{10{T[31]}},T[30:9]};

8'b011_01010: Y = {{11{T[31]}},T[30:10]};

8'b011_01011: Y = {{12{T[31]}},T[30:11]};

8'b011_01100: Y = {{13{T[31]}},T[30:12]};

8'b011_01101: Y = {{14{T[31]}},T[30:13]};

8'b011_01110: Y = {{15{T[31]}},T[30:14]};

8'b011_01111: Y = {{16{T[31]}},T[30:15]};

8'b011_10000: Y = {{17{T[31]}},T[30:16]};

8'b011_10001: Y = {{18{T[31]}},T[30:17]};

8'b011_10010: Y = {{19{T[31]}},T[30:18]};

8'b011_10011: Y = {{20{T[31]}},T[30:19]};

8'b011_10100: Y = {{21{T[31]}},T[30:20]};

8'b011_10101: Y = {{22{T[31]}},T[30:21]};

8'b011_10110: Y = {{23{T[31]}},T[30:22]};

8'b011_10111: Y = {{24{T[31]}},T[30:23]};

8'b011_11000: Y = {{25{T[31]}},T[30:24]};

8'b011_11001: Y = {{26{T[31]}},T[30:25]};

8'b011_11010: Y = {{27{T[31]}},T[30:26]};

8'b011_11011: Y = {{28{T[31]}},T[30:27]};

8'b011_11101: Y = {{29{T[31]}},T[30:28]};

8'b011_11110: Y = {{30{T[31]}},T[30:29]};

8'b011_11111: Y = {{31{T[31]}},T[30]};

// rotation left

8'b100_00001: Y = {T[30:0],T[31]};
```

```
8'b100_00010: Y = {T[29:0],T[31:30]};

8'b100_00011: Y = {T[28:0],T[31:29]};

8'b100_00100: Y = {T[27:0],T[31:28]};

8'b100_00101: Y = {T[26:0],T[31:27]};

8'b100_00110: Y = {T[25:0],T[31:26]};

8'b100_00111: Y = {T[24:0],T[31:25]};

8'b100_01000: Y = {T[23:0],T[31:24]};

8'b100_01001: Y = {T[22:0],T[31:23]};

8'b100_01010: Y = {T[21:0],T[31:22]};

8'b100_01011: Y = {T[20:0],T[31:21]};

8'b100_01100: Y = {T[19:0],T[31:20]};

8'b100_01101: Y = {T[18:0],T[31:19]};

8'b100_01110: Y = {T[17:0],T[31:18]};

8'b100_01111: Y = {T[16:0],T[31:17]};

8'b100_10000: Y = {T[15:0],T[31:16]};

8'b100_10001: Y = {T[14:0],T[31:15]};

8'b100_10010: Y = {T[13:0],T[31:14]};

8'b100_10011: Y = {T[12:0],T[31:13]};

8'b100_10100: Y = {T[11:0],T[31:12]};

8'b100_10101: Y = {T[10:0],T[31:11]};

8'b100_10110: Y = {T[9:0],T[31:10]};

8'b100_10111: Y = {T[8:0],T[31:9]};

8'b100_11000: Y = {T[7:0],T[31:8]};

8'b100_11001: Y = {T[6:0],T[31:7]};

8'b100_11010: Y = {T[5:0],T[31:6]};

8'b100_11011: Y = {T[4:0],T[31:5]};

8'b100_11100: Y = {T[3:0],T[31:4]};

8'b100_11101: Y = {T[2:0],T[31:3]};

8'b100_11110: Y = {T[1:0],T[31:2]};
```

```
8'b100_11111: Y = {T[0],T[31:1]};
// rotation right
8'b101_00001: Y = {T[0],T[31:1]};
8'b101_00010: Y = {T[1:0],T[31:2]};
8'b101_00011: Y = {T[2:0],T[31:3]};
8'b101_00100: Y = {T[3:0],T[31:4]};
8'b101_00101: Y = {T[4:0],T[31:5]};
8'b101_00110: Y = {T[5:0],T[31:6]};
8'b101_00111: Y = {T[6:0],T[31:7]};
8'b101_01000: Y = {T[7:0],T[31:8]};
8'b101_01001: Y = {T[8:0],T[31:9]};
8'b101_01010: Y = {T[9:0],T[31:10]};
8'b101_01011: Y = {T[10:0],T[31:11]};
8'b101_01100: Y = {T[11:0],T[31:12]};
8'b101_01101: Y = {T[12:0],T[31:13]};
8'b101_01110: Y = {T[13:0],T[31:14]};
8'b101_01111: Y = {T[14:0],T[31:15]};
8'b101_10000: Y = {T[15:0],T[31:16]};
8'b101_10001: Y = {T[16:0],T[31:17]};
8'b101_10010: Y = {T[17:0],T[31:18]};
8'b101_10011: Y = {T[18:0],T[31:19]};
8'b101_10100: Y = {T[19:0],T[31:20]};
8'b101_10101: Y = {T[20:0],T[31:21]};
8'b101_10110: Y = {T[21:0],T[31:22]};
8'b101_10111: Y = {T[22:0],T[31:23]};
8'b101_11000: Y = {T[23:0],T[31:24]};
8'b101_11001: Y = {T[24:0],T[31:25]};
8'b101_11010: Y = {T[25:0],T[31:26]};
```

```
                        8'b101_11011: Y = {T[26:0],T[31:27]};

                        8'b101_11100: Y = {T[27:0],T[31:28]};

                        8'b101_11101: Y = {T[28:0],T[31:29]};

                        8'b101_11110: Y = {T[29:0],T[31:30]};

                        8'b101_11111: Y = {T[30:0],T[31]};
                endcase

        end//always block

        assign Z = (Y == 31'b0) ? 1'b1 : 1'b0;

        assign N = Y[31];

endmodule

`timescale 1ns / 1ps
/****************************** C E C S 4 4 0 ******************************
 *
 * File Name: reg32.v
 * Project: Senior Design Project
 * Designer: Eduardo Marquez
 * Email: marquez.edu@hotmail.com
 * Rev. No.: Version 1.0
 * Rev. Date: 4/24/17
 *
 * Purpose: This is a 32 bit register module. This will be used to hold data.
 *
 * Notes:
 *

****************************************************************************/
module reg32(clk, reset, ld, D, Q);

        input                   clk, reset, ld;
        input           [31:0] D;
        output          [31:0] Q;

        reg             [31:0] Q;


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

endmodule

`timescale 1ns / 1ps
```

```
/****************************** C E C S 4 4 0 ******************************
 *
 * File Name: reg32NoLD.v
 * Project: Senior Design Project
 * Designer: Eduardo Marquez
 * Email: marquez.edu@hotmail.com
 * Rev. No.: Version 1.0
 * Rev. Date: 4/24/17
 *
 * Purpose: This is a 32 bit register module. This will be used to hold data.
 *
 * Notes:
 *

****************************************************************************/
module reg32NoLD(clk, reset, D, Q);

      input                    clk, reset;
      input        [31:0] D;
      output       [31:0] Q;

      reg          [31:0] Q;


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

endmodule
```

MIPS Data Memory

```
`timescale 1ns / 1ps
/****************************** C E C S 4 4 0 ******************************
 *
 * File Name: Data_Memory.v
 * Project: Lab_Assignment_5
 * Designer: Eduardo Marquez
 * Email: marquez.edu@hotmail.com
 * Rev. No.: Version 1.0
 * Rev. Date: 3/13/17
 *
 * Purpose: This is a memory module.
 *
 * Notes:
 *

****************************************************************************/
module Data_Memory(clk, dm_cs, dm_wr, dm_rd, Addr, D_In, D_Out);

       input      clk, dm_cs, dm_wr, dm_rd;
       input [31:0] Addr, D_In;
       output [31:0] D_Out;

       // 4096 x 8 register array(memory), byte addressable memory
       reg    [7:0] M [0:4095]; //0 is first in list, 4095 is last

       // Tri state D_Out, high impedance when im_wr is high
       //read
       assign D_Out = (dm_cs && dm_rd && !dm_wr) ?
                  {M[Addr],M[Addr+1],M[Addr+2],M[Addr+3]} : 32'bz;

       //write
       always @(posedge clk) begin
            if(dm_cs && dm_wr)
            {M[Addr],M[Addr+1],M[Addr+2],M[Addr+3]} <=
                              {D_In[31:24],D_In[23:16],D_In[15:8],D_In[7:0]};
            else
            {M[Addr],M[Addr+1],M[Addr+2],M[Addr+3]} <=
                              {M[Addr],M[Addr+1],M[Addr+2],M[Addr+3]};
       end

endmodule
```

MIPS I/O Memory

```
`timescale 1ns / 1ps
/****************************** C E C S 4 4 0 ******************************
 *
 * File Name: IO_Memory.v
 * Project:
 * Designer: Eduardo Marquez
 * Email: marquez.edu@hotmail.com
 * Rev. No.: Version 1.0
 * Rev. Date: 4/19/17
 *
 * Purpose: This is a I/O memory module.
 *
 * Notes:
 *

****************************************************************************/
module IO_Memory(clk, io_cs, io_wr, io_rd, Addr, D_In, D_Out);

       input      clk, io_cs, io_wr, io_rd;
       input  [31:0] Addr, D_In;
       output [31:0] D_Out;

       // 4096 x 8 register array(memory), byte addressable memory
       reg    [7:0] M [0:4095]; //0 is first in list, 4095 is last

       // Tri state D_Out, high impedance when im_wr is high
       //read
       assign D_Out = (io_cs && io_rd && !io_wr) ?
                  {M[Addr],M[Addr+1],M[Addr+2],M[Addr+3]} : 32'bz;

       //write
       always @(posedge clk) begin
            if(io_cs && io_wr)
            {M[Addr],M[Addr+1],M[Addr+2],M[Addr+3]} <=
                  {D_In[31:24],D_In[23:16],D_In[15:8],D_In[7:0]};
            else
            {M[Addr],M[Addr+1],M[Addr+2],M[Addr+3]} <=
                  {M[Addr],M[Addr+1],M[Addr+2],M[Addr+3]};
       end

endmodule
```

Memory Data Files / Outputs for Design Verification

Verification 1:

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

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

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

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

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

