

AJ Processor ISA

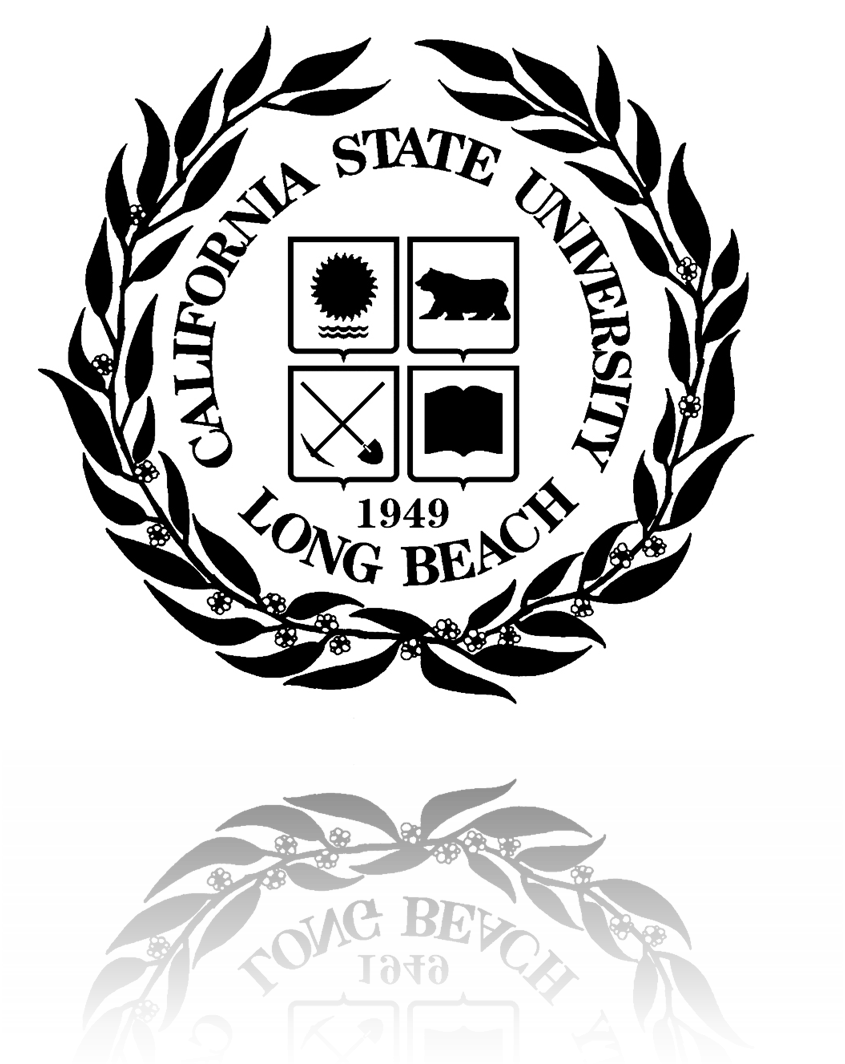
The Instruction Set Architecture for the AJ Processor



November 26, 2017

Jonathan Shihata & Chanartip Soonthornwan

Cal State University, Long Beach



this page is intentionally left blank

Table of Contents

[Introduction 5](#_Toc499583005)

[Memory Structure 6](#_Toc499583006)

[Harvard Memory Architcture 6](#_Toc499583007)

[Machine Register Set 7](#_Toc499583008)

[Register Table 7](#_Toc499583009)

[Flags Register 8](#_Toc499583010)

[PC Register 8](#_Toc499583011)

[IR Register 8](#_Toc499583012)

[Data Types 8](#_Toc499583013)

[Addressing Modes 9](#_Toc499583014)

[Immediate 16-bitAddressing 9](#_Toc499583015)

[PC-Relative Addressing 9](#_Toc499583016)

[Register Addressing 9](#_Toc499583017)

[Register Indirect Addressing 9](#_Toc499583018)

[Shift Ability 10](#_Toc499583019)

[Barrel Shifter Diagram 10](#_Toc499583020)

[CPU Bus Interface 11](#_Toc499583021)

[ALU Diagram 12](#_Toc499583022)

[CPU Architecture Diagram 13](#_Toc499583023)

[Instruction Type Formats 14](#_Toc499583024)

[R-Type Instruction 14](#_Toc499583025)

[I-Type Instruction 14](#_Toc499583026)

[J-Type Instruction 14](#_Toc499583027)

[R-Type Instructions 15](#_Toc499583028)

[Shift Left Logic – SLL 16](#_Toc499583029)

[Shift Right Logic – SRL 17](#_Toc499583030)

[Shift Right Arithmetic – SRA 18](#_Toc499583031)

[Jump Register – JR 19](#_Toc499583032)

[Move From Hi – MFHI 20](#_Toc499583033)

[Move From Lo – MFLO 21](#_Toc499583034)

[Multiply – MULT 22](#_Toc499583035)

[Divide – DIV 23](#_Toc499583036)

[Addition – ADD 24](#_Toc499583037)

[Unsigned Addition – ADDU 25](#_Toc499583038)

[Subtract – SUB 26](#_Toc499583039)

[Unsigned Subtraction – SUBU 27](#_Toc499583040)

[AND 28](#_Toc499583041)

[OR 29](#_Toc499583042)

[Exclusive Or – XOR 30](#_Toc499583043)

[Not Or – NOR 31](#_Toc499583044)

[Set On Less Than – SLT 32](#_Toc499583045)

[Set On Less Than Unsigned – SLTU 33](#_Toc499583046)

[Breakpoint – BREAK 34](#_Toc499583047)

[Set Interrupt – SETIE 35](#_Toc499583048)

[I-Type Instructions 36](#_Toc499583049)

[Branch on Equal – BEQ 37](#_Toc499583050)

[Branch on Not Equal – BNE 38](#_Toc499583051)

[Add Immediate – ADDI 39](#_Toc499583052)

[Set on Less Than Immediate – SLTI 40](#_Toc499583053)

[Set on Less Than Immediate Unsigned – SLTIU 41](#_Toc499583054)

[And Immmediate – ANDI 42](#_Toc499583055)

[Or Immediate – ORI 43](#_Toc499583056)

[Exclusive Or Immediate – XORI 44](#_Toc499583057)

[Load Upper Immediate – LUI 45](#_Toc499583058)

[Load Word – LW 46](#_Toc499583059)

[Store Word – SW 47](#_Toc499583060)

[J-Type Instructions 48](#_Toc499583061)

[Unconditional Jump – J 49](#_Toc499583062)

[Jump And Link – JAL 50](#_Toc499583063)

[Enhanced E-Type Instructions 51](#_Toc499583064)

[Input 52](#_Toc499583065)

[Output 53](#_Toc499583066)

[Reti—return from interupt 54](#_Toc499583067)

[Bge – BRANCH GREATER THAN EQUAL 55](#_Toc499583068)

[Djnz – DECREMENT AND JUMP IF NOT ZERO 56](#_Toc499583069)

[Rotl – rotate left 57](#_Toc499583070)

[Rotr – rotate RIGHT 58](#_Toc499583071)

[Clr – CLEAR 59](#_Toc499583072)

[Mov – MOVE 60](#_Toc499583073)

[Nop – no operation 61](#_Toc499583074)

[Push 62](#_Toc499583075)

[Pop 63](#_Toc499583076)

[Verilog Implementation 65](#_Toc499583077)

[Top Level Module 65](#_Toc499583078)

[CPU Module 68](#_Toc499583079)

[MCU 71](#_Toc499583080)

[ALU 102](#_Toc499583081)

[Barrell Shifter 105](#_Toc499583082)

[ALU MIPS 109](#_Toc499583083)

[ALU Multiplier 112](#_Toc499583084)

[ALU Divider 113](#_Toc499583085)

[Instruction Unit 114](#_Toc499583086)

[IDP 116](#_Toc499583087)

[Data Memory 119](#_Toc499583088)

[IO Memory 121](#_Toc499583089)

[PC 123](#_Toc499583090)

[Memory 124](#_Toc499583091)

[32-bit loadable register 126](#_Toc499583092)

[32-bit Non-Loadable Regiter 127](#_Toc499583093)

[Register File 128](#_Toc499583094)

[Verification 129](#_Toc499583095)

# Introduction

The purpose of this document is for the operation and function of the Instruction Set Architecture function of the AJ Processor, a non-pipelined RISC MIPS based machine. Presented is the memory architecture and how it operates with the processor, the CPU architecture and all of its registers and their functions, as well as the CPU control path and data path of the machine. The machine contains 32 general purpose registers some of which may be accessible by the programmer as described. The machine contains dedicated flag registers that are documented here in their operation. The addressing modes along with the instruction type formats are described in detail as well. The processor is MIPS based but contains enhancements influenced by the 8051 processor. This makes the AJ processor more programmer friendly while maintaining the proven architecture of a MIPS cpu.

# Memory Structure

### Harvard Memory Architcture

The Memory is structured using a Harvard Memory architecture. The Instruction memory is separated from the Data Memory and I/O Memory. The memory has 8-bit wide words and is 4KB deep, each address is 32-bits wide and is byte addressable. The memory is implemented in a “Big Endian” format. Since each memory locations holds an 8-bit word, 4 consecutive memory locations will be used to create a single 32-bit memory operand. The processor is optimized to execute instructions while reading/writing data in parallel.

|  |  |
| --- | --- |
| **Address** | **Memory** |
| 4095 | \_\_ |
| … | \_\_ |
| 3 | AB |
| 2 | CD |
| 1 | 12 |
| 0 | 34 |

**32’h ABCD\_1234**

1 Instruction

**CPU**

**Instruction Memory**

**4K**

**Data Memory**

**4K**

**I/O Memory**

**1K**

# Machine Register Set

The machine contains 32 user-available register each 32-bits wide; some of which are preserved across function calls. There are certain non-user directly modifiable registers such as, the $zero register which always contains 32’b0, the $at register which is used for temporary calculations by the assembler, $gp which is used for the global pointer, $sp which is used for the stack pointer, $fp which is used for the frame pointer, and the $ra holds the return address. The $s and $t registers are the most often used registers for operands of the ALU. Below is a table of the 32 machine registers of the CPU.

### Register Table

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Number | USE | Preservation on Call? |
| $zero | 0 | Constant value 0 | N/A |
| $at | 1 | Assembler Temporary | No |
| $v0 - $v1 | 2 – 3 | Value for Function results and expression evaluation | No |
| $a0 - $a3 | 4 – 7 | Arguments | No |
| $t0 - $t7 | 8 – 15 | Temporaries | No |
| $s0 - $s7 | 16 – 23 | Saved Temporaries | Yes |
| $t8 - $t9 | 24 – 25 | Temporaries | No |
| $k0 - $k1 | 26 – 27 | Reserved for OS kernel | No |
| $gp | 28 | Global Pointer | Yes |
| $sp | 29 | Stack Pointer | Yes |
| $fp | 30 | Frame Pointer | Yes |
| $ra | 31 | Return Address | Yes |

### Flags Register

The status of the flags are held in this register. Instructions that are affected or affect any of these registers are compared to these registers and update their contents. Most are set by arithmetic or logical operations except for the Interrupt Enable. The flags registers are shown in the table below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 | 3 | 2 | 1 | 0 |
| IE | C | N | Z | V |
| Interrupt Enable | Carry Flag | Negative Flag | Zero Flag | Overflow Flag |

### PC Register

The PC (Program Counter) register holds the 32-bit value of the address of the next instruction to be fetched. After each instruction is executed the PC register is incremented by four since each instruction consumes 4 memory locations. the register may be accessed directly by the programmer, or through I or J type instructions such as Branches and Jumps.