```
R E G I S T E R ' S A F T E R B R E A K

t= 620.0 ns   $r00 = 00000000  ||  t= 620.0 ns   $r16 = xxxxxxxx
t= 620.0 ns   $r01 = 12345678  ||  t= 620.0 ns   $r17 = xxxxxxxx
t= 620.0 ns   $r02 = 87654321  ||  t= 620.0 ns   $r18 = xxxxxxxx
t= 620.0 ns   $r03 = 12345678  ||  t= 620.0 ns   $r19 = xxxxxxxx
t= 620.0 ns   $r04 = xxxxxxxx  ||  t= 620.0 ns   $r20 = xxxxxxxx
t= 620.0 ns   $r05 = xxxxxxxx  ||  t= 620.0 ns   $r21 = xxxxxxxx
t= 620.0 ns   $r06 = xxxxxxxx  ||  t= 620.0 ns   $r22 = xxxxxxxx
t= 620.0 ns   $r07 = xxxxxxxx  ||  t= 620.0 ns   $r23 = xxxxxxxx
t= 620.0 ns   $r08 = xxxxxxxx  ||  t= 620.0 ns   $r24 = xxxxxxxx
t= 620.0 ns   $r09 = xxxxxxxx  ||  t= 620.0 ns   $r25 = xxxxxxxx
t= 620.0 ns   $r10 = xxxxxxxx  ||  t= 620.0 ns   $r26 = xxxxxxxx
t= 620.0 ns   $r11 = xxxxxxxx  ||  t= 620.0 ns   $r27 = xxxxxxxx
t= 620.0 ns   $r12 = xxxxxxxx  ||  t= 620.0 ns   $r28 = xxxxxxxx
t= 620.0 ns   $r13 = 100100c0  ||  t= 620.0 ns   $r29 = 000003fc
t= 620.0 ns   $r14 = 00000000  ||  t= 620.0 ns   $r30 = xxxxxxxx
t= 620.0 ns   $r15 = 00000000  ||  t= 620.0 ns   $r31 = xxxxxxxx

time= 620.0 ns M[3F0]=xxxxxxxx

              DATA MEMORY                                    IO MEMORY
t= 620.0 ns   DM[000000c0] = 12345678  ||  t= 620.0 ns   IOM[000000c0] = xxxxxxxx
t= 620.0 ns   DM[000000c4] = xxxxxxxx  ||  t= 620.0 ns   IOM[000000c4] = xxxxxxxx
t= 620.0 ns   DM[000000c8] = xxxxxxxx  ||  t= 620.0 ns   IOM[000000c8] = xxxxxxxx
t= 620.0 ns   DM[000000cc] = xxxxxxxx  ||  t= 620.0 ns   IOM[000000cc] = xxxxxxxx
t= 620.0 ns   DM[000000d0] = xxxxxxxx  ||  t= 620.0 ns   IOM[000000d0] = xxxxxxxx
t= 620.0 ns   DM[000000d4] = xxxxxxxx  ||  t= 620.0 ns   IOM[000000d4] = xxxxxxxx
t= 620.0 ns   DM[000000d8] = xxxxxxxx  ||  t= 620.0 ns   IOM[000000d8] = xxxxxxxx
t= 620.0 ns   DM[000000dc] = xxxxxxxx  ||  t= 620.0 ns   IOM[000000dc] = xxxxxxxx
t= 620.0 ns   DM[000000e0] = xxxxxxxx  ||  t= 620.0 ns   IOM[000000e0] = xxxxxxxx
t= 620.0 ns   DM[000000e4] = xxxxxxxx  ||  t= 620.0 ns   IOM[000000e4] = xxxxxxxx
t= 620.0 ns   DM[000000e8] = xxxxxxxx  ||  t= 620.0 ns   IOM[000000e8] = xxxxxxxx
t= 620.0 ns   DM[000000ec] = xxxxxxxx  ||  t= 620.0 ns   IOM[000000ec] = xxxxxxxx
t= 620.0 ns   DM[000000f0] = xxxxxxxx  ||  t= 620.0 ns   IOM[000000f0] = xxxxxxxx
t= 620.0 ns   DM[000000f4] = xxxxxxxx  ||  t= 620.0 ns   IOM[000000f4] = xxxxxxxx
t= 620.0 ns   DM[000000f8] = xxxxxxxx  ||  t= 620.0 ns   IOM[000000f8] = xxxxxxxx
t= 620.0 ns   DM[000000fc] = xxxxxxxx  ||  t= 620.0 ns   IOM[000000fc] = xxxxxxxx
```

Verification 2:

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

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

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

```
R E G I S T E R ' S A F T E R B R E A K

t=3570.0 ns   $r00 = 00000000  ||  t=3570.0 ns   $r16 = xxxxxxxx
t=3570.0 ns   $r01 = 0000ffff  ||  t=3570.0 ns   $r17 = xxxxxxxx
t=3570.0 ns   $r02 = 00000000  ||  t=3570.0 ns   $r18 = xxxxxxxx
t=3570.0 ns   $r03 = xxxxxxxx  ||  t=3570.0 ns   $r19 = xxxxxxxx
t=3570.0 ns   $r04 = xxxxxxxx  ||  t=3570.0 ns   $r20 = xxxxxxxx
t=3570.0 ns   $r05 = xxxxxxxx  ||  t=3570.0 ns   $r21 = xxxxxxxx
t=3570.0 ns   $r06 = xxxxxxxx  ||  t=3570.0 ns   $r22 = xxxxxxxx
t=3570.0 ns   $r07 = xxxxxxxx  ||  t=3570.0 ns   $r23 = xxxxxxxx
t=3570.0 ns   $r08 = xxxxxxxx  ||  t=3570.0 ns   $r24 = xxxxxxxx
t=3570.0 ns   $r09 = xxxxxxxx  ||  t=3570.0 ns   $r25 = xxxxxxxx
t=3570.0 ns   $r10 = xxxxxxxx  ||  t=3570.0 ns   $r26 = xxxxxxxx
t=3570.0 ns   $r11 = xxxxxxxx  ||  t=3570.0 ns   $r27 = xxxxxxxx
t=3570.0 ns   $r12 = xxxxxxxx  ||  t=3570.0 ns   $r28 = xxxxxxxx
t=3570.0 ns   $r13 = xxxxxxxx  ||  t=3570.0 ns   $r29 = 000003fc
t=3570.0 ns   $r14 = 5a5a3c3c  ||  t=3570.0 ns   $r30 = xxxxxxxx
t=3570.0 ns   $r15 = 10010100  ||  t=3570.0 ns   $r31 = xxxxxxxx

time=3570.0 ns M[3F0]=xxxxxxxx

              DATA MEMORY                                      IO MEMORY
t=3570.0 ns   DM[000000c0] = 7fffffff  ||  t=3570.0 ns   IOM[000000c0] = xxxxxxxx
t=3570.0 ns   DM[000000c4] = 3fffffff  ||  t=3570.0 ns   IOM[000000c4] = xxxxxxxx
t=3570.0 ns   DM[000000c8] = 1fffffff  ||  t=3570.0 ns   IOM[000000c8] = xxxxxxxx
t=3570.0 ns   DM[000000cc] = 0fffffff  ||  t=3570.0 ns   IOM[000000cc] = xxxxxxxx
t=3570.0 ns   DM[000000d0] = 07ffffff  ||  t=3570.0 ns   IOM[000000d0] = xxxxxxxx
t=3570.0 ns   DM[000000d4] = 03ffffff  ||  t=3570.0 ns   IOM[000000d4] = xxxxxxxx
t=3570.0 ns   DM[000000d8] = 01ffffff  ||  t=3570.0 ns   IOM[000000d8] = xxxxxxxx
t=3570.0 ns   DM[000000dc] = 00ffffff  ||  t=3570.0 ns   IOM[000000dc] = xxxxxxxx
t=3570.0 ns   DM[000000e0] = 007fffff  ||  t=3570.0 ns   IOM[000000e0] = xxxxxxxx
t=3570.0 ns   DM[000000e4] = 003fffff  ||  t=3570.0 ns   IOM[000000e4] = xxxxxxxx
t=3570.0 ns   DM[000000e8] = 001fffff  ||  t=3570.0 ns   IOM[000000e8] = xxxxxxxx
t=3570.0 ns   DM[000000ec] = 000fffff  ||  t=3570.0 ns   IOM[000000ec] = xxxxxxxx
t=3570.0 ns   DM[000000f0] = 0007ffff  ||  t=3570.0 ns   IOM[000000f0] = xxxxxxxx
t=3570.0 ns   DM[000000f4] = 0003ffff  ||  t=3570.0 ns   IOM[000000f4] = xxxxxxxx
t=3570.0 ns   DM[000000f8] = 0001ffff  ||  t=3570.0 ns   IOM[000000f8] = xxxxxxxx
t=3570.0 ns   DM[000000fc] = 0000ffff  ||  t=3570.0 ns   IOM[000000fc] = xxxxxxxx
```

Verification 3:

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

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

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

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

```
R E G I S T E R ' S A F T E R B R E A K

t=3570.0 ns   $r00 = 00000000  ||  t=3570.0 ns   $r16 = xxxxxxxx
t=3570.0 ns   $r01 = ffff8000  ||  t=3570.0 ns   $r17 = xxxxxxxx
t=3570.0 ns   $r02 = 00000000  ||  t=3570.0 ns   $r18 = xxxxxxxx
t=3570.0 ns   $r03 = xxxxxxxx  ||  t=3570.0 ns   $r19 = xxxxxxxx
t=3570.0 ns   $r04 = xxxxxxxx  ||  t=3570.0 ns   $r20 = xxxxxxxx
t=3570.0 ns   $r05 = xxxxxxxx  ||  t=3570.0 ns   $r21 = xxxxxxxx
t=3570.0 ns   $r06 = xxxxxxxx  ||  t=3570.0 ns   $r22 = xxxxxxxx
t=3570.0 ns   $r07 = xxxxxxxx  ||  t=3570.0 ns   $r23 = xxxxxxxx
t=3570.0 ns   $r08 = xxxxxxxx  ||  t=3570.0 ns   $r24 = xxxxxxxx
t=3570.0 ns   $r09 = xxxxxxxx  ||  t=3570.0 ns   $r25 = xxxxxxxx
t=3570.0 ns   $r10 = xxxxxxxx  ||  t=3570.0 ns   $r26 = xxxxxxxx
t=3570.0 ns   $r11 = xxxxxxxx  ||  t=3570.0 ns   $r27 = xxxxxxxx
t=3570.0 ns   $r12 = xxxxxxxx  ||  t=3570.0 ns   $r28 = xxxxxxxx
t=3570.0 ns   $r13 = xxxxxxxx  ||  t=3570.0 ns   $r29 = 000003fc
t=3570.0 ns   $r14 = 5a5a3c3c  ||  t=3570.0 ns   $r30 = xxxxxxxx
t=3570.0 ns   $r15 = 10010100  ||  t=3570.0 ns   $r31 = xxxxxxxx

time=3570.0 ns M[3F0]=xxxxxxxx

              DATA MEMORY                                          IO MEMORY
t=3570.0 ns   DM[000000c0] = c0007fff  ||  t=3570.0 ns   IOM[000000c0] = xxxxxxxx
t=3570.0 ns   DM[000000c4] = e0003fff  ||  t=3570.0 ns   IOM[000000c4] = xxxxxxxx
t=3570.0 ns   DM[000000c8] = f0001fff  ||  t=3570.0 ns   IOM[000000c8] = xxxxxxxx
t=3570.0 ns   DM[000000cc] = f8000fff  ||  t=3570.0 ns   IOM[000000cc] = xxxxxxxx
t=3570.0 ns   DM[000000d0] = fc0007ff  ||  t=3570.0 ns   IOM[000000d0] = xxxxxxxx
t=3570.0 ns   DM[000000d4] = fe0003ff  ||  t=3570.0 ns   IOM[000000d4] = xxxxxxxx
t=3570.0 ns   DM[000000d8] = ff0001ff  ||  t=3570.0 ns   IOM[000000d8] = xxxxxxxx
t=3570.0 ns   DM[000000dc] = ff8000ff  ||  t=3570.0 ns   IOM[000000dc] = xxxxxxxx
t=3570.0 ns   DM[000000e0] = ffc0007f  ||  t=3570.0 ns   IOM[000000e0] = xxxxxxxx
t=3570.0 ns   DM[000000e4] = ffe0003f  ||  t=3570.0 ns   IOM[000000e4] = xxxxxxxx
t=3570.0 ns   DM[000000e8] = fff0001f  ||  t=3570.0 ns   IOM[000000e8] = xxxxxxxx
t=3570.0 ns   DM[000000ec] = fff8000f  ||  t=3570.0 ns   IOM[000000ec] = xxxxxxxx
t=3570.0 ns   DM[000000f0] = fffc0007  ||  t=3570.0 ns   IOM[000000f0] = xxxxxxxx
t=3570.0 ns   DM[000000f4] = fffe0003  ||  t=3570.0 ns   IOM[000000f4] = xxxxxxxx
t=3570.0 ns   DM[000000f8] = ffff0001  ||  t=3570.0 ns   IOM[000000f8] = xxxxxxxx
t=3570.0 ns   DM[000000fc] = ffff8000  ||  t=3570.0 ns   IOM[000000fc] = xxxxxxxx
```

Verification 4:

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

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

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

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

```
R E G I S T E R ' S A F T E R B R E A K

t=4210.0 ns   $r00 = 00000000  ||  t=4210.0 ns   $r16 = xxxxxxxx
t=4210.0 ns   $r01 = ffff0000  ||  t=4210.0 ns   $r17 = xxxxxxxx
t=4210.0 ns   $r02 = 00000000  ||  t=4210.0 ns   $r18 = xxxxxxxx
t=4210.0 ns   $r03 = 00000000  ||  t=4210.0 ns   $r19 = xxxxxxxx
t=4210.0 ns   $r04 = xxxxxxxx  ||  t=4210.0 ns   $r20 = xxxxxxxx
t=4210.0 ns   $r05 = xxxxxxxx  ||  t=4210.0 ns   $r21 = xxxxxxxx
t=4210.0 ns   $r06 = xxxxxxxx  ||  t=4210.0 ns   $r22 = xxxxxxxx
t=4210.0 ns   $r07 = xxxxxxxx  ||  t=4210.0 ns   $r23 = xxxxxxxx
t=4210.0 ns   $r08 = xxxxxxxx  ||  t=4210.0 ns   $r24 = xxxxxxxx
t=4210.0 ns   $r09 = xxxxxxxx  ||  t=4210.0 ns   $r25 = xxxxxxxx
t=4210.0 ns   $r10 = xxxxxxxx  ||  t=4210.0 ns   $r26 = xxxxxxxx
t=4210.0 ns   $r11 = xxxxxxxx  ||  t=4210.0 ns   $r27 = xxxxxxxx
t=4210.0 ns   $r12 = xxxxxxxx  ||  t=4210.0 ns   $r28 = xxxxxxxx
t=4210.0 ns   $r13 = xxxxxxxx  ||  t=4210.0 ns   $r29 = 000003fc
t=4210.0 ns   $r14 = 5a5a3c3c  ||  t=4210.0 ns   $r30 = xxxxxxxx
t=4210.0 ns   $r15 = 10010100  ||  t=4210.0 ns   $r31 = xxxxxxxx

time=4210.0 ns M[3F0]=xxxxxxxx

              DATA MEMORY                                  IO MEMORY
t=4210.0 ns   DM[000000c0] = fffffffe  ||  t=4210.0 ns   IOM[000000c0] = xxxxxxxx
t=4210.0 ns   DM[000000c4] = fffffffc  ||  t=4210.0 ns   IOM[000000c4] = xxxxxxxx
t=4210.0 ns   DM[000000c8] = fffffff8  ||  t=4210.0 ns   IOM[000000c8] = xxxxxxxx
t=4210.0 ns   DM[000000cc] = fffffff0  ||  t=4210.0 ns   IOM[000000cc] = xxxxxxxx
t=4210.0 ns   DM[000000d0] = ffffffe0  ||  t=4210.0 ns   IOM[000000d0] = xxxxxxxx
t=4210.0 ns   DM[000000d4] = ffffffc0  ||  t=4210.0 ns   IOM[000000d4] = xxxxxxxx
t=4210.0 ns   DM[000000d8] = ffffff80  ||  t=4210.0 ns   IOM[000000d8] = xxxxxxxx
t=4210.0 ns   DM[000000dc] = ffffff00  ||  t=4210.0 ns   IOM[000000dc] = xxxxxxxx
t=4210.0 ns   DM[000000e0] = fffffe00  ||  t=4210.0 ns   IOM[000000e0] = xxxxxxxx
t=4210.0 ns   DM[000000e4] = fffffc00  ||  t=4210.0 ns   IOM[000000e4] = xxxxxxxx
t=4210.0 ns   DM[000000e8] = fffff800  ||  t=4210.0 ns   IOM[000000e8] = xxxxxxxx
t=4210.0 ns   DM[000000ec] = fffff000  ||  t=4210.0 ns   IOM[000000ec] = xxxxxxxx
t=4210.0 ns   DM[000000f0] = ffffe000  ||  t=4210.0 ns   IOM[000000f0] = xxxxxxxx
t=4210.0 ns   DM[000000f4] = ffffc000  ||  t=4210.0 ns   IOM[000000f4] = xxxxxxxx
t=4210.0 ns   DM[000000f8] = ffff8000  ||  t=4210.0 ns   IOM[000000f8] = xxxxxxxx
t=4210.0 ns   DM[000000fc] = ffff0000  ||  t=4210.0 ns   IOM[000000fc] = xxxxxxxx
```

Verification 5:

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

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

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

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

```
R E G I S T E R ' S A F T E R B R E A K

t=4210.0 ns   $r00 = 00000000  ||  t=4210.0 ns   $r16 = xxxxxxxx
t=4210.0 ns   $r01 = ffff0000  ||  t=4210.0 ns   $r17 = xxxxxxxx
t=4210.0 ns   $r02 = 00000000  ||  t=4210.0 ns   $r18 = xxxxxxxx
t=4210.0 ns   $r03 = 00000000  ||  t=4210.0 ns   $r19 = xxxxxxxx
t=4210.0 ns   $r04 = xxxxxxxx  ||  t=4210.0 ns   $r20 = xxxxxxxx
t=4210.0 ns   $r05 = xxxxxxxx  ||  t=4210.0 ns   $r21 = xxxxxxxx
t=4210.0 ns   $r06 = xxxxxxxx  ||  t=4210.0 ns   $r22 = xxxxxxxx
t=4210.0 ns   $r07 = xxxxxxxx  ||  t=4210.0 ns   $r23 = xxxxxxxx
t=4210.0 ns   $r08 = xxxxxxxx  ||  t=4210.0 ns   $r24 = xxxxxxxx
t=4210.0 ns   $r09 = xxxxxxxx  ||  t=4210.0 ns   $r25 = xxxxxxxx
t=4210.0 ns   $r10 = xxxxxxxx  ||  t=4210.0 ns   $r26 = xxxxxxxx
t=4210.0 ns   $r11 = xxxxxxxx  ||  t=4210.0 ns   $r27 = xxxxxxxx
t=4210.0 ns   $r12 = xxxxxxxx  ||  t=4210.0 ns   $r28 = xxxxxxxx
t=4210.0 ns   $r13 = xxxxxxxx  ||  t=4210.0 ns   $r29 = 000003fc
t=4210.0 ns   $r14 = 5a5a3c3c  ||  t=4210.0 ns   $r30 = xxxxxxxx
t=4210.0 ns   $r15 = 10010100  ||  t=4210.0 ns   $r31 = xxxxxxxx

time=4210.0 ns M[3F0]=xxxxxxxx

              DATA MEMORY                                   IO MEMORY
t=4210.0 ns   DM[000000c0] = fffffffe  ||  t=4210.0 ns   IOM[000000c0] = xxxxxxxx
t=4210.0 ns   DM[000000c4] = fffffffc  ||  t=4210.0 ns   IOM[000000c4] = xxxxxxxx
t=4210.0 ns   DM[000000c8] = fffffff8  ||  t=4210.0 ns   IOM[000000c8] = xxxxxxxx
t=4210.0 ns   DM[000000cc] = fffffff0  ||  t=4210.0 ns   IOM[000000cc] = xxxxxxxx
t=4210.0 ns   DM[000000d0] = ffffffe0  ||  t=4210.0 ns   IOM[000000d0] = xxxxxxxx
t=4210.0 ns   DM[000000d4] = ffffffc0  ||  t=4210.0 ns   IOM[000000d4] = xxxxxxxx
t=4210.0 ns   DM[000000d8] = ffffff80  ||  t=4210.0 ns   IOM[000000d8] = xxxxxxxx
t=4210.0 ns   DM[000000dc] = ffffff00  ||  t=4210.0 ns   IOM[000000dc] = xxxxxxxx
t=4210.0 ns   DM[000000e0] = fffffe00  ||  t=4210.0 ns   IOM[000000e0] = xxxxxxxx
t=4210.0 ns   DM[000000e4] = fffffc00  ||  t=4210.0 ns   IOM[000000e4] = xxxxxxxx
t=4210.0 ns   DM[000000e8] = fffff800  ||  t=4210.0 ns   IOM[000000e8] = xxxxxxxx
t=4210.0 ns   DM[000000ec] = fffff000  ||  t=4210.0 ns   IOM[000000ec] = xxxxxxxx
t=4210.0 ns   DM[000000f0] = ffffe000  ||  t=4210.0 ns   IOM[000000f0] = xxxxxxxx
t=4210.0 ns   DM[000000f4] = ffffc000  ||  t=4210.0 ns   IOM[000000f4] = xxxxxxxx
t=4210.0 ns   DM[000000f8] = ffff8000  ||  t=4210.0 ns   IOM[000000f8] = xxxxxxxx
t=4210.0 ns   DM[000000fc] = ffff0000  ||  t=4210.0 ns   IOM[000000fc] = xxxxxxxx
```

Verification 6:

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

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

```
R E G I S T E R ' S A F T E R B R E A K

t=4860.0 ns   $r00 = 00000000  ||  t=4860.0 ns   $r16 = xxxxxxxx
t=4860.0 ns   $r01 = 12345678  ||  t=4860.0 ns   $r17 = 000075cc
t=4860.0 ns   $r02 = 89abcdef  ||  t=4860.0 ns   $r18 = xxxxxxxx
t=4860.0 ns   $r03 = a5a5a5a5  ||  t=4860.0 ns   $r19 = xxxxxxxx
t=4860.0 ns   $r04 = 5a5a5a5a  ||  t=4860.0 ns   $r20 = xxxxxxxx
t=4860.0 ns   $r05 = 2468ace0  ||  t=4860.0 ns   $r21 = xxxxxxxx
t=4860.0 ns   $r06 = 13579bdf  ||  t=4860.0 ns   $r22 = xxxxxxxx
t=4860.0 ns   $r07 = 0f0f0f0f  ||  t=4860.0 ns   $r23 = xxxxxxxx
t=4860.0 ns   $r08 = f0f0f0f0  ||  t=4860.0 ns   $r24 = xxxxxxxx
t=4860.0 ns   $r09 = 00000009  ||  t=4860.0 ns   $r25 = xxxxxxxx
t=4860.0 ns   $r10 = 0000000a  ||  t=4860.0 ns   $r26 = xxxxxxxx
t=4860.0 ns   $r11 = 0000000b  ||  t=4860.0 ns   $r27 = xxxxxxxx
t=4860.0 ns   $r12 = 0000000c  ||  t=4860.0 ns   $r28 = xxxxxxxx
t=4860.0 ns   $r13 = 00000000  ||  t=4860.0 ns   $r29 = 000003fc
t=4860.0 ns   $r14 = 10010100  ||  t=4860.0 ns   $r30 = xxxxxxxx
t=4860.0 ns   $r15 = 10010040  ||  t=4860.0 ns   $r31 = xxxxxxxx