### IR Register

The IR (Instruction Register) is a 32-bit user direct addressable register that holds the current instruction being executed or decoded. In it is contained the opcode, operands, a function, and shift amount, or immediate value, or jump address as defined by the instruction format type shown in this document.

# Data Types

The Processor has two possible data types: 32-bit signed and unsigned integers. Instructions can utilize integers smaller than 32-bits, but no larger. In the case of multiply and divide instructions the resultant is stored in two 32-bit registers (HI and LO), thus creating a 64-bit resultant. In a divide instruction, the 2 32-bit registers are used for the 32-bit remainder and 32-bit quotient.

* 32-bit Unsigned Integer
  + Updated Flags consist of Overflow, Carry, and Zero
* 32-bit Signed Integer
  + Updated Flags consist of Overflow, Negative, Carry, and Zero

# Addressing Modes

### Immediate 16-bitAddressing

* + I type instructions use a 16-bit immediate field at the least significant bit field of the instruction.

|  |
| --- |
| 16 bits |
| Immediate |

* + Used in branch instructions where if condition is met, the 16-bit immediate field is sign extended into 32 bits with 2 bits of 0, and shifted left 2 places, then written to the program counter and instructions continue executing.

i.e. PC 🡨 { Sign Ext[imm]<<2, 2’b0}

|  |  |  |
| --- | --- | --- |
| 14 bits | 16 bits | 2 bits |
| Sign Extension | **Immediate << 2** | **00** |

### PC-Relative Addressing

* + J-Type instructions (Jumps) use a 26-bit immediate addressing mode, where the lower 26-bits of the instruction represent the immediate field. The 26-bits are shifted left by 2 with 2 bits of 0 and the upper 4 bits of the PC are concatenated together to create the 32-bit address to jump to.

Example: PC 🡨 {4’(PC+4), Sign Ext[26’imm]<<2, 2’b0}

|  |  |  |
| --- | --- | --- |
| 4 bits | 26 bits | 2 bits |
| PC+4 | **Sign Extension Immediate 16 << 2** | **00** |

### Register Addressing

* + Used in R-Type instructions, operands of instruction are pointed by $rs and $rt, while the output value is placed in the destination register pointed by $rd.

Example: ADD $s1, $s2, $s3 R[$s1] 🡨 $s2 + $s3

### Register Indirect Addressing

* + Used for LOAD/STORE instructions.
  + Format: LOAD/STORE $rt offset($rs)
    - LOAD uses address from $rs and adds a 16-bit offset and loads $rt with the memory contents at that location
    - STORE uses the value in $rt and stores its contents into the memory at location pointed at by $rs with a 16-bit offset
  + Example: LW $rt 8($rs) R[$rt] 🡨 Mem[8+$rs]

# Shift Ability

The processor functionality is expanded to add the ability to perform shifts. A barrel shifter is implemented into the design to allow for this. The implementation of the barrel shifter allows for the shifting of up to 32-bits in either direction in one clock cycle. This allows our design to preform right rotations. The shifter operates in parallel to the ALU independently. The diagram below shows the simplified path for the barrel shifter.

### Barrel Shifter Diagram

Register File

RS RT

Shamt

Barrel Shifter

ALU

Sel

Y-mux

Y

# CPU Bus Interface

CPU

D\_in

D\_out

ALU\_out

**Address**

**Data**

**Data**

I/O Memory

Data Memory

Addr

D\_in

D\_out

Addr

D\_in

D\_out

# ALU Diagram

Multiplier

S x T

[63:0] Product

Product

S

S

ALU

T

T

V

C

C

V

S

{Y\_Hi. Y\_Lo}

Y-MUX

{Y\_Hi. Y\_Lo}

FS

T

Z

**=**

FS

0

FS

Divider

S / T

S

N

Y\_Hi[31]

0

1

T

Y\_Lo[31]

FS

Remainder, Quotient

[63:0]

{Remainder, Quotient}

# CPU Architecture Diagram

S\_sel

SP\_sel

**Instruction Data Path**

Jump

Branch

DY (Mem/IO)

0

1

S\_Addr

PC\_in

ALU\_out

$sp

S\_Addr

D\_EN

T\_Addr

S

Register File

T

D\_Addr

D\_in

RS

PC\_in

PC\_out

Instruction Unit

Sign\_ext 16

2

1

0

D\_in

0

1

S

Y\_Lo

ALU

Y\_Hi

T

T\_addr

$sp

$ra

D\_addr

ALU\_out

100

011

Y-MUX

000

010

001

RT

0

1

2

3

**ALU\_out**

T-MUX

0 1 2 3

**D\_out**

Barrel Shifter

D\_in

Sh\_sel

Shamt

Hi

Lo

Y\_sel

{27’b0,flags}

PC\_in

DA\_sel

shamt

SH\_sel

T\_sel

# Instruction Type Formats

### R-Type Instruction

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **opcode**  **000000** | **rs**  **00000** | **rt**  **00000** | **rd**  **00000** | **shamt**  **00000** | **funct**  **000000** |

6 bits 5 bits 5 bits 5 bits 5 bits 6 bits

### I-Type Instruction

31 26 25 21 20 16 15 0

|  |  |  |  |
| --- | --- | --- | --- |
| **opcode**  **000000** | **rs** | **rt** | **offset** |

6 bits 5 bits 5 bits 16 bits

### J-Type Instruction

31 26 25 0

|  |  |
| --- | --- |
| **opcode**  **000000** | **offset** |

6 bits 26 bits

# R-Type Instructions

**Instruction Format Bit Assignment:**

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **opcode**  **000000** | **rs**  **00000** | **rt**  **00000** | **rd**  **00000** | **shamt**  **00000** | **funct**  **000000** |

6 bits 5 bits 5 bits 5 bits 5 bits 6 bits

**Flags:**

**C**: Carry flag

**N**: Negative flag

**Z**: Zero flag

**O**: Overflow flag

## Shift Left Logic – SLL

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **000000** | **00000** | **rt** | **rd** | **shamt** | **funct** |

6 bits 5 bits 5 bits 5 bits 5 bits 6 bits

format: sll rd, rt, shamt

purpose: shift left contents of rt by shamt and store it in rd

description: R[rd]= R[rt] << shamt

the destination register(rd) gets the contents of the source(rt) register shifted logically left by the shamt field. zeros are shifted in for the shamt.

restrictions:

the shift amount is limited by what can be specified using only 5 bits or a value of 32.

operation:

PC ← PC +4; IR ← M[PC];

RS ← reg[IR[25-21]]; RT ← reg[IR[20-16]];

ALU\_OUT <- RT << shamt;

reg[IR[15-11]] ← ALU\_OUT(RT << shamt);

exceptions:

n/a

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| SLL R3, R1, 29 | 000000\_00000\_00001\_00011\_11101\_000000 |
| SLL R10, R13, 4 | 000000\_00000\_01101\_01010\_00100\_000000 |

## Shift Right Logic – SRL

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **000000** | **00000** | **rt** | **rd** | **shamt** | **funct** |

6 bits 5 bits 5 bits 5 bits 5 bits 6 bits

format: srl rd, rt, shamt

purpose: shift right the contents of rt by shamt and store it in rd

description: R[rd]= R[rt] >> shamt

the destination register(rd) gets the contents of the source(rt) register shifted logically right by the shamt field. zeros are shifted in for the shamt.

restrictions:

the shift amount is limited by what can be specified using only 5 bits or a value of 32.

operation:

PC ← PC +4; IR <- M[PC];

RS ← reg[IR[25-21]]; RT ← reg[IR[20-16]];

ALU\_OUT <- RT >> shamt;

reg[IR[15-11]] ← ALU\_OUT(RT >> shamt);

exceptions:

n/a

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| SRL R3, R1, 29 | 000000\_00000\_00001\_00011\_11101\_000000 |
| SRL R10, R13, 4 | 000000\_00000\_01101\_01010\_00100\_000000 |

## Shift Right Arithmetic – SRA

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **000000** | **00000** | **rt** | **rd** | **shamt** | **000011** |

6 bits 5 bits 5 bits 5 bits 5 bits 6 bits

format: sra rd, rt, shamt

purpose: shift right the contents of rt by shamt and store it in rd

description: R[rd]= R[rt] >>> shamt

the destination register(rd) gets the contents of the source(rt) register shifted arithmetically right by the shamt field. sign bit is shifted in.

restrictions:

the shift amount is limited by what can be specified using only 5 bits or a value of 32.

operation:

PC ← PC +4; IR ← M[PC];

RS ← reg[IR[25-21]]; RT ← reg[IR[20-16]];

ALU\_OUT ← RT >>> shamt;

reg[IR[15-11]] <- ALU\_OUT(RT >>> shamt);

exceptions:

n/a

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| SRA R3, R1, 6 | 000000\_00001\_00000\_00011\_00110\_00011 |
| SRA R10, R13, 20 | 000000\_01101\_00000\_01010\_10100\_00011 |

## Jump Register – JR

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **000000** | **rs** | **00000** | **00000** | **00000** | **001000** |

6 bits 5 bits 5 bits 5 bits 5 bits 6 bits

format: jr rs

purpose: jump to the address stored in rs.

description: pc ← M[rs]

jumps to the effective address specified by $rs.

restrictions:

if either of the two least significant bits are not zero, then an error occurs and will violate the branch address.

operation:

PC ← PC +4; IR ← M[PC];

RS ← REG[IR[25-21]]; RT ← REG[IR[20-16]];

ALU\_OUT ← RS;

REG[IR[15-11]] ← ALU\_OUT(RT >>> SHAMT);

exceptions:

n/a

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| JR R31 | 000000\_11111\_00000\_00000\_00000\_001000 |
| JR R16 | 000000\_10000\_00000\_00000\_00000\_001000 |

## Move From Hi – MFHI

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **000000** | **00000** | **00000** | **rd** | **00000** | **010000** |

6 bits 5 bits 5 bits 5 bits 5 bits 6 bits

format: mfhi rd

purpose: to copy the special hi register to the general purpose register rd.

description: R[rd] ← hi

the contents of special register hi is loaded into rd.

restrictions:

mfhi will not allow a multiply or divide instruction within two instructions after it has been executed because it violates how the mips pipeline works.

operation:

PC ← PC +4; IR ← M[PC];

RS ← REG[IR[25-21]]; RT ← REG[IR[20-16]];

REG[IR[15-11]] ← ALU\_OUT[31-16];

exceptions:

n/a

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| MFHI R6 | 000000\_00000\_00000\_00000\_00110\_010000 |
| MFHI R8 | 000000\_00000\_00000\_00000\_01000\_010000 |