time=4860.0 ns M[3F0]=xxxxxxxx

              DATA MEMORY                                      IO MEMORY
t=4860.0 ns   DM[000000c0] = c3c3c3c3  ||  t=4860.0 ns   IOM[000000c0] = xxxxxxxx
t=4860.0 ns   DM[000000c4] = 12345678  ||  t=4860.0 ns   IOM[000000c4] = xxxxxxxx
t=4860.0 ns   DM[000000c8] = 89abcdef  ||  t=4860.0 ns   IOM[000000c8] = xxxxxxxx
t=4860.0 ns   DM[000000cc] = a5a5a5a5  ||  t=4860.0 ns   IOM[000000cc] = xxxxxxxx
t=4860.0 ns   DM[000000d0] = 5a5a5a5a  ||  t=4860.0 ns   IOM[000000d0] = xxxxxxxx
t=4860.0 ns   DM[000000d4] = 2468ace0  ||  t=4860.0 ns   IOM[000000d4] = xxxxxxxx
t=4860.0 ns   DM[000000d8] = 13579bdf  ||  t=4860.0 ns   IOM[000000d8] = xxxxxxxx
t=4860.0 ns   DM[000000dc] = 0f0f0f0f  ||  t=4860.0 ns   IOM[000000dc] = xxxxxxxx
t=4860.0 ns   DM[000000e0] = f0f0f0f0  ||  t=4860.0 ns   IOM[000000e0] = xxxxxxxx
t=4860.0 ns   DM[000000e4] = 00000009  ||  t=4860.0 ns   IOM[000000e4] = xxxxxxxx
t=4860.0 ns   DM[000000e8] = 0000000a  ||  t=4860.0 ns   IOM[000000e8] = xxxxxxxx
t=4860.0 ns   DM[000000ec] = 0000000b  ||  t=4860.0 ns   IOM[000000ec] = xxxxxxxx
t=4860.0 ns   DM[000000f0] = 0000000c  ||  t=4860.0 ns   IOM[000000f0] = xxxxxxxx
t=4860.0 ns   DM[000000f4] = 0000000d  ||  t=4860.0 ns   IOM[000000f4] = xxxxxxxx
t=4860.0 ns   DM[000000f8] = fffffff8  ||  t=4860.0 ns   IOM[000000f8] = xxxxxxxx
t=4860.0 ns   DM[000000fc] = 000075cc  ||  t=4860.0 ns   IOM[000000fc] = xxxxxxxx
```

Verification 7:

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

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

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

```
R E G I S T E R ' S A F T E R B R E A K

t=5010.0 ns   $r0 = 00000000  ||  t=5010.0 ns   $r16 = xxxxxxxx
t=5010.0 ns   $r1 = 12345678  ||  t=5010.0 ns   $r17 = 000075cc
t=5010.0 ns   $r2 = 89abcdef  ||  t=5010.0 ns   $r18 = xxxxxxxx
t=5010.0 ns   $r3 = a5a5a5a5  ||  t=5010.0 ns   $r19 = xxxxxxxx
t=5010.0 ns   $r4 = 5a5a5a5a  ||  t=5010.0 ns   $r20 = xxxxxxxx
t=5010.0 ns   $r5 = 2468ace0  ||  t=5010.0 ns   $r21 = xxxxxxxx
t=5010.0 ns   $r6 = 13579bdf  ||  t=5010.0 ns   $r22 = xxxxxxxx
t=5010.0 ns   $r7 = 0f0f0f0f  ||  t=5010.0 ns   $r23 = xxxxxxxx
t=5010.0 ns   $r8 = f0f0f0f0  ||  t=5010.0 ns   $r24 = xxxxxxxx
t=5010.0 ns   $r9 = 00000009  ||  t=5010.0 ns   $r25 = xxxxxxxx
t=5010.0 ns   $r10 = 0000000a  ||  t=5010.0 ns   $r26 = xxxxxxxx
t=5010.0 ns   $r11 = 0000000b  ||  t=5010.0 ns   $r27 = xxxxxxxx
t=5010.0 ns   $r12 = 0000000c  ||  t=5010.0 ns   $r28 = xxxxxxxx
t=5010.0 ns   $r13 = 00000000  ||  t=5010.0 ns   $r29 = 000003fc
t=5010.0 ns   $r14 = 10010100  ||  t=5010.0 ns   $r30 = xxxxxxxx
t=5010.0 ns   $r15 = ffffffff  ||  t=5010.0 ns   $r31 = 00000048

time=5010.0 ns M[3F0]=xxxxxxxx

             DATA MEMORY                                  IO MEMORY
t=5010.0 ns   DM[000000c0] = c3c3c3c3  ||  t=5010.0 ns   IOM[000000c0] = xxxxxxxx
t=5010.0 ns   DM[000000c4] = 12345678  ||  t=5010.0 ns   IOM[000000c4] = xxxxxxxx
t=5010.0 ns   DM[000000c8] = 89abcdef  ||  t=5010.0 ns   IOM[000000c8] = xxxxxxxx
t=5010.0 ns   DM[000000cc] = a5a5a5a5  ||  t=5010.0 ns   IOM[000000cc] = xxxxxxxx
t=5010.0 ns   DM[000000d0] = 5a5a5a5a  ||  t=5010.0 ns   IOM[000000d0] = xxxxxxxx
t=5010.0 ns   DM[000000d4] = 2468ace0  ||  t=5010.0 ns   IOM[000000d4] = xxxxxxxx
t=5010.0 ns   DM[000000d8] = 13579bdf  ||  t=5010.0 ns   IOM[000000d8] = xxxxxxxx
t=5010.0 ns   DM[000000dc] = 0f0f0f0f  ||  t=5010.0 ns   IOM[000000dc] = xxxxxxxx
t=5010.0 ns   DM[000000e0] = f0f0f0f0  ||  t=5010.0 ns   IOM[000000e0] = xxxxxxxx
t=5010.0 ns   DM[000000e4] = 00000009  ||  t=5010.0 ns   IOM[000000e4] = xxxxxxxx
t=5010.0 ns   DM[000000e8] = 0000000a  ||  t=5010.0 ns   IOM[000000e8] = xxxxxxxx
t=5010.0 ns   DM[000000ec] = 0000000b  ||  t=5010.0 ns   IOM[000000ec] = xxxxxxxx
t=5010.0 ns   DM[000000f0] = 0000000c  ||  t=5010.0 ns   IOM[000000f0] = xxxxxxxx
t=5010.0 ns   DM[000000f4] = 0000000d  ||  t=5010.0 ns   IOM[000000f4] = xxxxxxxx
t=5010.0 ns   DM[000000f8] = fffffff8  ||  t=5010.0 ns   IOM[000000f8] = xxxxxxxx
t=5010.0 ns   DM[000000fc] = 000075cc  ||  t=5010.0 ns   IOM[000000fc] = xxxxxxxx
```

Verification 8:

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

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

```
3c 0e ff ff  // fail4:     lui  $14, 0xFFFF
35 ce ff fa  //            ori  $14, 0xFFFA        # $r14 <-- 0xFFFFFFFA
(Fail flag 4)
00 00 00 0d  //            break
```

```
R E G I S T E R ' S A F T E R B R E A K

t=1110.0 ns   $r00 = 00000000  ||  t=1110.0 ns   $r16 = xxxxxxxx
t=1110.0 ns   $r01 = 00000019  ||  t=1110.0 ns   $r17 = xxxxxxxx
t=1110.0 ns   $r02 = 000003e8  ||  t=1110.0 ns   $r18 = xxxxxxxx
t=1110.0 ns   $r03 = ffffffe7  ||  t=1110.0 ns   $r19 = xxxxxxxx
t=1110.0 ns   $r04 = fffffc18  ||  t=1110.0 ns   $r20 = xxxxxxxx
t=1110.0 ns   $r05 = 000061a8  ||  t=1110.0 ns   $r21 = xxxxxxxx
t=1110.0 ns   $r06 = ffff9e58  ||  t=1110.0 ns   $r22 = xxxxxxxx
t=1110.0 ns   $r07 = ffffffff  ||  t=1110.0 ns   $r23 = xxxxxxxx
t=1110.0 ns   $r08 = 000061a8  ||  t=1110.0 ns   $r24 = xxxxxxxx
t=1110.0 ns   $r09 = ffff9e58  ||  t=1110.0 ns   $r25 = xxxxxxxx
t=1110.0 ns   $r10 = ffffffff  ||  t=1110.0 ns   $r26 = xxxxxxxx
t=1110.0 ns   $r11 = ffff9e58  ||  t=1110.0 ns   $r27 = xxxxxxxx
t=1110.0 ns   $r12 = ffffffff  ||  t=1110.0 ns   $r28 = xxxxxxxx
t=1110.0 ns   $r13 = 000061a8  ||  t=1110.0 ns   $r29 = 000003fc
t=1110.0 ns   $r14 = 00000000  ||  t=1110.0 ns   $r30 = xxxxxxxx
t=1110.0 ns   $r15 = 10010000  ||  t=1110.0 ns   $r31 = xxxxxxxx

time=1110.0 ns M[3F0]=xxxxxxxx

              DATA MEMORY                                  IO MEMORY
t=1110.0 ns   DM[000000c0] = xxxxxxxx  ||  t=1110.0 ns   IOM[000000c0] = xxxxxxxx
t=1110.0 ns   DM[000000c4] = xxxxxxxx  ||  t=1110.0 ns   IOM[000000c4] = xxxxxxxx
t=1110.0 ns   DM[000000c8] = xxxxxxxx  ||  t=1110.0 ns   IOM[000000c8] = xxxxxxxx
t=1110.0 ns   DM[000000cc] = xxxxxxxx  ||  t=1110.0 ns   IOM[000000cc] = xxxxxxxx
t=1110.0 ns   DM[000000d0] = xxxxxxxx  ||  t=1110.0 ns   IOM[000000d0] = xxxxxxxx
t=1110.0 ns   DM[000000d4] = xxxxxxxx  ||  t=1110.0 ns   IOM[000000d4] = xxxxxxxx
t=1110.0 ns   DM[000000d8] = xxxxxxxx  ||  t=1110.0 ns   IOM[000000d8] = xxxxxxxx
t=1110.0 ns   DM[000000dc] = xxxxxxxx  ||  t=1110.0 ns   IOM[000000dc] = xxxxxxxx
t=1110.0 ns   DM[000000e0] = xxxxxxxx  ||  t=1110.0 ns   IOM[000000e0] = xxxxxxxx
t=1110.0 ns   DM[000000e4] = xxxxxxxx  ||  t=1110.0 ns   IOM[000000e4] = xxxxxxxx
t=1110.0 ns   DM[000000e8] = xxxxxxxx  ||  t=1110.0 ns   IOM[000000e8] = xxxxxxxx
t=1110.0 ns   DM[000000ec] = xxxxxxxx  ||  t=1110.0 ns   IOM[000000ec] = xxxxxxxx
t=1110.0 ns   DM[000000f0] = xxxxxxxx  ||  t=1110.0 ns   IOM[000000f0] = xxxxxxxx
t=1110.0 ns   DM[000000f4] = xxxxxxxx  ||  t=1110.0 ns   IOM[000000f4] = xxxxxxxx
t=1110.0 ns   DM[000000f8] = xxxxxxxx  ||  t=1110.0 ns   IOM[000000f8] = xxxxxxxx
t=1110.0 ns   DM[000000fc] = xxxxxxxx  ||  t=1110.0 ns   IOM[000000fc] = xxxxxxxx
```

Verification 9:

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

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

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

00 22 18 2a  // slt_tests: slt  $03, $01, $02      # for signed# r01 < r02
14 60 00 02  //            bne  $03, $00, slt1     #   thus, we should branch
20 0e ff ff  //            addi $14, $00, -1       # fail flag1 r14 <--
FFFF_FFFF
```

```
00 00 00 0d  //            break
20 04 00 c0  // slt1:      addi $04, $00, 0xC0   # pass flag1 M[C0] <-- C0
ad e4 00 00  //            sw   $04, 0x00($15)

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

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

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