## Move From Lo – MFLO

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **000000** | **00000** | **00000** | **rd** | **00000** | **010010** |

6 bits 5 bits 5 bits 5 bits 5 bits 6 bits

format: mflo rd

purpose to copy the special lo register to the general purpose register rd.

description: R[rd] ← lo

the contents of special register lo is loaded into rd.

restrictions:

mflo will not allow a multiply or divide instruction within two instructions after it has been executed because it violates how the mips pipeline works.

operation:

PC ← PC +4; IR ← M[PC];

RS ← REG[IR[25-21]]; RT ← REG[IR[20-16]];

REG[IR[15-11]] ← ALU\_OUT[15-16];

exceptions:

n/a

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| MFLO R6 | 000000\_00000\_00000\_00000\_00110\_010010 |
| MFLO R8 | 000000\_00000\_00000\_00000\_01000\_010010 |

## Multiply – MULT

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **000000** | **rs** | **rt** | **00000** | **00000** | **011000** |

6 bits 5 bits 5 bits 5 bits 5 bits 6 bits

format: mult rs, rt

purpose to multiply 32 bit signed integers

description: {hi, lo} ← R[rs] \* R[rt]

the 32-bit value in $rt are multiplied by the 32-bit value in $rs, and storing the the 64-bit result in {hi,lo}

restrictions:

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

operation:

PC ← PC +4; IR ← M[PC];

RS ← REG[IR[25-21]]; RT ← REG[IR[20-16]];

ALU\_OUT[15-16] ← RS \* RT

exceptions:

n/a

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| MULT R3, R1 | 000000\_00011\_00001\_00000\_00000\_011000 |
| MULT R10, R13 | 000000\_01010\_01101\_00000\_00000\_011000 |

## Divide – DIV

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **000000** | **rs** | **rt** | **00000** | **00000** | **011010** |

6 bits 5 bits 5 bits 5 bits 5 bits 6 bits

format: div rs, rt

purpose to divide 32 bit signed integers

description: lo ← R[rs] / R[rt], hi ← R[rs] % R[rt]

the 32-bit value in $rt are divided by the 32-bit value in $rs, and storing the the 64-bit result in {hi,lo}

restrictions:

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

operation:

PC ← PC +4; IR ← M[PC];

RS ← REG[IR[25-21]]; RT ← REG[IR[20-16]];

ALU\_OUT[15-16] ← RS / RT

exceptions:

n/a

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| DIV R3, R1 | 000000\_00011\_00001\_00000\_00000\_011010 |
| DIV R10, R13 | 000000\_01010\_01101\_00000\_00000\_011010 |

## Addition – ADD

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **000000** | **rs** | **rt** | **rd** | **00000** | **100000** |

6 bits 5 bits 5 bits 5 bits 5 bits 6 bits

format: add rd, rs, rt

purpose to add 32 bit signed integers

description: R[rd] ← R[rs] + R[rt]

the 32-bit value in $rt is added to the 32-bit value in $rs, and storing the the 32-bit result in $rd

restrictions:

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

operation:

PC ← PC +4; IR ← M[PC];

RS ← REG[IR[25-21]]; RT ← REG[IR[20-16]];

ALU\_OUT ← RS + RT

REG[IR[15-11]]; RT ← ALU\_OUT(RS + RT);

exceptions:

n/a

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| ADD R3, R1, R2 | 000000\_00001\_00010\_00011\_00000\_100000 |
| ADD R10, R13, R6 | 000000\_01101\_00110\_01010\_00000\_100000 |

## Unsigned Addition – ADDU

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **000000** | **rs** | **rt** | **rd** | **00000** | **100001** |

6 bits 5 bits 5 bits 5 bits 5 bits 6 bits

format: addu rd, rs, rt

purpose to add 32 bit signed integers

description: R[rd] ← R[rs] + R[rt]

the 32-bit value in $rt is added to the 32-bit value in $rs, and storing the the 32-bit result in $rd

restrictions:

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

operation:

PC ← PC +4; IR ← M[PC];

RS ← REG[IR[25-21]]; RT ← REG[IR[20-16]];

ALU\_OUT ← RS + RT

REG[IR[15-11]]; RT ← ALU\_OUT(RS + RT);

exceptions:

n/a

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| ADDU R3, R1, R2 | 000000\_00001\_00010\_00011\_00000\_100001 |
| ADDU R10, R13, R6 | 000000\_01101\_00110\_01010\_00000\_100001 |

## Subtract – SUB

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **000000** | **rs** | **rt** | **rd** | **00000** | **100010** |

6 bits 5 bits 5 bits 5 bits 5 bits 6 bits

format: sub rd, rs, rt

purpose to subtract 32 bit signed integers

description: R[rd] ← R[rs] - R[rt]

the 32-bit value in $rt is sutracted to the 32-bit value in $rs, and storing the the 32-bit result in $rd

restrictions:

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

operation:

PC ← PC +4; IR ← M[PC];

RS ← REG[IR[25-21]]; RT ← REG[IR[20-16]];

ALU\_OUT ← RS - RT

REG[IR[15-11]]; RT ← ALU\_OUT(RS - RT);

exceptions:

n/a

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| SUB R3, R1, R2 | 000000\_00001\_00010\_00011\_00000\_100010 |
| SUB R10, R13, R6 | 000000\_01101\_00110\_01010\_00000\_100010 |

## Unsigned Subtraction – SUBU

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **000000** | **rs** | **rt** | **rd** | **00000** | **100011** |

6 bits 5 bits 5 bits 5 bits 5 bits 6 bits

format: subu rd, rs, rt

purpose to subtract 32 bit signed integers

description: R[rd] ← R[rs] - R[rt]

the 32-bit value in $rt is subtracted to the 32-bit value in $rs, and storing the the 32-bit result in $rd

restrictions:

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

operation:

PC ← PC +4; IR ← M[PC];

RS ← REG[IR[25-21]]; RT ← REG[IR[20-16]];

ALU\_OUT ← RS - RT

REG[IR[15-11]]; RT ← ALU\_OUT(RS - RT);

exceptions:

n/a

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| SUBU R3, R1, R2 | 000000\_00001\_00010\_00011\_00000\_100011 |
| SUBU R10, R13, R6 | 000000\_01101\_00110\_01010\_00000\_100011 |

## AND

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **000000** | **rs** | **rt** | **rd** | **00000** | **100100** |

6 bits 5 bits 5 bits 5 bits 5 bits 6 bits

format: and rd, rs, rt

purpose to do a bitwise and

description: R[rd] ← R[rs] & R[rt]

the 32-bit value in $rt is and’d bit by bit to the 32-bit value in $rs, and storing the the 32-bit result in $rd

restrictions:

n/a

operation:

PC ← PC +4; IR ← M[PC];

RS ← REG[IR[25-21]]; RT ← REG[IR[20-16]];

ALU\_OUT ← RS & RT

REG[IR[15-11]]; RT ← ALU\_OUT(RS & RT);

exceptions:

n/a

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| AND R3, R1, R2 | 000000\_00001\_00010\_00011\_00000\_100100 |
| AND R10, R13, R6 | 000000\_01101\_00110\_01010\_00000\_100100 |

## OR

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **000000** | **rs** | **rt** | **rd** | **00000** | **100101** |

6 bits 5 bits 5 bits 5 bits 5 bits 6 bits

format: or rd, rs, rt

purpose to do a bitwise or

description: R[rd] ← R[rs] | R[rt]

the 32-bit value in $rt is or’d bit by bit to the 32-bit value in $rs, and storing the the 32-bit result in $rd

restrictions:

n/a

operation:

PC ← PC +4; IR ← M[PC];

RS ← REG[IR[25-21]]; RT ← REG[IR[20-16]];

ALU\_OUT ← RS | RT

REG[IR[15-11]]; RT ← ALU\_OUT(RS | RT);

exceptions:

n/a

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| OR R3, R1, R2 | 000000\_00001\_00010\_00011\_00000\_100101 |
| OR R10, R13, R6 | 000000\_01101\_00110\_01010\_00000\_100101 |

## Exclusive Or – XOR

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **000000** | **rs** | **rt** | **rd** | **00000** | **100111** |

6 bits 5 bits 5 bits 5 bits 5 bits 6 bits

format: xor rd, rs, rt

purpose to do a bitwise exclusive or

description: R[rd] ← R[rs] ^ R[rt]

the 32-bit value in $rt is exclusive or’d bit by bit to the 32-bit value in $rs, and storing the the 32-bit result in $rd

restrictions:

n/a

operation:

PC ← PC +4; IR ← M[PC];

RS ← REG[IR[25-21]]; RT ← REG[IR[20-16]];

ALU\_OUT ← RS ^ RT

REG[IR[15-11]]; RT ← ALU\_OUT(RS ^ RT);

exceptions:

n/a

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| XOR R3, R1, R2 | 000000\_00001\_00010\_00011\_00000\_100110 |
| XOR R10, R13, R6 | 000000\_01101\_00110\_01010\_00000\_100110 |

## Not Or – NOR

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **000000** | **rs** | **rt** | **rd** | **00000** | **100111** |

6 bits 5 bits 5 bits 5 bits 5 bits 6 bits

format: nor rd, rs, rt

purpose to do a bitwise nor

description: R[rd] ← ~(R[rs] | R[rt])

the 32-bit value in $rt is nor’d bit by bit to the 32-bit value in $rs, and storing the the 32-bit result in $rd

restrictions:

n/a

operation:

PC ← PC +4; IR ← M[PC];

RS ← REG[IR[25-21]]; RT ← REG[IR[20-16]];

ALU\_OUT ← ~(RS | RT)

REG[IR[15-11]]; RT ← ALU\_OUT(~(RS | RT));

exceptions:

n/a

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| NOR R3, R1, R2 | 000000\_00001\_00010\_00011\_00000\_100111 |
| NOR R10, R13, R6 | 000000\_01101\_00110\_01010\_00000\_100111 |

## Set On Less Than – SLT

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **000000** | **rs** | **rt** | **rd** | **00000** | **101010** |

6 bits 5 bits 5 bits 5 bits 5 bits 6 bits

format: slt rd, rs, rt

purpose to compare for less than and return the boolean result as a 0 or 1

description: R[rd] ← (R[rs] < R[rt]) ? 1:0

the 32-bit value in $rt is compared to be less than the 32-bit value in $rs, and storing a 1 in $rd if its true or 0 if its false.

restrictions:

n/a

operation:

PC ← PC +4; IR ← M[PC];

RS ← REG[IR[25-21]]; RT ← REG[IR[20-16]];

ALU\_OUT ← (RS < RT) ? 1:0;

REG[IR[15-11]]; RT ← ALU\_OUT((RS < RT) ? 1:0);

exceptions:

$rd is set to 0 if $rs and $rt are equal to each other.