```
R E G I S T E R ' S A F T E R B R E A K

t=1730.0 ns   $r00 = 00000000  ||  t=1730.0 ns   $r16 = xxxxxxxx
t=1730.0 ns   $r01 = ffffff8a  ||  t=1730.0 ns   $r17 = xxxxxxxx
t=1730.0 ns   $r02 = 0000008a  ||  t=1730.0 ns   $r18 = xxxxxxxx
t=1730.0 ns   $r03 = 00000000  ||  t=1730.0 ns   $r19 = xxxxxxxx
t=1730.0 ns   $r04 = 000000c0  ||  t=1730.0 ns   $r20 = xxxxxxxx
t=1730.0 ns   $r05 = 000000c4  ||  t=1730.0 ns   $r21 = xxxxxxxx
t=1730.0 ns   $r06 = 000000c8  ||  t=1730.0 ns   $r22 = xxxxxxxx
t=1730.0 ns   $r07 = 000000cc  ||  t=1730.0 ns   $r23 = xxxxxxxx
t=1730.0 ns   $r08 = effeff35  ||  t=1730.0 ns   $r24 = xxxxxxxx
t=1730.0 ns   $r09 = effeff35  ||  t=1730.0 ns   $r25 = xxxxxxxx
t=1730.0 ns   $r10 = ffffffff  ||  t=1730.0 ns   $r26 = xxxxxxxx
t=1730.0 ns   $r11 = ffffffff  ||  t=1730.0 ns   $r27 = xxxxxxxx
t=1730.0 ns   $r12 = 88778877  ||  t=1730.0 ns   $r28 = xxxxxxxx
t=1730.0 ns   $r13 = 77887788  ||  t=1730.0 ns   $r29 = 000003fc
t=1730.0 ns   $r14 = 00000000  ||  t=1730.0 ns   $r30 = xxxxxxxx
t=1730.0 ns   $r15 = 100100c0  ||  t=1730.0 ns   $r31 = 00000014

time=1730.0 ns M[3F0]=xxxxxxxx

              DATA MEMORY                                    IO MEMORY
t=1730.0 ns   DM[000000c0] = 000000c0  ||  t=1730.0 ns   IOM[000000c0] = xxxxxxxx
t=1730.0 ns   DM[000000c4] = 000000c4  ||  t=1730.0 ns   IOM[000000c4] = xxxxxxxx
t=1730.0 ns   DM[000000c8] = 000000c8  ||  t=1730.0 ns   IOM[000000c8] = xxxxxxxx
t=1730.0 ns   DM[000000cc] = 000000cc  ||  t=1730.0 ns   IOM[000000cc] = xxxxxxxx
t=1730.0 ns   DM[000000d0] = effeff35  ||  t=1730.0 ns   IOM[000000d0] = xxxxxxxx
t=1730.0 ns   DM[000000d4] = xxxxxxxx  ||  t=1730.0 ns   IOM[000000d4] = xxxxxxxx
t=1730.0 ns   DM[000000d8] = xxxxxxxx  ||  t=1730.0 ns   IOM[000000d8] = xxxxxxxx
t=1730.0 ns   DM[000000dc] = xxxxxxxx  ||  t=1730.0 ns   IOM[000000dc] = xxxxxxxx
t=1730.0 ns   DM[000000e0] = xxxxxxxx  ||  t=1730.0 ns   IOM[000000e0] = xxxxxxxx
t=1730.0 ns   DM[000000e4] = xxxxxxxx  ||  t=1730.0 ns   IOM[000000e4] = xxxxxxxx
t=1730.0 ns   DM[000000e8] = xxxxxxxx  ||  t=1730.0 ns   IOM[000000e8] = xxxxxxxx
t=1730.0 ns   DM[000000ec] = xxxxxxxx  ||  t=1730.0 ns   IOM[000000ec] = xxxxxxxx
t=1730.0 ns   DM[000000f0] = xxxxxxxx  ||  t=1730.0 ns   IOM[000000f0] = xxxxxxxx
t=1730.0 ns   DM[000000f4] = xxxxxxxx  ||  t=1730.0 ns   IOM[000000f4] = xxxxxxxx
t=1730.0 ns   DM[000000f8] = xxxxxxxx  ||  t=1730.0 ns   IOM[000000f8] = xxxxxxxx
t=1730.0 ns   DM[000000fc] = xxxxxxxx  ||  t=1730.0 ns   IOM[000000fc] = xxxxxxxx
```

Verification 10:

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

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

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

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

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

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

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

```
R E G I S T E R ' S A F T E R B R E A K

t=1460.0 ns   $r0 = 00000000  ||  t=1460.0 ns   $r16 = xxxxxxxx
t=1460.0 ns   $r1 = 00040911  ||  t=1460.0 ns   $r17 = xxxxxxxx
t=1460.0 ns   $r2 = 000003e8  ||  t=1460.0 ns   $r18 = xxxxxxxx
t=1460.0 ns   $r3 = fffbf6ef  ||  t=1460.0 ns   $r19 = xxxxxxxx
t=1460.0 ns   $r4 = fffffc18  ||  t=1460.0 ns   $r20 = xxxxxxxx
t=1460.0 ns   $r5 = 00000108  ||  t=1460.0 ns   $r21 = xxxxxxxx
t=1460.0 ns   $r6 = 000001d1  ||  t=1460.0 ns   $r22 = xxxxxxxx
t=1460.0 ns   $r7 = fffffef8  ||  t=1460.0 ns   $r23 = xxxxxxxx
t=1460.0 ns   $r8 = fffffe2f  ||  t=1460.0 ns   $r24 = xxxxxxxx
t=1460.0 ns   $r9 = 00000108  ||  t=1460.0 ns   $r25 = xxxxxxxx
t=1460.0 ns   $r10 = fffffe2f  ||  t=1460.0 ns   $r26 = xxxxxxxx
t=1460.0 ns   $r11 = 00000000  ||  t=1460.0 ns   $r27 = xxxxxxxx
t=1460.0 ns   $r12 = 00000000  ||  t=1460.0 ns   $r28 = xxxxxxxx
t=1460.0 ns   $r13 = 00000000  ||  t=1460.0 ns   $r29 = 000003fc
t=1460.0 ns   $r14 = 00000000  ||  t=1460.0 ns   $r30 = xxxxxxxx
t=1460.0 ns   $r15 = 10010000  ||  t=1460.0 ns   $r31 = xxxxxxxx

time=1460.0 ns M[3F0]=xxxxxxxx

             DATA MEMORY                                  IO MEMORY
t=1460.0 ns   DM[000000c0] = xxxxxxxx  ||  t=1460.0 ns   IOM[000000c0] = xxxxxxxx
t=1460.0 ns   DM[000000c4] = xxxxxxxx  ||  t=1460.0 ns   IOM[000000c4] = xxxxxxxx
t=1460.0 ns   DM[000000c8] = xxxxxxxx  ||  t=1460.0 ns   IOM[000000c8] = xxxxxxxx
t=1460.0 ns   DM[000000cc] = xxxxxxxx  ||  t=1460.0 ns   IOM[000000cc] = xxxxxxxx
t=1460.0 ns   DM[000000d0] = xxxxxxxx  ||  t=1460.0 ns   IOM[000000d0] = xxxxxxxx
t=1460.0 ns   DM[000000d4] = xxxxxxxx  ||  t=1460.0 ns   IOM[000000d4] = xxxxxxxx
t=1460.0 ns   DM[000000d8] = xxxxxxxx  ||  t=1460.0 ns   IOM[000000d8] = xxxxxxxx
t=1460.0 ns   DM[000000dc] = xxxxxxxx  ||  t=1460.0 ns   IOM[000000dc] = xxxxxxxx
t=1460.0 ns   DM[000000e0] = xxxxxxxx  ||  t=1460.0 ns   IOM[000000e0] = xxxxxxxx
t=1460.0 ns   DM[000000e4] = xxxxxxxx  ||  t=1460.0 ns   IOM[000000e4] = xxxxxxxx
t=1460.0 ns   DM[000000e8] = xxxxxxxx  ||  t=1460.0 ns   IOM[000000e8] = xxxxxxxx
t=1460.0 ns   DM[000000ec] = xxxxxxxx  ||  t=1460.0 ns   IOM[000000ec] = xxxxxxxx
t=1460.0 ns   DM[000000f0] = xxxxxxxx  ||  t=1460.0 ns   IOM[000000f0] = xxxxxxxx
t=1460.0 ns   DM[000000f4] = xxxxxxxx  ||  t=1460.0 ns   IOM[000000f4] = xxxxxxxx
t=1460.0 ns   DM[000000f8] = xxxxxxxx  ||  t=1460.0 ns   IOM[000000f8] = xxxxxxxx
t=1460.0 ns   DM[000000fc] = xxxxxxxx  ||  t=1460.0 ns   IOM[000000fc] = xxxxxxxx
```

Verification 11:

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

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

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

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

2c 23 ff 89  //            sltiu  $03, $01, -119     # for unsigned# r01 !<
se(0xFF89)
```

```
10 60 00 02  //             beq    $03, $00, slt_p2   #   thus, we should
branch
20 0e ff fe  //             addi   $14, $00, -2       # fail flag2 r14 <--
FFFF_FFFE
00 00 00 0d  //             break
20 05 00 c4  // slt_p2:     addi   $05, $00, 0xC4     # pass flag2 M[C4] <--
C4
ad e5 00 04  //             sw     $05, 0x04($15)

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

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

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

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

```
R E G I S T E R ' S A F T E R B R E A K

t=1890.0 ns   $r00 = 00000000  ||  t=1890.0 ns   $r16 = xxxxxxxx
t=1890.0 ns   $r01 = ffffff8a  ||  t=1890.0 ns   $r17 = xxxxxxxx
t=1890.0 ns   $r02 = 0000008a  ||  t=1890.0 ns   $r18 = xxxxxxxx
t=1890.0 ns   $r03 = 00000000  ||  t=1890.0 ns   $r19 = xxxxxxxx
t=1890.0 ns   $r04 = 000000c0  ||  t=1890.0 ns   $r20 = xxxxxxxx
t=1890.0 ns   $r05 = 000000c4  ||  t=1890.0 ns   $r21 = xxxxxxxx
t=1890.0 ns   $r06 = 000000d4  ||  t=1890.0 ns   $r22 = xxxxxxxx
t=1890.0 ns   $r07 = 0000f0f0  ||  t=1890.0 ns   $r23 = xxxxxxxx
t=1890.0 ns   $r08 = 00000000  ||  t=1890.0 ns   $r24 = xxxxxxxx
t=1890.0 ns   $r09 = ffffffff  ||  t=1890.0 ns   $r25 = xxxxxxxx
t=1890.0 ns   $r10 = 0000f0f0  ||  t=1890.0 ns   $r26 = xxxxxxxx
t=1890.0 ns   $r11 = ffffffff  ||  t=1890.0 ns   $r27 = xxxxxxxx
t=1890.0 ns   $r12 = fffffaf5  ||  t=1890.0 ns   $r28 = xxxxxxxx
t=1890.0 ns   $r13 = ffff5555  ||  t=1890.0 ns   $r29 = 000003fc
t=1890.0 ns   $r14 = 00000000  ||  t=1890.0 ns   $r30 = xxxxxxxx
t=1890.0 ns   $r15 = 100100c0  ||  t=1890.0 ns   $r31 = 00000014

time=1890.0 ns M[3F0]=xxxxxxxx

              DATA MEMORY                                    IO MEMORY
t=1890.0 ns   DM[000000c0] = 000000c0  ||  t=1890.0 ns   IOM[000000c0] = xxxxxxxx
t=1890.0 ns   DM[000000c4] = 000000c4  ||  t=1890.0 ns   IOM[000000c4] = xxxxxxxx
t=1890.0 ns   DM[000000c8] = 000000c8  ||  t=1890.0 ns   IOM[000000c8] = xxxxxxxx
t=1890.0 ns   DM[000000cc] = 000000cc  ||  t=1890.0 ns   IOM[000000cc] = xxxxxxxx
t=1890.0 ns   DM[000000d0] = 000000d0  ||  t=1890.0 ns   IOM[000000d0] = xxxxxxxx
t=1890.0 ns   DM[000000d4] = 000000d4  ||  t=1890.0 ns   IOM[000000d4] = xxxxxxxx
t=1890.0 ns   DM[000000d8] = ffffff8a  ||  t=1890.0 ns   IOM[000000d8] = xxxxxxxx
t=1890.0 ns   DM[000000dc] = xxxxxxxx  ||  t=1890.0 ns   IOM[000000dc] = xxxxxxxx
t=1890.0 ns   DM[000000e0] = xxxxxxxx  ||  t=1890.0 ns   IOM[000000e0] = xxxxxxxx
t=1890.0 ns   DM[000000e4] = xxxxxxxx  ||  t=1890.0 ns   IOM[000000e4] = xxxxxxxx
t=1890.0 ns   DM[000000e8] = xxxxxxxx  ||  t=1890.0 ns   IOM[000000e8] = xxxxxxxx
t=1890.0 ns   DM[000000ec] = xxxxxxxx  ||  t=1890.0 ns   IOM[000000ec] = xxxxxxxx
t=1890.0 ns   DM[000000f0] = xxxxxxxx  ||  t=1890.0 ns   IOM[000000f0] = xxxxxxxx
t=1890.0 ns   DM[000000f4] = xxxxxxxx  ||  t=1890.0 ns   IOM[000000f4] = xxxxxxxx
t=1890.0 ns   DM[000000f8] = xxxxxxxx  ||  t=1890.0 ns   IOM[000000f8] = xxxxxxxx
t=1890.0 ns   DM[000000fc] = xxxxxxxx  ||  t=1890.0 ns   IOM[000000fc] = xxxxxxxx
```

Verification 12:

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

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

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

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

```
20 0e 00 00  // bgtz_p3:   addi $14, $00, 0        # set $r14 to 0000_0000
03 e0 00 08  //            jr   $31                # return from subroutine


R E G I S T E R ' S A F T E R B R E A K

t=1220.0 ns   $r00 = 00000000  ||  t=1220.0 ns   $r16 = xxxxxxxx
t=1220.0 ns   $r01 = ffffff8a  ||  t=1220.0 ns   $r17 = xxxxxxxx
t=1220.0 ns   $r02 = 0000008a  ||  t=1220.0 ns   $r18 = xxxxxxxx
t=1220.0 ns   $r03 = 000000c0  ||  t=1220.0 ns   $r19 = xxxxxxxx
t=1220.0 ns   $r04 = 000000c4  ||  t=1220.0 ns   $r20 = xxxxxxxx
t=1220.0 ns   $r05 = 000000c8  ||  t=1220.0 ns   $r21 = xxxxxxxx
t=1220.0 ns   $r06 = 000000cc  ||  t=1220.0 ns   $r22 = xxxxxxxx
t=1220.0 ns   $r07 = 000000d0  ||  t=1220.0 ns   $r23 = xxxxxxxx
t=1220.0 ns   $r08 = 000000d4  ||  t=1220.0 ns   $r24 = xxxxxxxx
t=1220.0 ns   $r09 = xxxxxxxx  ||  t=1220.0 ns   $r25 = xxxxxxxx
t=1220.0 ns   $r10 = xxxxxxxx  ||  t=1220.0 ns   $r26 = xxxxxxxx
t=1220.0 ns   $r11 = xxxxxxxx  ||  t=1220.0 ns   $r27 = xxxxxxxx
t=1220.0 ns   $r12 = xxxxxxxx  ||  t=1220.0 ns   $r28 = xxxxxxxx
t=1220.0 ns   $r13 = xxxxxxxx  ||  t=1220.0 ns   $r29 = 000003fc
t=1220.0 ns   $r14 = 00000000  ||  t=1220.0 ns   $r30 = xxxxxxxx
t=1220.0 ns   $r15 = 100100c0  ||  t=1220.0 ns   $r31 = 00000014

time=1220.0 ns M[3F0]=xxxxxxxx

              DATA MEMORY                                     IO MEMORY
t=1220.0 ns   DM[000000c0] = 000000c0  ||  t=1220.0 ns   IOM[000000c0] = xxxxxxxx
t=1220.0 ns   DM[000000c4] = 000000c4  ||  t=1220.0 ns   IOM[000000c4] = xxxxxxxx
t=1220.0 ns   DM[000000c8] = 000000c8  ||  t=1220.0 ns   IOM[000000c8] = xxxxxxxx
t=1220.0 ns   DM[000000cc] = 000000cc  ||  t=1220.0 ns   IOM[000000cc] = xxxxxxxx
t=1220.0 ns   DM[000000d0] = 000000d0  ||  t=1220.0 ns   IOM[000000d0] = xxxxxxxx
t=1220.0 ns   DM[000000d4] = 000000d4  ||  t=1220.0 ns   IOM[000000d4] = xxxxxxxx
t=1220.0 ns   DM[000000d8] = ffffff8a  ||  t=1220.0 ns   IOM[000000d8] = xxxxxxxx
t=1220.0 ns   DM[000000dc] = 0000008a  ||  t=1220.0 ns   IOM[000000dc] = xxxxxxxx
t=1220.0 ns   DM[000000e0] = xxxxxxxx  ||  t=1220.0 ns   IOM[000000e0] = xxxxxxxx
t=1220.0 ns   DM[000000e4] = xxxxxxxx  ||  t=1220.0 ns   IOM[000000e4] = xxxxxxxx
t=1220.0 ns   DM[000000e8] = xxxxxxxx  ||  t=1220.0 ns   IOM[000000e8] = xxxxxxxx
t=1220.0 ns   DM[000000ec] = xxxxxxxx  ||  t=1220.0 ns   IOM[000000ec] = xxxxxxxx
t=1220.0 ns   DM[000000f0] = xxxxxxxx  ||  t=1220.0 ns   IOM[000000f0] = xxxxxxxx
t=1220.0 ns   DM[000000f4] = xxxxxxxx  ||  t=1220.0 ns   IOM[000000f4] = xxxxxxxx
t=1220.0 ns   DM[000000f8] = xxxxxxxx  ||  t=1220.0 ns   IOM[000000f8] = xxxxxxxx
t=1220.0 ns   DM[000000fc] = xxxxxxxx  ||  t=1220.0 ns   IOM[000000fc] = xxxxxxxx
```

Verification 13:

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

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

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

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

76 11 00 00  // out_IO:    output $17, 0($16)      # output  [R16], R17
00 11 88 43  //            sra    $17, $17, 1      # change the pattern
22 10 00 04  //            addi   $16, $16, 4      # increment the memory
pointer 4 bytes
22 52 ff ff  //            addi   $18, $18, -1     # decrement the loop
counter
```

```
16 40 ff fb  //            bne    $18, $00, out_IO #  and jmp to top if not
finished

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

```
R E G I S T E R ' S A F T E R B R E A K

t=4950.0 ns   $r00 = 00000000  ||  t=4950.0 ns   $r16 = 100100c0
t=4950.0 ns   $r01 = 12345678  ||  t=4950.0 ns   $r17 = ffff8000
t=4950.0 ns   $r02 = 87654321  ||  t=4950.0 ns   $r18 = 00000000
t=4950.0 ns   $r03 = abcdef01  ||  t=4950.0 ns   $r19 = 8000ffff
t=4950.0 ns   $r04 = 01fedcba  ||  t=4950.0 ns   $r20 = c0007fff
t=4950.0 ns   $r05 = 5a5a5a5a  ||  t=4950.0 ns   $r21 = e0003fff
t=4950.0 ns   $r06 = ffffffff  ||  t=4950.0 ns   $r22 = f0001fff
t=4950.0 ns   $r07 = 100103f0  ||  t=4950.0 ns   $r23 = f8000fff
t=4950.0 ns   $r08 = fffffeff  ||  t=4950.0 ns   $r24 = fc0007ff
t=4950.0 ns   $r09 = fffffefe  ||  t=4950.0 ns   $r25 = xxxxxxxx
t=4950.0 ns   $r10 = fffffefd  ||  t=4950.0 ns   $r26 = xxxxxxxx
t=4950.0 ns   $r11 = fffffefc  ||  t=4950.0 ns   $r27 = xxxxxxxx
t=4950.0 ns   $r12 = fffffefb  ||  t=4950.0 ns   $r28 = xxxxxxxx
t=4950.0 ns   $r13 = fffffefa  ||  t=4950.0 ns   $r29 = 000003fc
t=4950.0 ns   $r14 = fffffef9  ||  t=4950.0 ns   $r30 = xxxxxxxx
t=4950.0 ns   $r15 = fffffef8  ||  t=4950.0 ns   $r31 = 00000048

time=4950.0 ns M[3F0]=fffffef8

              DATA MEMORY                                  IO MEMORY
t=4950.0 ns   DM[000000c0] = xxxxxxxx  ||  t=4950.0 ns   IOM[000000c0] = 8000ffff
t=4950.0 ns   DM[000000c4] = xxxxxxxx  ||  t=4950.0 ns   IOM[000000c4] = c0007fff
t=4950.0 ns   DM[000000c8] = xxxxxxxx  ||  t=4950.0 ns   IOM[000000c8] = e0003fff
t=4950.0 ns   DM[000000cc] = xxxxxxxx  ||  t=4950.0 ns   IOM[000000cc] = f0001fff
t=4950.0 ns   DM[000000d0] = xxxxxxxx  ||  t=4950.0 ns   IOM[000000d0] = f8000fff
t=4950.0 ns   DM[000000d4] = xxxxxxxx  ||  t=4950.0 ns   IOM[000000d4] = fc0007ff
t=4950.0 ns   DM[000000d8] = xxxxxxxx  ||  t=4950.0 ns   IOM[000000d8] = fe0003ff
t=4950.0 ns   DM[000000dc] = xxxxxxxx  ||  t=4950.0 ns   IOM[000000dc] = ff0001ff
t=4950.0 ns   DM[000000e0] = xxxxxxxx  ||  t=4950.0 ns   IOM[000000e0] = ff8000ff
t=4950.0 ns   DM[000000e4] = xxxxxxxx  ||  t=4950.0 ns   IOM[000000e4] = ffc0007f
t=4950.0 ns   DM[000000e8] = xxxxxxxx  ||  t=4950.0 ns   IOM[000000e8] = ffe0003f
t=4950.0 ns   DM[000000ec] = xxxxxxxx  ||  t=4950.0 ns   IOM[000000ec] = fff0001f
t=4950.0 ns   DM[000000f0] = xxxxxxxx  ||  t=4950.0 ns   IOM[000000f0] = fff8000f
t=4950.0 ns   DM[000000f4] = xxxxxxxx  ||  t=4950.0 ns   IOM[000000f4] = fffc0007
t=4950.0 ns   DM[000000f8] = xxxxxxxx  ||  t=4950.0 ns   IOM[000000f8] = fffe0003
t=4950.0 ns   DM[000000fc] = xxxxxxxx  ||  t=4950.0 ns   IOM[000000fc] = ffff0001
```

Verification 14:

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

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

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

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

76 11 00 00  // out_IO:    output $17, 0($16)      # output  [R16], R17
00 11 88 43  //            sra    $17, $17, 1      # change the pattern
22 10 00 04  //            addi   $16, $16, 4      # increment the memory
pointer 4 bytes
22 52 ff ff  //            addi   $18, $18, -1     # decrement the loop
counter
```

```
16 40 ff fb  //            bne    $18, $00, out_IO #  and jmp to top if not
finished

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