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| SLT R3, R1, R2 | 000000\_00001\_00010\_00011\_00000\_101010 |
| SLT R10, R13, R6 | 000000\_01101\_00110\_01010\_00000\_101010 |

## Set On Less Than Unsigned – SLTU

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **000000** | **rs** | **rt** | **rd** | **00000** | **101011** |

6 bits 5 bits 5 bits 5 bits 5 bits 6 bits

format: sltu rd, rs, rt

purpose to compare for an unsigned less than and return the Boolean result as a 0 or 1.

description: R[rd] ← (R[rs] < R[rt]) ? 1:0

the 32-bit value in $rt is compared to be less than the 32-bit value in $rs, and storing a 1 in $rd if its true or 0 if its false.

restrictions:

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

operation:

PC ← PC +4; IR ← M[PC];

RS ← REG[IR[25-21]]; RT ← REG[IR[20-16]];

ALU\_OUT ← (RS < RT) ? 1:0;

REG[IR[15-11]]; RT ← ALU\_OUT((RS < RT) ? 1:0);

exceptions:

n/a

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| SLT R3, R1, R2 | 000000\_00001\_00010\_00011\_00000\_101011 |
| SLT R10, R13, R6 | 000000\_01101\_00110\_01010\_00000\_101011 |

## Breakpoint – BREAK

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **000000** | **00000** | **00000** | **00000** | **00000** | **001101** |

6 bits 5 bits 5 bits 5 bits 5 bits 6 bits

format: break

purpose to stop in a program

description:

will display that the break instruction has been fetched with the time, then inform of a safe break and complete a register dump task and a memory dump task.

restrictions:

n/a

operation:

PC ← PC +4; IR ← M[PC];

RS ← REG[IR[25-21]]; RT ← REG[IR[20-16]];

NS ← BREAK;

exceptions:

n/a

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| BREAK | 000000\_00000\_00000\_00000\_00000\_001101 |

## Set Interrupt – SETIE

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **000000** | **00000** | **00000** | **00000** | **00000** | **011111** |

6 bits 5 bits 5 bits 5 bits 5 bits 6 bits

format: setie rd, rs, rt

purpose to set the interrupt flag high

description:

the current interrupt flag will be updated to a value of 1’b1 and pass it to the next interrupt enable flag.

restrictions:

n/A

operation:

PC ← PC +4; IR ← M[PC];

RS ← REG[IR[25-21]]; RT ← REG[IR[20-16]];

NS

exceptions:

n/a

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| SETIE | 000000\_00000\_00000\_00000\_00000\_011111 |

# I-Type Instructions

31 26 25 21 20 16 15 0

|  |  |  |  |
| --- | --- | --- | --- |
| **OPCODE** | **RS** | **RT** | **IMMEDIATE / OFFSET** |

6 bits 5 bits 5 bits 16 bits

**Flags:**

**C**: Carry flag

**N**: Negative flag

**Z**: Zero flag

**O**: Overflow flag

## Branch on Equal – BEQ

31 26 25 21 20 16 15 0

|  |  |  |  |
| --- | --- | --- | --- |
| **000100** | **rs** | **rt** | **offset** |

6 bits 5 bits 5 bits 16 bits

format: beq rt, rs, offset

purpose: equal register-register compare and branch with 16-bit offest.

description: if(r[rs] == r[rt]) pc ← pc + 4 + branch\_addr

if the condition is true, the branch is taken and the follwed instruction will not be executed. if the branch is taken, 16-bit offset will be used to reform a pc-relative effective address by concatenated with 14-bit of offset[15], the offset, and two zeroes.

restrictions:

the branch may be followed by another branch or jump that would delay the execution

operation:

branch\_addr ← {14{offset[15]}, offset, {00}}

beq: cond ← r[rs] = r[rt]

if cond then

pc ← (pc+4+ branch\_addr)

end if

exceptions:

The offset must be within the range -32,768 and 32,767.

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| BEQ R1, R2, LOOP | 001100\_00001\_00010\_0000101001011010 |
| BEQ R0, R3, EXIT | 001100\_00000\_00011\_1111111111111111 |

## Branch on Not Equal – BNE

31 26 25 21 20 16 15 0

|  |  |  |  |
| --- | --- | --- | --- |
| **000101** | **rs** | **rt** | **offset** |

6 bits 5 bits 5 bits 16 bits

format: bne rt, rs, offset

purpose: not equal register-register compare and branch with 16-bit offest.

description: if(r[rs] != r[rt]) pc ← pc + 4 + branch\_addr

if the condition is true, the branch is taken and the follwed instruction will not be executed. if the branch is taken, 16-bit offset will be used to reform a pc-relative effective address by concatenated with 14-bit of offset[15], the offset, and two zeroes.

restrictions:

the branch may be followed by another branch or jump that would delay the execution

operation:

branch\_addr ← {14{offset[15]}, offset, {00} }

bne: cond ← r[rs] ≠ r[rt]

if cond then

pc ← (pc+4+ branch\_addr)

end if

exceptions:

The offset must be within the range -32,768 and 32,767.

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| BNE R1, R2, LOOP | 000101\_00001\_00010\_0000101001011010 |
| BNE R0, R3, EXIT | 000101\_00000\_00011\_1111111111111111 |

## Add Immediate – ADDI

31 26 25 21 20 16 15 0

|  |  |  |  |
| --- | --- | --- | --- |
| **001000** | **rs** | **rt** | **immediate** |

6 bits 5 bits 5 bits 16 bits

format: addi rt, rs, offset

purpose: add a constant with 16-bit offest.

description: r[rt] ← r[rs] + sign\_extend(immediate)

adding 32-bit value in r[rs] with 16-bit signed *offet* and the 32-bit logical result is placed into r[rt].

restrictions:

The Immediate value must be within the range -32,768 and 32,767.

operation:

sign\_extend ← {16{immediate[15]}, immediate}

temp ← r[rs] + sign\_extend(immediate)

r[rt] ← temp

exceptions:

n/a

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| ADDI R2, R1, 5 | 001000\_00001\_00010\_0000000000001010 |
| ADDI R3, R0, -1 | 001000\_00000\_00011\_1111111111111111 |

## Set on Less Than Immediate – SLTI

31 26 25 21 20 16 15 0

|  |  |  |  |
| --- | --- | --- | --- |
| **001010** | **rs** | **rt** | **immediate** |

6 bits 5 bits 5 bits 16 bits

format: slti rt, rs, offset

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

description: r[rt] ← r[rs] < sign\_extend(immediate)

compare the content of r[rs] and the 16-bit signed *immediate*, and then record a boolean result of the comparison in r[rt]. if r[rs] less than the immediate is true, return 1 (true). Or 0 (flase) otherwise.

restrictions:

The Immediate value must be within the range -32,768 and 32,767.

operation:

sign\_extend ← {16{immediate[15]}, immediate}

if r[rs] < sign\_extend(immediate) then

r[rt] **←** zero || 1

else

r[rt] **←** zero

exceptions:

n/a

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| SLTI R2, R1, 5 | 001010\_00001\_00010\_0000000000000101 |
| SLTI R3, R0, -1 | 001010\_00000\_00011\_1111111111111111 |

## Set on Less Than Immediate Unsigned – SLTIU

31 26 25 21 20 16 15 0

|  |  |  |  |
| --- | --- | --- | --- |
| **001011** | **rs** | **rt** | **immediate** |

6 bits 5 bits 5 bits 16 bits

format: sltiu rt, rs, immediate

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

description: r[rt] ← r[rs] < sign\_extend(immediate)

compare the content of r[rs] and the 16-bit signed *immediate* as unsigned integer, and then record a boolean result of the comparison in r[rt]. if r[rs] less than the immediate is true, return 1 (true). Or 0 (flase) otherwise.

restrictions:

The Immediate value must be within the range -32,768 and 32,767.

operation:

if r[rs] < sign\_extend(immediate) then

r[rt] **←** zero || 1

else

r[rt] **←** zero

exceptions:

n/a

programming notes:

since sign-extend of immediate is 16 bit, 16th bit could extend the range of unsigned number from [0,32767] to [0, 65535]

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| SLTI R2, R1, 0x4040 | 001010\_00001\_00010\_0100000001000000 |
| SLTI R3, R0, 0x8000 | 001010\_00000\_00011\_1000000000000000 |

## And Immmediate – ANDI

31 26 25 21 20 16 15 0

|  |  |  |  |
| --- | --- | --- | --- |
| **001100** | **rs** | **rt** | **immediate** |

6 bits 5 bits 5 bits 16 bits

format: andi rt, rs, immediate

**purpose:** perform a bitwise logic and with a constant.

description: r[rt] ← r[rs] & zero\_extend(immediate)

extend immediate with 16-bit zero, then bitwise logical and the content of r[rs], and then place the result into r[rt].

restrictions:

The Immediate value must be within the range -32,768 and 32,767.

operation:

zero\_extend **←** {16{zero}, immediate}

r[rt] **←** r[rs] & zero\_extend

exceptions:

n/a

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| ANDI R2, R1, 8 | 001100\_00001\_00010\_0000000000001000 |
| ANDI R3, R0, -1 | 001100\_00000\_00011\_1111111111111111 |

## Or Immediate – ORI

31 26 25 21 20 16 15 0

|  |  |  |  |
| --- | --- | --- | --- |
| **001101** | **rs** | **rt** | **immediate** |

6 bits 5 bits 5 bits 16 bits

format: ori rt, rs, immediate

**purpose:** perform a bitwise logic or with a constant.

description: r[rt] ← r[rs] | zero\_extend(immediate)

extend immediate with 16-bit zero, then bitwise logical or the content of r[rs], and then place the result into r[rt].

restrictions:

The Immediate value must be within the range -32,768 and 32,767.

operation:

zero\_extend **←** {16{zero}, immediate}

r[rt] **←** r[rs] | zero\_extend

exceptions:

n/a

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| ORI R2,0xFFFF | 001101\_00000\_00010\_1111111111111111 |
| ORI R3, 0x0001 | 001101\_00000\_00011\_0000000000000001 |

## Exclusive Or Immediate – XORI

31 26 25 21 20 16 15 0

|  |  |  |  |
| --- | --- | --- | --- |
| **001110** | **rs** | **rt** | **immediate** |

6 bits 5 bits 5 bits 16 bits

format: xori rt, rs, immediate

**purpose:** perform a bitwise logic xor with a constant.

description: r[rt] ← r[rs] ^ zero\_extend(immediate)

extend immediate with 16-bit zero, then bitwise logical xor the content of r[rs], and then place the result into r[rt].

restrictions:

The Immediate value must be within the range -32,768 and 32,767.

operation:

zero\_extend **←** {16{zero}, immediate}

r[rt] **←** r[rs] ^ zero\_extend

exceptions:

n/a

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| XORI R2, R1, 8 | 001100\_00001\_00010\_0000000000001000 |
| XORI R3, R0, -1 | 001100\_00000\_00011\_1111111111111111 |

## Load Upper Immediate – LUI

31 26 25 21 20 16 15 0

|  |  |  |  |
| --- | --- | --- | --- |
| **001111** | **rt** | **00000** | **immediate** |

6 bits 5 bits 5 bits 16 bits

format: lui rt, offset

**purpose:** load an upper half word to a register

description: r[rt] ← {immediate, 16{zero}}

extend immediate with 16-bit zero, then bitwise logical xor the content of r[rs], and then place the result into r[rt].

restrictions:

The Immediate value must be within the range -32,768 and 32,767.

operation:

r[rt] **←** {immediate, 16{zero}}

exceptions:

n/a

programming notes:

assign rs as 5-bit zero at rs bits in the instruction.

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| LUI R2,0xFFFF | 001111\_00000\_00010\_1111111111111111 |
| LUI R3, 0x0001 | 001111\_00000\_00011\_0000000000000001 |

## Load Word – LW

31 26 25 21 20 16 15 0

|  |  |  |  |
| --- | --- | --- | --- |
| **100011** | **base** | **rt** | **offset** |

6 bits 5 bits 5 bits 16 bits

format: lw rt, offset(base)

**purpose:** load a word from memory.

description: r[rt] ← m[gpr[base] + offset]

load 32-bit word from a memory located at the fetched effective address (base+offset) and then place to r[rt].

restrictions:

effective address 2 least-significant bits have to be two zeroes, otherwise, address error exception will occurs.

operation:

sign\_extend ← {16{offset}, offset}

maddr **←** sign\_extend + gpr[base]

r[rt] **←** memory[maddr]

exceptions:

n/a

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| LW R2,0(R15) | 100011\_01111\_00010\_0000000000000000 |
| LW R3, 4(R15) | 100011\_01111\_00011\_0000000000000100 |

## Store Word – SW

31 26 25 21 20 16 15 0

|  |  |  |  |
| --- | --- | --- | --- |
| **101011** | **base** | **rt** | **offset** |

6 bits 5 bits 5 bits 16 bits

format: lw rt, offset(base)

**purpose:** store a word from memory.

description: M[gpr[base]+offset] ← r[rt]

store 32-bit word to a memory located at the fetched effective address (base+offset) and then place to r[rt].

restrictions:

effective address 2 least-significant bits have to be two zeroes, otherwise, address error exception will occurs.

operation:

sign\_extend ← {16{offset}, offset}

maddr **←** sign\_extend + gpr[base]

memory[maddr] **←** r[rt]

exceptions:

n/a

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| SW R2,0(R15) | 101011\_01111\_00010\_0000000000000000 |
| SW R3, 4(R15) | 101011\_01111\_00011\_0000000000000100 |

# J-Type Instructions

31 26 25 0

|  |  |
| --- | --- |
| **OPCODE** | **ADDRESS** |

6 bits 26 bits

**Flags:**

**C**: Carry flag

**N**: Negative flag

**Z**: Zero flag

**O**: Overflow flag

## Unconditional Jump – J

31 26 25 0

|  |  |
| --- | --- |
| **000010** | **address** |

6 bits 26 bits

format: j address

**purpose:** unconditional branch (or jump)

description: pc ← jmp\_address

form a pc-related effective target address from concatenated 4 most-significant bit of pc+4, offset, and two zeroes, and then update the program counter (pc) with the new pc.

restrictions:

the branch may be followed by another branch or jump that would delay the execution

operation:

jmp\_address ← {pc+4 [31:28], offset, 2{zero}}

pc ← jmp\_address

exceptions:

n/a

programming notes:

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| J top | 100011\_01111000100000000000000000 |
| J exit | 100011\_01111000110000000000000100 |

## Jump And Link – JAL

31 26 25 0

|  |  |
| --- | --- |
| **000011** | **address** |

6 bits 26 bits

format: jal address

**purpose:** jump and link

description: gpr[31] ← pc+8; pc ← jmp\_address

store the return address in returning address register (gpr[31]). form a pc-related effective target address from concatenated 4 most-significant bit of pc+4, offset, and two zeroes, and then update the program counter (pc) with the new pc.

restrictions:

the branch may be followed by another branch or jump that would delay the execution

operation:

jmp\_address ← {pc+4 [31:28], offset, 2{zero}}

gpr[31] ← pc+8

pc ← jmp\_address

exceptions:

n/a

programming notes:

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| JAL top | 000011\_01111000100000000000000000 |
| JAL exit | 000011\_01111000110000000000000100 |

# Enhanced E-Type Instructions

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **opcode** | **rs**  **00000** | **rt**  **00000** | **rd**  **00000** | **shamt**  **00000** | **funct**  **000000** |

6 bits 5 bits 5 bits 5 bits 5 bits 6 bits

31 26 25 21 20 16 15 0

|  |  |  |  |
| --- | --- | --- | --- |
| **opcode** | **rs**  **00000** | **rt**  **00000** | **immediate** |

6 bits 5 bits 5 bits 16 bits

31 26 25 0

|  |  |
| --- | --- |
| **opcode** | **offset** |

6 bits 26 bits

26 bits

**Flags:**

**C**: Carry flag

**N**: Negative flag

**Z**: Zero flag

**O**: Overflow flag

## Input

31 26 25 21 20 16 15 0

|  |  |  |  |
| --- | --- | --- | --- |
| **0x1C** | **rs** | **rt** | **immediate** |

6 bits 5 bits 5 bits 16 bits

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:

N/A

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:

n/a

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| input $R5, 0($R3) | 011100\_00011\_00101\_0000000000000000 |
| input $R1, 4($R0) | 011100\_00001\_00000\_0000000000000100 |

## Output

31 26 25 21 20 16 15 0

|  |  |  |  |
| --- | --- | --- | --- |
| **0x1D** | **rs** | **rt** | **immediate** |

6 bits 5 bits 5 bits 16 bits

format: OUTPUT rt offset(rs)

**purpose:** to receive an input from IO and load it into a GPR

description: ioM[rs + offset]← $rt

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

**restrictions:**

N/A

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:

n/a

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| OUTPUT $R5, 0($R3) | 011101\_00011\_00101\_0000000000000000 |
| OUTPUT $R1, 4($R0) | 011101\_00001\_00000\_0000000000000100 |

## Reti—return from interupt

31 26 25 21 20 16 15 0

|  |  |  |  |
| --- | --- | --- | --- |
| **0x1E** | **rs** | **rt** | **immediate** |

6 bits 5 bits 5 bits 16 bits

format: RETI

**purpose:** to return from a interrupt service routine to the location called into the ISR from.

description: PC ← dM[$sp]

Passes in the value at $sp and at the memory location pointed at to by the stack pointer, 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:**

N/A

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:

n/a

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| RETI | 011110\_00000\_00000\_0000000000000000 |

## Bge – BRANCH GREATER THAN EQUAL

31 26 25 21 20 16 15 0

|  |  |  |  |
| --- | --- | --- | --- |
| **0x2D** | **rs** | **rt** | **immediate** |

6 bits 5 bits 5 bits 16 bits

format: BGE rs, rt, branch offset

**purpose:** to branch only if $Rs >= $Rt

**description**: IF ($Rs >= $Rt) PC ← branch offset

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

restrictions:

N/A

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:

n/a

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| BGE $R5, $R3, 3 | 101101\_00101\_00011\_0000000000000011 |
| BGE $R1, $R4, 256 | 101101\_00001\_00100\_0000000100000000 |

## Djnz – DECREMENT AND JUMP IF NOT ZERO

31 26 25 21 20 16 15 0

|  |  |  |  |
| --- | --- | --- | --- |
| **0x2E** | **rs** | **rt** | **immediate** |

6 bits 5 bits 5 bits 16 bits

format: DJNZ $Rs, branch offset

purpose: to branch only if the register decremented is not 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, no jump is done.

restrictions:

N/A

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:

n/a

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| DJNZ $R5, 3 | 101110\_00011\_00000\_0000000000000011 |
| DJNZ $R1, 128 | 101110\_00001\_00000\_0000000010000000 |

## Rotl – rotate left

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **000000** | **00000** | **rt** | **rd** | **shamt** | **000001** |

6 bits 5 bits 5 bits 5 bits 5 bits 6 bits

format: ROTL rd, rt, shamt

purpose: To rotate a word to the left by a specified number of bits, with no carry

**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:

N/A

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:

n/a

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| ROTL $R5, $R2, 4 | 000000\_00000\_00101\_00010\_00100\_000001 |
| ROTL $R1, $R3, 25 | 000000\_00000\_00001\_00011\_11001\_000001 |

## Rotr – rotate RIGHT

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **000000** | **00000** | **rt** | **rd** | **shamt** | **000010** |

6 bits 5 bits 5 bits 5 bits 5 bits 6 bits

format: ROTR rd, rt, shamt

purpose: To rotate a word to the right by a specified number of bits, with no carry

**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:

N/A

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:

n/a

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| ROTR $R5, $R2, 4 | 000000\_00000\_00101\_00010\_00100\_000010 |
| ROTR $R1, $R3, 25 | 000000\_00000\_00001\_00011\_11001\_000010 |

## Clr – CLEAR

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **000000** | **00000** | **rt** | **00000** | **00000** | **000101** |

6 bits 5 bits 5 bits 5 bits 5 bits 6 bits

format: CLR rt

purpose: To clear a GPR to all 0.

**description**: $rt ← 32’B0

Load 32 bits of 0’s into the specified GPR.

restrictions:

N/A

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:

n/a

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| CLR $R5 | 000000\_00000\_00101\_00000\_00000\_000101 |
| CLR $R1 | 000000\_00000\_00001\_00000\_00000\_000101 |

## Mov – MOVE

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **000000** | **rs** | **rt** | **00000** | **00000** | **000111** |

6 bits 5 bits 5 bits 5 bits 5 bits 6 bits

format: MOV rs, rt

purpose: To move a 32 bit value from one register to another

**description**: $Rt ← $Rs

moves the contents of $Rs into $Rt.

restrictions:

N/A

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:

n/a

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| MOV $R5, $R7 | 000000\_00101\_00111\_00000\_00000\_000111 |
| MOV $R1, $R2 | 000000\_00001\_00010\_00000\_00000\_000111 |

## Nop – no operation

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **000000** | **00000** | **00000** | **00000** | **00000** | **001000** |

6 bits 5 bits 5 bits 5 bits 5 bits 6 bits

format: NOP

purpose: to provide 1 “empty” clock tick where no instruction is being preformed

**description**:

The value in the ALU\_OUT register is passed back into ALU\_OUT so that the value isn’t changed, while using up a clock tick to do it.

restrictions:

N/A

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:

n/a

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| NOP | 000000\_00000\_00000\_00000\_00000\_001000 |

## Push

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **000000** | **00000** | **rt** | **00000** | **00000** | **001001** |

6 bits 5 bits 5 bits 5 bits 5 bits 6 bits

format: PUSH rt

purpose: To push a 32 bit value from a register onto the stack

**description**: dM[$sp] ← $Rt

Pushes $rt into the stack. push is defined as a pre-decrement, updates the $sp before pushing into stack.

restrictions:

N/A

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:

n/a

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| PUSH $R5 | 000000\_00000\_00101\_00000\_00000\_001001 |
| PUSH $R7 | 000000\_00000\_00111\_00000\_00000\_001001 |

## Pop

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **000000** | **00000** | **rt** | **00000** | **00000** | **001010** |

6 bits 5 bits 5 bits 5 bits 5 bits 6 bits

format: POP rt

purpose: To pop a 32 bit value from the top of the stack into a register

**description**: $Rt ← dM[$sp]

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

restrictions:

N/A

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:

n/a

**example:**

|  |  |
| --- | --- |
| Assembly Code | Machine Code |
| POP $R5 | 000000\_00000\_00101\_00000\_00000\_001010 |
| POP $R7 | 000000\_00000\_00111\_00000\_00000\_001010 |

this page is intentionally left blank

# Verilog Implementation

## Top Level Module

`timescale 1ns **/** 1ps

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* C E C S 4 4 0 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* File Name: CPU\_Test\_Module.v

\* Project: Final\_Project

\* Designer: Chanartip Soonthornwan, Jonathan Shihata

\* Email: Chanartip.Soonthornwan@gmail.com, JonnyShihata@gmail.com

\* Rev. No.: Version 1.0

\* Rev. Date: Date 11/17/2017

\*

\* Rev. No.1: Version 1.1

\* Rev. Date: Current Rev. Date 11/21/2017

\* Update: Discarded readmemh for other DM, only left for DM14

\* to perform INTR and RETI

\*

\* Purpose: Top level of CPU, Data Memory, and IO Memory.