```
R E G I S T E R ' S A F T E R B R E A K

t=5020.0 ns   $r00 = 00000000  ||  t=5020.0 ns   $r16 = 100100c0
t=5020.0 ns   $r01 = 12345678  ||  t=5020.0 ns   $r17 = ffff8000
t=5020.0 ns   $r02 = 87654321  ||  t=5020.0 ns   $r18 = 00000000
t=5020.0 ns   $r03 = abcdef01  ||  t=5020.0 ns   $r19 = 8000ffff
t=5020.0 ns   $r04 = 01fedcba  ||  t=5020.0 ns   $r20 = c0007fff
t=5020.0 ns   $r05 = 5a5a5a5a  ||  t=5020.0 ns   $r21 = e0003fff
t=5020.0 ns   $r06 = ffffffff  ||  t=5020.0 ns   $r22 = f0001fff
t=5020.0 ns   $r07 = 100103f0  ||  t=5020.0 ns   $r23 = f8000fff
t=5020.0 ns   $r08 = fffffeff  ||  t=5020.0 ns   $r24 = fc0007ff
t=5020.0 ns   $r09 = fffffefe  ||  t=5020.0 ns   $r25 = xxxxxxxx
t=5020.0 ns   $r10 = fffffefd  ||  t=5020.0 ns   $r26 = xxxxxxxx
t=5020.0 ns   $r11 = fffffefc  ||  t=5020.0 ns   $r27 = xxxxxxxx
t=5020.0 ns   $r12 = fffffefb  ||  t=5020.0 ns   $r28 = xxxxxxxx
t=5020.0 ns   $r13 = fffffefa  ||  t=5020.0 ns   $r29 = ffffffff
t=5020.0 ns   $r14 = fffffef9  ||  t=5020.0 ns   $r30 = xxxxxxxx
t=5020.0 ns   $r15 = fffffef8  ||  t=5020.0 ns   $r31 = 00000048

time=5020.0 ns M[3F0]=fffffef8

              DATA MEMORY                                    IO MEMORY
t=5020.0 ns   DM[000000c0] = xxxxxxxx  ||  t=5020.0 ns   IOM[000000c0] = 8000ffff
t=5020.0 ns   DM[000000c4] = xxxxxxxx  ||  t=5020.0 ns   IOM[000000c4] = c0007fff
t=5020.0 ns   DM[000000c8] = xxxxxxxx  ||  t=5020.0 ns   IOM[000000c8] = e0003fff
t=5020.0 ns   DM[000000cc] = xxxxxxxx  ||  t=5020.0 ns   IOM[000000cc] = f0001fff
t=5020.0 ns   DM[000000d0] = xxxxxxxx  ||  t=5020.0 ns   IOM[000000d0] = f8000fff
t=5020.0 ns   DM[000000d4] = xxxxxxxx  ||  t=5020.0 ns   IOM[000000d4] = fc0007ff
t=5020.0 ns   DM[000000d8] = xxxxxxxx  ||  t=5020.0 ns   IOM[000000d8] = fe0003ff
t=5020.0 ns   DM[000000dc] = xxxxxxxx  ||  t=5020.0 ns   IOM[000000dc] = ff0001ff
t=5020.0 ns   DM[000000e0] = xxxxxxxx  ||  t=5020.0 ns   IOM[000000e0] = ff8000ff
t=5020.0 ns   DM[000000e4] = xxxxxxxx  ||  t=5020.0 ns   IOM[000000e4] = ffc0007f
t=5020.0 ns   DM[000000e8] = xxxxxxxx  ||  t=5020.0 ns   IOM[000000e8] = ffe0003f
t=5020.0 ns   DM[000000ec] = xxxxxxxx  ||  t=5020.0 ns   IOM[000000ec] = fff0001f
t=5020.0 ns   DM[000000f0] = xxxxxxxx  ||  t=5020.0 ns   IOM[000000f0] = fff8000f
t=5020.0 ns   DM[000000f4] = xxxxxxxx  ||  t=5020.0 ns   IOM[000000f4] = fffc0007
t=5020.0 ns   DM[000000f8] = xxxxxxxx  ||  t=5020.0 ns   IOM[000000f8] = fffe0003
t=5020.0 ns   DM[000000fc] = xxxxxxxx  ||  t=5020.0 ns   IOM[000000fc] = ffff0001
```

Enhanced Operations Verification :

```
@0
3c 01 12 34       //lui $1, 0x1234
34 21 56 78       //ori $1, 0x5678  # $1 <- 0x12345678
00 01 00 30       //push $1         # dM[3F8] <- 0x12345678
3c 02 00 00       //lui $2, 0x0000
34 42 00 02       //ori $2, 0x0002  # $2 <- 0x00000002
20 03 00 03       //addi $3, $0, 03 # $3 <- 0x00000003
3c 04 ff ff       //lui $4 0xFFFF
34 84 ff ff       //ori $4 0xFFFF         # $4 <- 0xFFFFFFFF
00 04 00 30       //push $4         # dM[3F4] <- 0xFFFFFFFF
20 05 00 05       //addi $5, $0, 05 # $5 <- 0x00000005
20 06 00 10       //addi $6, $0, 16 # $6 <- 0x00000010
00 82 38 20       //add $7, $4, $2  # $7 <- 0x00000001
c8 03 00 03       //blt $0, $3, 3
3c 1b ff ff       //lui $27, 0xFFFF
37 7b ff ff       //ori $27, 0xFFFF # $27 <- 0xFFFFFFFF FAIL
00 00 00 0d       //break
20 08 00 08       //addi $8, $0, 08 # $8 <- 0x00000008
cc c3 00 03       //bge $6, $5, 3
3c 1a ff ff       //lui $26, 0xFFFF
37 5a ff ff       //ori $26, 0xFFFF # $26 <- 0xFFFFFFFF FAIL
00 00 00 0d       //break
20 07 00 07       //addi $7, $0, 07 # $7 <- 0x00000007
00 8a 00 34       //mov $10, $4           # $10 <- 0xFFFFFFFF
00 06 00 33       //clr $6          # $6 <- 0x00000000
20 c6 00 06       //add $6, $6, 06  # $6 <- 0x00000006 FAIL if 0x00000016
00 0a 00 30       //push $10        # dM[3F0] <- 0xFFFFFFFF
00 0c 00 31       //pop $12         # $12 <- 0xFFFFFFFF
20 0e 00 10       //addi $14, $0, 14      # $14 <- 0x00000010
00 0e 00 30       //push $14        # dM[3F0] <- 0x00000010
00 0d 00 31       //pop $13         # $13 <- 0x00000010
3c 0f 10 01       //lui $15, 0x1001
35 ef 00 c0       //ori $15, 0x00C0 # $15 <- 0x100100C0
00 00 00 32       //nop
ad e4 00 00       //loop      //sw $4, 0($15)   # dM[0C0] <- 0xFFFFFFFF to
0xFFFFFFFF
21 ef 00 04       //addi $15, $15, 04     # $15 <- 0x100100C4 to 0x100100C4
00 04 20 42       //srl  $4, $4, 1        # logical shift right 1 bit
c1 ad ff fc       //djnz $13, loop  loop = -4 = fffc
00 01 00 33       //clr $1          # $1 <- 0x00000000
00 11 00 31       //pop $17         # $17 <- 0xFFFFFFFF
00 10 00 31       //pop $16         # $16 <- 0x12345678
00 00 00 32       //nop
00 00 00 32       //nop
00 00 00 0d       //break
```

```
 R E G I S T E R ' S A F T E R B R E A K

t=4070.0 ns   $r0 = 00000000  ||  t=4070.0 ns   $r16 = 12345678
t=4070.0 ns   $r1 = 00000000  ||  t=4070.0 ns   $r17 = ffffffff
t=4070.0 ns   $r2 = 00000002  ||  t=4070.0 ns   $r18 = xxxxxxxx
t=4070.0 ns   $r3 = 00000003  ||  t=4070.0 ns   $r19 = xxxxxxxx
t=4070.0 ns   $r4 = 0000ffff  ||  t=4070.0 ns   $r20 = xxxxxxxx
t=4070.0 ns   $r5 = 00000005  ||  t=4070.0 ns   $r21 = xxxxxxxx
t=4070.0 ns   $r6 = 00000006  ||  t=4070.0 ns   $r22 = xxxxxxxx
t=4070.0 ns   $r7 = 00000007  ||  t=4070.0 ns   $r23 = xxxxxxxx
t=4070.0 ns   $r8 = 00000008  ||  t=4070.0 ns   $r24 = xxxxxxxx
t=4070.0 ns   $r9 = xxxxxxxx  ||  t=4070.0 ns   $r25 = xxxxxxxx
t=4070.0 ns   $r10 = ffffffff  ||  t=4070.0 ns   $r26 = xxxxxxxx
t=4070.0 ns   $r11 = xxxxxxxx  ||  t=4070.0 ns   $r27 = xxxxxxxx
t=4070.0 ns   $r12 = ffffffff  ||  t=4070.0 ns   $r28 = xxxxxxxx
t=4070.0 ns   $r13 = 00000000  ||  t=4070.0 ns   $r29 = 000003fc
t=4070.0 ns   $r14 = 00000010  ||  t=4070.0 ns   $r30 = xxxxxxxx
t=4070.0 ns   $r15 = 10010100  ||  t=4070.0 ns   $r31 = xxxxxxxx