\* Performing as a test fixture of CPU and utilizing instruction

\* memory and data memory from dat files.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**module** CPU\_Test\_Module**;**

// Inputs

**reg** sys\_clk**;**

**reg** sys\_rst**;**

// Interconnection wires

**wire** **[**31**:**0**]** MAddr**;** // Memory Address from CPU

**wire** **[**31**:**0**]** idp\_out**;** // Data output from IDP in CPU

**wire** **[**31**:**0**]** Mem\_out**,** Mem\_in**;**// DM and IO Memory bus

// Wires for Data Memory

**wire** dm\_cs**,** dm\_rd**,** dm\_wr**;** // chip select, write enable, read enable

**wire** **[**31**:**0**]** DM\_out**;** // Data Memory output

// Wires for IO Memory

**wire** io\_intr**;** // IO interrupt request input

**wire** io\_int\_ack**;** // IO interrupt acknowledge output

**wire** io\_cs**,** io\_wr**,** io\_rd**;** // chip select, write enable, read enable

//wire [31:0] IO\_out, IO\_in; // IO Memory output

// Instantiate CPU

CPU cpu **(**

**.**sys\_clk**(**sys\_clk**),**

**.**reset**(**sys\_rst**),**

**.**intr**(**io\_intr**),** // interrupt input from IO memory

**.**int\_ack**(**io\_int\_ack**),** // interrupt ourput to IO memory

**.**DM\_out**(**DM\_out**),** // Data input from both memories

**.**MAddr**(**MAddr**),** // Memory address to access both memories

**.**idp\_out**(**idp\_out**),** // Data output to both memories

**.**dm\_cs**(**dm\_cs**),** // Data memory

**.**dm\_rd**(**dm\_rd**),** // control

**.**dm\_wr**(**dm\_wr**),** // words

**.**io\_cs**(**io\_cs**),** // IO memory

**.**io\_rd**(**io\_rd**),** // control

**.**io\_wr**(**io\_wr**)** // words

**);**

// Instantiate Data Memory

Data\_Memory DM**(**

**.**clk**(**sys\_clk**),**

**.**dm\_cs**(**dm\_cs**),**

**.**dm\_wr**(**dm\_wr**),**

**.**dm\_rd**(**dm\_rd**),**

**.**Addr**({**20'h0**,**MAddr**[**11**:**0**]}),**

**.**DM\_In**(**idp\_out**),**

**.**DM\_Out**(**DM\_out**)**

**);**

// Instancitate IO Memory

/\* Will Change this one later\*/

IO\_Memory IO**(**

**.**clk**(**sys\_clk**),**

**.**cs**(**io\_cs**),**

**.**wr**(**io\_wr**),**

**.**rd**(**io\_rd**),**

**.**int\_r**(**io\_intr**),**

**.**int\_ack**(**io\_int\_ack**),**

**.**Addr**({**20'h0**,**MAddr**[**11**:**0**]}),**

**.**IO\_In**(**idp\_out**),**

**.**IO\_Out**(**DM\_out**)**

**);**

// Generate 10ns clock period

**always** **#**5 sys\_clk **=** **~**sys\_clk**;**

// Initialization

**initial** **begin**

$timeformat**(-**9**,** 1**,** " ns"**,** 9**);**

sys\_clk **=** 0**;**

sys\_rst **=** 1**;**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

// Initialize Instruction Memory //

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

**@(negedge** sys\_clk**)**

//$readmemh("iMem01\_Sp17\_commented.dat",cpu.iu.IM.Mem);

//$readmemh("iMem02\_Sp17\_commented.dat",cpu.iu.IM.Mem);

//$readmemh("iMem03\_Sp17\_commented.dat",cpu.iu.IM.Mem);

//$readmemh("iMem04\_Sp17\_commented.dat",cpu.iu.IM.Mem);

//$readmemh("iMem05\_Sp17\_commented.dat",cpu.iu.IM.Mem);

//$readmemh("iMem06\_Sp17\_commented.dat",cpu.iu.IM.Mem);

//$readmemh("iMem07\_Sp17\_commented.dat",cpu.iu.IM.Mem);

//$readmemh("iMem08\_Sp17\_commented.dat",cpu.iu.IM.Mem);

//$readmemh("iMem09\_Sp17\_commented.dat",cpu.iu.IM.Mem);

//$readmemh("iMem10\_Sp17\_commented.dat",cpu.iu.IM.Mem);

//$readmemh("iMem11\_Sp17\_commented.dat",cpu.iu.IM.Mem);

//$readmemh("iMem12\_Sp17\_commented.dat",cpu.iu.IM.Mem);

//$readmemh("iMem13\_Fa17\_w\_isr\_commented.dat",cpu.iu.IM.Mem);

//$readmemh("iMem14\_Fa17\_w\_isr\_commented.dat",cpu.iu.IM.Mem);

$readmemh**(**"enhanced.dat"**,**cpu**.**iu**.**IM**.**Mem**);**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

// Initialize Data Memory //

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

**@(negedge** sys\_clk**)**

//$readmemh("dMem01\_Sp17.dat",DM.Mem);

//$readmemh("dMem02\_Sp17.dat",DM.Mem);

//$readmemh("dMem03\_Sp17.dat",DM.Mem);

//$readmemh("dMem04\_Sp17.dat",DM.Mem);

//$readmemh("dMem05\_Sp17.dat",DM.Mem);

//$readmemh("dMem06\_Sp17.dat",DM.Mem);

//$readmemh("dMem07\_Sp17.dat",DM.Mem);

//$readmemh("dMem08\_Sp17.dat",DM.Mem);

//$readmemh("dMem09\_Sp17.dat",DM.Mem);

//$readmemh("dMem10\_Sp17.dat",DM.Mem);

//$readmemh("dMem11\_Sp17.dat",DM.Mem);

//$readmemh("dMem12\_Sp17.dat",DM.Mem);

//$readmemh("dMem13\_Sp17.dat",DM.Mem);

//$readmemh("dMem14\_Sp17.dat",DM.Mem);

$readmemh**(**"dMem\_enhanced.dat"**,**DM**.**Mem**);**

// Bring system to 'known state'

**@(negedge** sys\_clk**)**

sys\_rst **=** 0**;**

**end**

**endmodule**

## CPU Module

`timescale 1ns **/** 1ps

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* C E C S 4 4 0 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* File Name: CPU.v

\* Project: Final\_Project

\* Designer: Chanartip Soonthornwan, Jonathan Shihata

\* Email: Chanartip.Soonthornwan@gmail.com, JonnyShihata@gmail.com

\* Rev. No.: Version 1.0

\* Rev. Date: Date 11/17/2017

\*

\* Rev. No.1: Version 1.1

\* Rev. Date: Current Rev. Date 11/21/2017

\* update: Add wires for perform RETI and INTR (SP\_Sel, S\_Sel,

\* flag\_in, flag\_out).

\* Update MCU and IDP Port list.

\*

\* Purpose: Top level of Instruction Unit(IU, Integer Data Path(IDP),

\* and Control Unit(MCU). Shows connection between modules above.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**module** CPU**(**

**input** sys\_clk**,** reset**,** intr**,** // System clock, reset, IO\_interupt

**output** int\_ack**,** // IO Interrupt Acknowledge

// From and To Memory

**input** **[**31**:**0**]** DM\_out**,** // Data output from Memory

**output** **[**31**:**0**]** MAddr**,** idp\_out**,** // Memory Address and Data in to memory

**output** dm\_cs**,** dm\_rd**,** dm\_wr**,** // Data Memory Controls

**output** io\_cs**,** io\_rd**,** io\_wr // IO Memory Controls

**);**

// Controls from MCU to Instruction Unit (IU)

**wire** pc\_ld**,** pc\_inc**,** ir\_ld**;** // Program Counter Register

**wire** im\_cs**,** im\_rd**,** im\_wr**;** // Instruction Memory

// Controls from MCU to Integer Data Path (IDP)

**wire** D\_En**,** HILO\_ld**;** // Register File

**wire** **[**1**:**0**]** T\_Sel**;** // Controls

**wire** SP\_Sel**,** S\_Sel**;** // Stack Pointer Control

**wire** **[**1**:**0**]** pc\_sel**,** DA\_sel**;** // T-MUX and DA\_MUX for T or D\_Addr

**wire** **[**2**:**0**]** Y\_Sel**;** // ALU\_Out select

**wire** **[**4**:**0**]** FS**;** // Function select

// Flags status

// From IDP and MCU

**wire** c**,**n**,**z**,**v**;**

**wire** **[**4**:**0**]** flag\_in**,** flag\_out**;** // input-output flag

// Interconnection

// From Instruction Unit(IU) to IDP

**wire** **[**31**:**0**]** IR\_out**,** pc\_out**,** se\_16**;**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

// Instantiate MCU //

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

MCU mcu **(**

// Inputs

**.**sys\_clk**(**sys\_clk**),** **.**reset**(**reset**),** **.**intr**(**intr**),**

**.**c**(**c**),** **.**n**(**n**),** **.**z**(**z**),** **.**v**(**v**),**

**.**IR**(**IR\_out**),**

**.**sp\_flags\_in**(**flag\_in**),**

// Outputs

**.**int\_ack**(**int\_ack**),**

// Controls For Instruction Unit

**.**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**),**

// Controls For Integer Data Path (IDP)

**.**D\_En**(**D\_En**),** **.**DA\_sel**(**DA\_sel**),** **.**T\_Sel**(**T\_Sel**),**

**.**HILO\_ld**(**HILO\_ld**),** **.**Y\_Sel**(**Y\_Sel**),** **.**FS**(**FS**),**

**.**SP\_Sel**(**SP\_Sel**),** **.**S\_Sel**(**S\_Sel**),**

**.**sp\_flags\_out**(**flag\_out**),**

// Controls For Data Memory

**.**dm\_cs**(**dm\_cs**),** **.**dm\_rd**(**dm\_rd**),** **.**dm\_wr**(**dm\_wr**),**

// Controls For IO Memory

**.**io\_cs**(**io\_cs**),** **.**io\_rd**(**io\_rd**),** **.**io\_wr**(**io\_wr**)**

**);**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

// Instruction Unit //

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

Instruction\_Unit iu**(**

// Scalar IO

**.**clk**(**sys\_clk**),** **.**reset**(**reset**),**

**.**pc\_sel**(**pc\_sel**),** **.**pc\_ld**(**pc\_ld**),** **.**pc\_inc**(**pc\_inc**),** **.**ir\_ld**(**ir\_ld**),**

**.**im\_cs**(**im\_cs**),** **.**im\_wr**(**im\_wr**),** **.**im\_rd**(**im\_rd**),**

// 32-bit IO

**.**PC\_in**(**MAddr**),** **.**PC\_out**(**pc\_out**),**

**.**IR\_out**(**IR\_out**),** **.**SE\_16**(**se\_16**)**

**);**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

// Integer Datapath //

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

IDP idp**(** // Inputs

**.**clk**(**sys\_clk**),**

**.**reset**(**reset**),**

**.**D\_En**(**D\_En**),** // Data write Enable

**.**DA\_Sel**(**DA\_sel**),** // Destination Address Select

**.**D\_Addr**(**IR\_out**[**15**:**11**]),** // $rd from IR\_out

**.**S\_Addr**(**IR\_out**[**25**:**21**]),** // $rs from IR\_out

**.**T\_Addr**(**IR\_out**[**20**:**16**]),** // $rt from IR\_out

**.**DT**(**se\_16**),** // SignExtension from IU

**.**T\_Sel**(**T\_Sel**),**

**.**FS**(**FS**),** // Address from IR\_out

**.**shamt**(**IR\_out**[**10**:**6**]),** // Shifting amount from IR\_out

**.**HILO\_ld**(**HILO\_ld**),**

**.**DY**(**DM\_out**),** // Input from Data Memory output

**.**pc\_in**(**pc\_out**),** // Input from Instruction Unit PC\_out

**.**Y\_Sel**(**Y\_Sel**),**

**.**SP\_Sel**(**SP\_Sel**),** // Stack

**.**S\_Sel**(**S\_Sel**),** // Pointer Control

**.**sp\_flags\_in**(**flag\_out**),** //

// Outputs

**.**sp\_flags\_out**(**flag\_in**),** //

**.**C**(**c**),** // Flag

**.**V**(**v**),** // status

**.**N**(**n**),** // to

**.**Z**(**z**),** // MCU

**.**ALU\_OUT**(**MAddr**),** // Memory Address to Data and IO memory

**.**D\_OUT**(**idp\_out**)** // Data output to Data and IO memory

**);**

**endmodule**

## MCU

`timescale 1ns **/** 1ps

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* C E C S 4 4 0 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* File Name: MCU.v

\* Project: Final Project

\* Designer: Chanartip Soonthornwan

\* Jonathan Shihata

\* Email: Chanartip.Soonthornwan@gmail.com

\* Jonnyshihata@gmail.com

\*

\* Rev. No.1: Version 1.0

\* Rev. Date: 10/24/2017

\*

\* Rev. No.2: Version 1.1

\* Rev. Date: 11/18/2017

\* update : add Dump\_mem to display 32 registers on Data Memory and IO Memory

\*

\* Rev. No.3: Version 2.0

\* Rev. Date: Current Rev. 11/21/2017

\* update : Added SP\_Sel, S\_Sel for stack PC and flags on Memory.

\* Added ports for stacked flags.

\* Added RETI and Edited INTR states

\*

\* 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.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*---------------------------------------------------------------------------

\* 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, T\_Sel, HILO\_ld, Y\_Sel} = 9'b0\_00\_00\_0\_000;

\* {dm\_cs, dm\_rd, dm\_wr} = 3'b0\_0\_0;

\* {io\_cs, io\_rd, io\_wr} = 3'b0\_0\_0;

\* {SP\_Sel, S\_Sel} = 2'b0\_0;

\* #1 {nsi, nsc, nsv, nsn, nsz} = {psi, psc, psv, psn, psz};

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**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**,** HILO\_ld**,** Y\_Sel**,**

SP\_Sel**,** S\_Sel**,** sp\_flags\_in**,** sp\_flags\_out**,**

dm\_cs**,** dm\_rd**,** dm\_wr**,**

io\_cs**,** io\_rd**,** io\_wr**,**

FS**);**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**input** sys\_clk**,** reset**,** intr**;** // system clock, reset, interrupt request

**input** c**,** n**,** z**,** v**;** // Integer ALU status inputs

**input** **[**31**:**0**]** IR**;** // Instruction Register input from IU

**output** **reg** int\_ack**;** // Interrupt acknowledge

**input** **[**4**:**0**]** sp\_flags\_in**;** // Stack Pointer flags input

// All OF THE REMAINING CONTROL WORD OUTPUTS

// For Instruction Unit

**output** **reg** pc\_ld**,** pc\_inc**,** ir\_ld**;** // Program Counter Register

**output** **reg** im\_cs**,** im\_rd**,** im\_wr**;** // Instruction Memory

// For Data and IO Memory

**output** **reg** dm\_cs**,** dm\_rd**,** dm\_wr**;** // Data Memory

**output** **reg** io\_cs**,** io\_rd**,** io\_wr**;** // IO Memory

//

// For Integer Data Path (IDP)

**output** **reg** D\_En**,** HILO\_ld**;** // Register File

**output** **reg** **[**1**:**0**]** T\_Sel**;** // Controls

**output** **reg** SP\_Sel**,** S\_Sel**;** // Stack Pointer Controls

**output** **reg** **[**4**:**0**]** sp\_flags\_out**;** // Stack Pointer flags output

**output** **reg** **[**1**:**0**]** pc\_sel**,** DA\_sel**;** // T-MUX and DA\_MUX for T or D\_Addr

**output** **reg** **[**2**:**0**]** Y\_Sel**;** // ALU\_Out select

**output** **reg** **[**4**:**0**]** FS**;** // Function select

// Counter variables

**integer** i**,** X**,** Y**;**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// internal data structures

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// state assignments

**parameter**

RESET **=** 00**,** FETCH **=** 01**,** DECODE **=** 02**,**

// R-type (20 instructions)

SLL **=** 3**,** SRL **=** 4**,** SRA **=** 5**,** JR **=** 6**,** MFHI **=** 7**,** MFLO **=** 8**,** MULT **=** 9**,**

DIV **=** 10**,** ADD **=** 11**,** ADDU **=** 12**,** SUB **=** 13**,** SUBU **=** 14**,** AND **=** 15**,** OR **=** 16**,**

XOR **=** 17**,** NOR **=** 18**,** SLT **=** 19**,** SLTU **=** 20**,** SETIE **=** 21**,**

//R-type Continued

JR\_2 **=** 106**,**

// I-type (13 instructions)

BEQ **=** 24**,** BNE **=** 25**,** BLEZ **=** 26**,** BGTZ **=** 27**,** ADDI **=** 28**,**

SLTI **=** 29**,** SLTIU **=** 30**,** ANDI **=** 31**,** ORI **=** 32**,** XORI **=** 33**,**

LUI **=** 34**,** LW **=** 35**,** SW **=** 36**,**

// I-type Continued

BEQ\_2 **=** 124**,** BNE\_2 **=** 125**,** BLEZ\_2 **=** 126**,** BGTZ\_2 **=** 127**,**

LW\_2 **=** 135**,**

// J-type (2 instructions)

J **=** 37 **,** JAL **=** 38**,**

// J-typd Continued

JAL\_2 **=** 138**,**

// E-type

INPUT **=** 80**,** OUTPUT **=** 81**,** RETI **=** 82**,**

INPUT\_2 **=** 180**,** OUTPUT\_2 **=** 181**,** RETI\_2 **=** 182**,**

ROTL **=** 84**,** ROTR **=** 85**,** RETI\_3 **=** 282**,**

ROTL\_2 **=** 184**,** ROTR\_2 **=** 185**,** RETI\_4 **=** 382**,**

BLT **=** 86**,** BGE **=** 87**,** RETI\_5 **=** 482**,**

BLT\_2 **=** 186**,** BGE\_2 **=** 187**,** RETI\_6 **=** 483**,**

CLR **=** 88**,** DJNZ **=** 89**,** MOV **=** 90**,**

CLR\_2 **=** 188**,** DJNZ\_2 **=** 189**,** MOV\_2 **=** 190**,**

NOP **=** 91**,**

PUSH **=** 92**,** POP **=** 96**,**

PUSH\_2 **=** 93**,** POP\_2 **=** 97**,**

PUSH\_3 **=** 94**,** POP\_3 **=** 98**,**

PUSH\_4 **=** 95**,** POP\_4 **=** 99**,**

POP\_5 **=** 100**,**

// Extras

WB\_alu **=** 50**,** WB\_imm **=** 51**,** WB\_Din **=** 52**,** INTR\_1 **=** 501**,**

WB\_hi **=** 53**,** WB\_lo **=** 54**,** WB\_mem **=** 55**,** INTR\_2 **=** 502**,**

BREAK **=** 510**,** INTR\_3 **=** 503**,**

ILLEGAL\_OP **=** 511**,** INTR\_4 **=** 504**,**

INTR\_5 **=** 505**,**

INTR\_6 **=** 506**;**

//state register (up to 512 states)

**reg** **[**8**:**0**]** state**;**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Flags register \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**reg** psi**,** psc**,** psv**,** psn**,** psz**;** // flags present state registers

**reg** nsi**,** nsc**,** nsv**,** nsn**,** nsz**;** // flags next state registers

// Updating 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**};**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**always** **@(posedge** sys\_clk**,** **posedge** reset**)**

**if** **(**reset**)**

**begin**

// 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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** 5'b0**;**

state **=** RESET**;**

**end**

**else**

**case** **(**state**)**

FETCH**:**

**if** **(**int\_ack**==**0 **&** **(**intr**==**1 **&** psi**==**1**))** // Recieve Interrupt Signal

**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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** INTR\_1**;**

**end**

**else** // No Interrupt Signal involved

**begin** //\*\*\* no new interrupt pending; fetch and instruction \*\*\*

**if((**int\_ack**==**1 **&** intr**==**1**)** **||** **(**psi**==**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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** 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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b1\_11\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** FETCH**;**

**end**

DECODE**:**

**begin**

**@(negedge** sys\_clk**)**

// check for MIPS format

// [000000][rs][rt][rd][shmt][func] - R type

**if** **(**CPU**.**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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

// check for function for R type

**case** **(**CPU**.**iu**.**IR\_out**[**5**:**0**])**

6'h00 **:** state **=** SLL**;**

6'h02 **:** state **=** SRL**;**

6'h03 **:** state **=** SRA**;**

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**;**

**default:** state **=** ILLEGAL\_OP**;**

**endcase**

**end** // end of if for R-type Format

**else** **if(**CPU**.**iu**.**IR\_out**[**31**:**26**]** **==** 6'h1F**)**

**begin**

// E-key is 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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

// check for function for E-key type

**case** **(**CPU**.**iu**.**IR\_out**[**5**:**0**])**

6'h01 **:** state **=** ROTL**;**

6'h02 **:** state **=** ROTR**;**

6'h05 **:** state **=** CLR**;**

6'h07 **:** state **=** MOV**;**

6'h08 **:** state **=** NOP**;**

6'h09 **:** state **=** PUSH**;**

6'h0A **:** state **=** POP**;**

**default:** state **=** ILLEGAL\_OP**;**

**endcase**

**end** // end of if E-key format

**else**

**begin**

// it is an I-type or J-type format

// [pppppp][rs][rt][16'b imme] - I type

// [pppppp][26'b imme] - J type

// control word assignments: RS <- $rs, RT <- DT(se\_16)

int\_ack**=**0**;** FS**=**5'h00**;**

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b00\_0\_0\_0**;**

**{**im\_cs**,** im\_rd**,** im\_wr**}** **=** 3'b0\_0\_0**;**

**{**D\_En**,** DA\_sel**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_01\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

// Check opcode for I and J type

**case** **(**CPU**.**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'h2C **:** state **=** BLT**;**

6'h2D **:** state **=** BGE**;**

6'h2E **:** state **=** DJNZ**;**

**default:** state **=** ILLEGAL\_OP**;**

**endcase**

// Case of Branches

// if T\_Sel = 0, RT <- $rt

// so IR[15:0] will be used to calculate

// Branch address.