time=4070.0 ns M[3F0]=00000010

              DATA MEMORY                                 IO MEMORY
t=4070.0 ns   DM[000000c0] = ffffffff  ||  t=4070.0 ns   IOM[000000c0] = xxxxxxxx
t=4070.0 ns   DM[000000c4] = 7fffffff  ||  t=4070.0 ns   IOM[000000c4] = xxxxxxxx
t=4070.0 ns   DM[000000c8] = 3fffffff  ||  t=4070.0 ns   IOM[000000c8] = xxxxxxxx
t=4070.0 ns   DM[000000cc] = 1fffffff  ||  t=4070.0 ns   IOM[000000cc] = xxxxxxxx
t=4070.0 ns   DM[000000d0] = 0fffffff  ||  t=4070.0 ns   IOM[000000d0] = xxxxxxxx
t=4070.0 ns   DM[000000d4] = 07ffffff  ||  t=4070.0 ns   IOM[000000d4] = xxxxxxxx
t=4070.0 ns   DM[000000d8] = 03ffffff  ||  t=4070.0 ns   IOM[000000d8] = xxxxxxxx
t=4070.0 ns   DM[000000dc] = 01ffffff  ||  t=4070.0 ns   IOM[000000dc] = xxxxxxxx
t=4070.0 ns   DM[000000e0] = 00ffffff  ||  t=4070.0 ns   IOM[000000e0] = xxxxxxxx
t=4070.0 ns   DM[000000e4] = 007fffff  ||  t=4070.0 ns   IOM[000000e4] = xxxxxxxx
t=4070.0 ns   DM[000000e8] = 003fffff  ||  t=4070.0 ns   IOM[000000e8] = xxxxxxxx
t=4070.0 ns   DM[000000ec] = 001fffff  ||  t=4070.0 ns   IOM[000000ec] = xxxxxxxx
t=4070.0 ns   DM[000000f0] = 000fffff  ||  t=4070.0 ns   IOM[000000f0] = xxxxxxxx
t=4070.0 ns   DM[000000f4] = 0007ffff  ||  t=4070.0 ns   IOM[000000f4] = xxxxxxxx
t=4070.0 ns   DM[000000f8] = 0003ffff  ||  t=4070.0 ns   IOM[000000f8] = xxxxxxxx
t=4070.0 ns   DM[000000fc] = 0001ffff  ||  t=4070.0 ns   IOM[000000fc] = xxxxxxxx
```

$13<-0x10h; loop:addi $15,4; srl $4; sw $4, 0($15);

djnz $13, loop

Barrel Shifter Verification :

R1 <- 0xFFFF8000

Do SLA by 2, store to R2

Do SRA by 2, store to R5

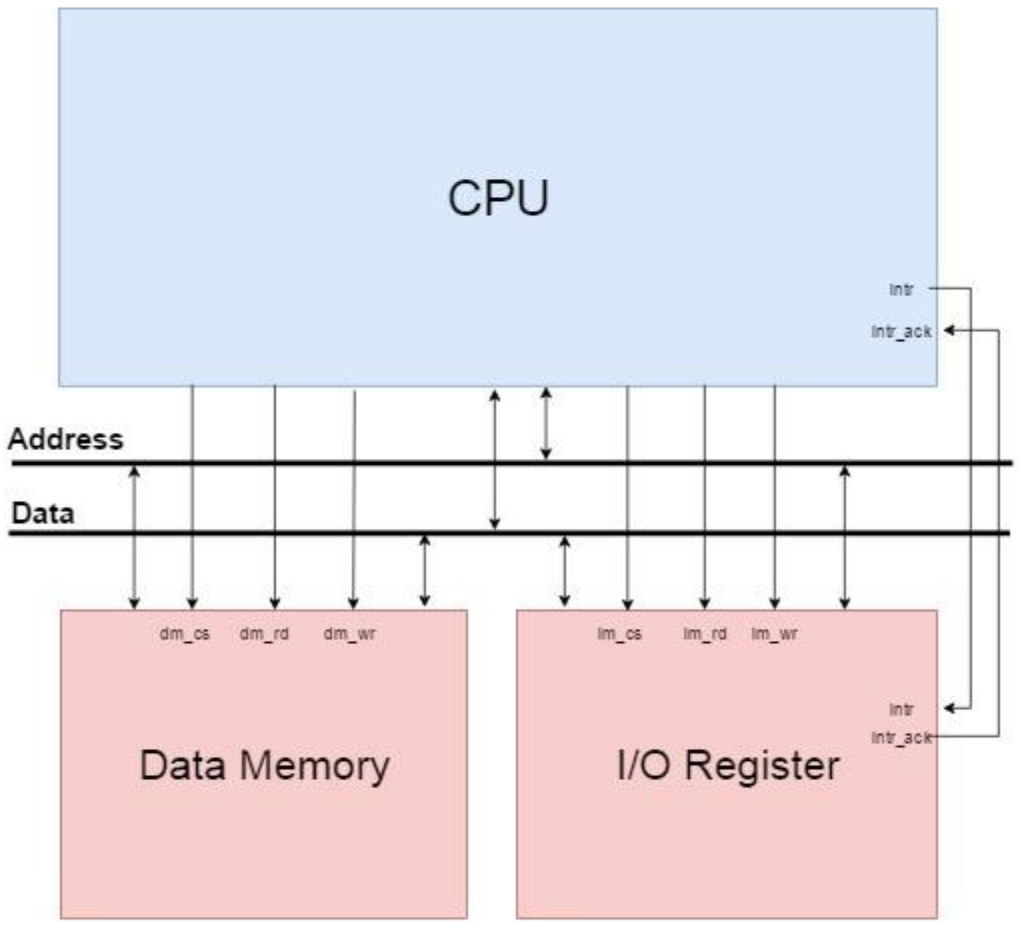
```
R E G I S T E R ' S A F T E R B R E A K

t= 300.0 ns   $r0 = 00000000  ||  t= 300.0 ns   $r16 = xxxxxxxx
t= 300.0 ns   $r1 = ffff8000  ||  t= 300.0 ns   $r17 = xxxxxxxx
t= 300.0 ns   $r2 = fffe0000  ||  t= 300.0 ns   $r18 = xxxxxxxx
t= 300.0 ns   $r3 = fff0001f  ||  t= 300.0 ns   $r19 = xxxxxxxx
t= 300.0 ns   $r4 = ffff8000  ||  t= 300.0 ns   $r20 = xxxxxxxx
t= 300.0 ns   $r5 = ffff8000  ||  t= 300.0 ns   $r21 = xxxxxxxx
t= 300.0 ns   $r6 = xxxxxxxx  ||  t= 300.0 ns   $r22 = xxxxxxxx
t= 300.0 ns   $r7 = xxxxxxxx  ||  t= 300.0 ns   $r23 = xxxxxxxx
t= 300.0 ns   $r8 = xxxxxxxx  ||  t= 300.0 ns   $r24 = xxxxxxxx
t= 300.0 ns   $r9 = xxxxxxxx  ||  t= 300.0 ns   $r25 = xxxxxxxx
t= 300.0 ns   $r10 = xxxxxxxx  ||  t= 300.0 ns   $r26 = xxxxxxxx
t= 300.0 ns   $r11 = xxxxxxxx  ||  t= 300.0 ns   $r27 = xxxxxxxx
t= 300.0 ns   $r12 = xxxxxxxx  ||  t= 300.0 ns   $r28 = xxxxxxxx
t= 300.0 ns   $r13 = xxxxxxxx  ||  t= 300.0 ns   $r29 = 000003fc
t= 300.0 ns   $r14 = xxxxxxxx  ||  t= 300.0 ns   $r30 = xxxxxxxx
t= 300.0 ns   $r15 = xxxxxxxx  ||  t= 300.0 ns   $r31 = xxxxxxxx

time= 300.0 ns M[3F0]=xxxxxxxx

             DATA MEMORY                                  IO MEMORY
t= 300.0 ns   DM[000000c0] = xxxxxxxx  ||  t= 300.0 ns   IOM[000000c0] = xxxxxxxx
t= 300.0 ns   DM[000000c4] = xxxxxxxx  ||  t= 300.0 ns   IOM[000000c4] = xxxxxxxx
t= 300.0 ns   DM[000000c8] = xxxxxxxx  ||  t= 300.0 ns   IOM[000000c8] = xxxxxxxx
t= 300.0 ns   DM[000000cc] = xxxxxxxx  ||  t= 300.0 ns   IOM[000000cc] = xxxxxxxx
t= 300.0 ns   DM[000000d0] = xxxxxxxx  ||  t= 300.0 ns   IOM[000000d0] = xxxxxxxx
t= 300.0 ns   DM[000000d4] = xxxxxxxx  ||  t= 300.0 ns   IOM[000000d4] = xxxxxxxx
t= 300.0 ns   DM[000000d8] = xxxxxxxx  ||  t= 300.0 ns   IOM[000000d8] = xxxxxxxx
t= 300.0 ns   DM[000000dc] = xxxxxxxx  ||  t= 300.0 ns   IOM[000000dc] = xxxxxxxx
t= 300.0 ns   DM[000000e0] = xxxxxxxx  ||  t= 300.0 ns   IOM[000000e0] = xxxxxxxx
t= 300.0 ns   DM[000000e4] = xxxxxxxx  ||  t= 300.0 ns   IOM[000000e4] = xxxxxxxx
t= 300.0 ns   DM[000000e8] = xxxxxxxx  ||  t= 300.0 ns   IOM[000000e8] = xxxxxxxx
t= 300.0 ns   DM[000000ec] = xxxxxxxx  ||  t= 300.0 ns   IOM[000000ec] = xxxxxxxx
t= 300.0 ns   DM[000000f0] = xxxxxxxx  ||  t= 300.0 ns   IOM[000000f0] = xxxxxxxx
t= 300.0 ns   DM[000000f4] = xxxxxxxx  ||  t= 300.0 ns   IOM[000000f4] = xxxxxxxx
t= 300.0 ns   DM[000000f8] = xxxxxxxx  ||  t= 300.0 ns   IOM[000000f8] = xxxxxxxx
t= 300.0 ns   DM[000000fc] = xxxxxxxx  ||  t= 300.0 ns   IOM[000000fc] = xxxxxxxx
```

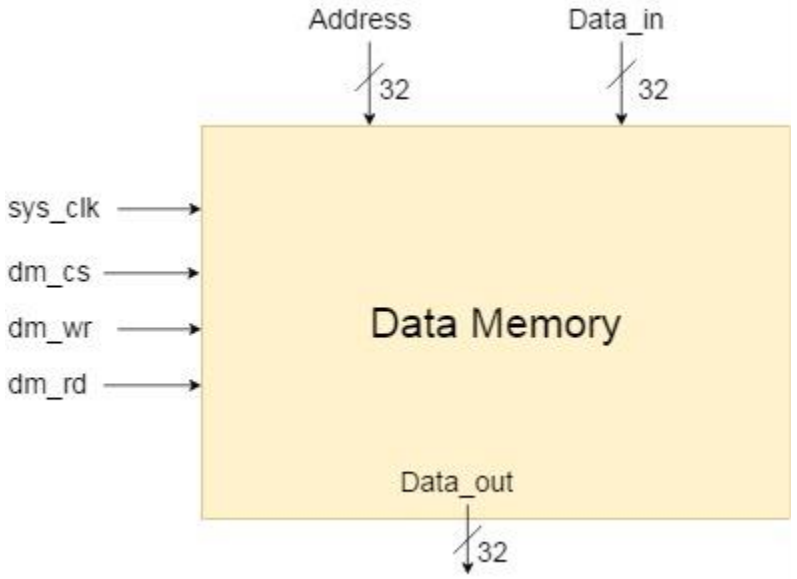
R1 <- 0xFFFF8000

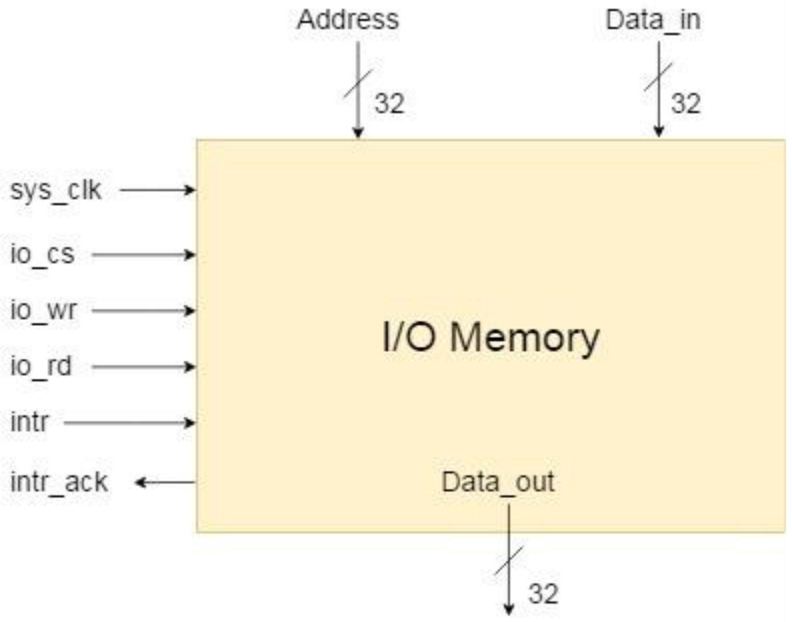
Do ROL by 5, store to R3

Do ROR by 5, store to R4

Hardware Implementation




Address
32
Data_in
32
sys_clk
dm_cs
dm_wr
dm_rd
Data Memory
Data_out
32


Address
32
Data_in
32
sys_clk
io_cs
io_wr
io_rd
intr
intr_ack
I/O Memory
Data_out
32

Additional Discussion and Comments