**if(**state **==** BEQ **||** state **==** BNE **||**

state **==** BLEZ**||** state **==** BGTZ**||**

state **==** BLT **||** state **==** BGE **)**

T\_Sel **=** 2'b00**;**

**else**

T\_Sel **=** 2'b01**;**

**end** // end of else for I-type or J-type formats

**end** // end of DECODE

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

R-type Instruction

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

SLL**:**

**begin**

// control word assignments: ALU\_Out <- $rt << shamt

**@(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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** WB\_alu**;**

**end**

SRL**:**

**begin**

// control word assignments: ALU\_Out <- $rt >> shamt

**@(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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** WB\_alu**;**

**end**

SRA**:**

**begin**

// control word assignments: ALU\_Out <- $rt >> shamt

**@(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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** c**,** v**,** n**,** z**};**

state **=** WB\_alu**;**

**end**

JR**:**

**begin**

// control word assignments: PC <- [$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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** JR\_2**;**

**end**

JR\_2**:**

**begin**

// control word assignments: PC <- [$rt]

**@(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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** FETCH**;**

**end**

MFHI**:**

**begin**

// control word assignments: R[$rd] <- Hi

**@(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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b1\_00\_00\_0\_001**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** FETCH**;**

**end**

MFLO**:**

**begin**

// control word assignments: R[$rd] <- Lo

**@(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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b1\_00\_00\_0\_010**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** FETCH**;**

**end**

MULT**:**

**begin**

// control word assignments: {Hi,Lo} <- R[$rs] \* R[$rt]

**@(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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_1\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** c**,** v**,** n**,** z**};**

state **=** FETCH**;**

**end**

DIV**:**

**begin**

// control word assignments: Lo <- R[$rs] / R[$rt], Hi <- R[$rs] % R[$rt]

**@(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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_1\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** c**,** v**,** n**,** z**};**

state **=** 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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** c**,** v**,** n**,** z**};**

state **=** WB\_alu**;**

**end**

ADDU**:**

**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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** c**,** v**,** n**,** z**};**

state **=** WB\_alu**;**

**end**

SUB**:**

**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'b00\_0\_0\_0**;**

**{**im\_cs**,** im\_rd**,** im\_wr**}** **=** 3'b0\_0\_0**;**

**{**D\_En**,** DA\_sel**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** c**,** v**,** n**,** z**};**

state **=** WB\_alu**;**

**end**

SUBU**:**

**begin**

// control word assignments:

**@(negedge** sys\_clk**)**

// control word assignments: ALU\_Out <- $rs - $rt

**@(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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** c**,** v**,** n**,** z**};**

state **=** WB\_alu**;**

**end**

AND**:**

**begin**

// control word assignments: ALU\_Out <- $rs & $rt

**@(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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** WB\_alu**;**

**end**

OR**:**

**begin**

// control word assignments: ALU\_Out <- $rs | $rt

**@(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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** WB\_alu**;**

**end**

XOR**:**

**begin**

// control word assignments: ALU\_Out <- $rs ^ $rt

**@(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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** WB\_alu**;**

**end**

NOR**:**

**begin**

// control word assignments: ALU\_Out <- ~($rs | $rt)

**@(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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** WB\_alu**;**

**end**

SLT**:**

**begin**

// control word assignments: ALU\_Out <- $rs < $rt ? 1 : 0

**@(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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** c**,** v**,** n**,** z**};**

state **=** WB\_alu**;**

**end**

SLTU**:**

**begin**

// control word assignments: ALU\_Out <- $rs < $rt ? 1 : 0

**@(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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** c**,** v**,** n**,** z**};**

state **=** WB\_alu**;**

**end**

BREAK**:**

**begin**

$display**(**"BREAK INSTRUCTION FETCHED %t"**,**$time**);**

// control word assignments for "deasserting" everything

**@(negedge** sys\_clk**)**

int\_ack**=**0**;** FS**=**5'h00**;**

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b00\_0\_0\_0**;**

**{**im\_cs**,** im\_rd**,** im\_wr**}** **=** 3'b0\_0\_0**;**

**{**D\_En**,** DA\_sel**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

$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**,** **{**CPU\_Test\_Module**.**DM**.**Mem**[**12'h3F0**],**

CPU\_Test\_Module**.**DM**.**Mem**[**12'h3F1**],**

CPU\_Test\_Module**.**DM**.**Mem**[**12'h3F2**],**

CPU\_Test\_Module**.**DM**.**Mem**[**12'h3F3**]});**

$display**(**" "**);**

Dump\_Mem**;**

$display**(**" "**);**

$finish**;**

**end**

SETIE**:**

**begin**

// control word assignments: psi <- 1'b1

**@(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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**1'b1**,** psc**,** psv**,** psn**,** psz**};**

state **=** FETCH**;**

**end**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* I-type Instruction \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

J**:**

**begin**

// ctrl word assignments for PC <- PC + signext(IR[25:0])<<2

**@(negedge** sys\_clk**)**

int\_ack**=**0**;** FS**=**5'h00**;**

**{**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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_100**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** FETCH**;**

**end**

JAL**:**

**begin**

// ctrl word assignments for PC <- PC + signext(IR[25:0])<<2

// R[ra]<- PC

**@(negedge** sys\_clk**)**

int\_ack**=**0**;** FS**=**5'h00**;**

**{**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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b1\_10\_00\_0\_100**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** FETCH**;**

**end**

BEQ**:**

**begin**

// ctrl word assignments for AluOut <- $rs - $rt

**@(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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** c**,** v**,** n**,** z**};**

state **=** BEQ\_2**;**

**end**

BEQ\_2**:**

**begin**

// ctrl word assignments for if(z==1), PC <- PC+signext(IR[15:0])<<2

int\_ack**=**0**;** FS**=**5'h00**;**

**if(**psz **==** 1**)** // rs - rt == 0

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b10\_1\_0\_0**;**

**else**

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b00\_0\_0\_0**;**

**{**im\_cs**,** im\_rd**,** im\_wr**}** **=** 3'b0\_0\_0**;**

**{**D\_En**,** DA\_sel**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** FETCH**;**

**end**

BNE**:**

**begin**

// ctrl word assignments for AluOut <- $rs - $rt

**@(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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** c**,** v**,** n**,** z**};**

state **=** BNE\_2**;**

**end**

BNE\_2**:**

**begin**

// ctrl word assignments for if(z!=1), PC <- PC + signext(IR[15:0])<<2

**@(negedge** sys\_clk**)**

int\_ack**=**0**;** FS**=**5'h00**;**

**if(**psz **==** 0**)** // rs - rt != 0

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b10\_1\_0\_0**;**

**else**

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b00\_0\_0\_0**;**

**{**im\_cs**,** im\_rd**,** im\_wr**}** **=** 3'b0\_0\_0**;**

**{**D\_En**,** DA\_sel**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** FETCH**;**

**end**

BLEZ**:**

**begin**

// ctrl word assignments for RS <- $rs, RT <- $zero

**@(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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** c**,** v**,** n**,** z**};**

state **=** BLEZ\_2**;**

**end**

BLEZ\_2**:**

**begin**

// ctrl word assignments for if(RS<=0), PC <- PC+signext(IR[15:0])<<2

**@(negedge** sys\_clk**)**

int\_ack**=**0**;** FS**=**5'h00**;**

**if(**psn **==** 1 **||** psz **==** 1**)**

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b10\_1\_0\_0**;**

**else**

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b00\_0\_0\_0**;**

**{**im\_cs**,** im\_rd**,** im\_wr**}** **=** 3'b0\_0\_0**;**

**{**D\_En**,** DA\_sel**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** FETCH**;**

**end**

BGTZ**:**

**begin**

// ctrl word assignments for if(RS >= 0), PC <- PC + signext(IR[15:0])<<2

**@(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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** c**,** v**,** n**,** z**};**

state **=** BGTZ\_2**;**

**end**

BGTZ\_2**:**

**begin**

// ctrl word assignments for if(RS >= 0), PC <- PC + signext(IR[15:0])<<2

**@(negedge** sys\_clk**)**

int\_ack**=**0**;** FS**=**5'h00**;**

**if(**psn **==** 0 **||** psz **==** 1**)**

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b10\_1\_0\_0**;**

**else**

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b00\_0\_0\_0**;**

**{**im\_cs**,** im\_rd**,** im\_wr**}** **=** 3'b0\_0\_0**;**

**{**D\_En**,** DA\_sel**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** WB\_imm**;**

**end**

ADDI**:**

**begin**

// control word assignments for ALU\_Out <- $rs + $rt(se\_16)

**@(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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** c**,** v**,** n**,** z**};**

state **=** WB\_imm**;**

**end**

SLTI**:**

**begin**

// ctrl word assignments for ALU\_Out <- if($rs < $rt(se\_16)) 1:0

**@(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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** c**,** v**,** n**,** z**};**

state **=** WB\_imm**;**

**end**

SLTIU**:**

**begin**

// ctrl word assignments for ALU\_Out <- if($rs < $rt(se\_16)) 1:0

**@(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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** c**,** v**,** n**,** z**};**

state **=** WB\_imm**;**

**end**

ANDI**:**

**begin**

// ctrl word assignments for ALU\_Out <- $rs & $rt(se\_16)

**@(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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** 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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** WB\_imm**;**

**end**

XORI**:**

**begin**

// ctrl word assignments for ALU\_Out <- $rs ^ {16'h0, RT[15:0]}

**@(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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** 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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** WB\_imm**;**

**end**

LW**:**

**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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** c**,** v**,** n**,** z**};**

state **=** LW\_2**;**

**end**

LW\_2**:**

**begin**

// control word assignments for D\_in <- M[ALU\_Out]

**@(negedge** sys\_clk**)**

int\_ack**=**0**;** FS**=**5'h00**;**

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b00\_0\_0\_0**;**

**{**im\_cs**,** im\_rd**,** im\_wr**}** **=** 3'b0\_0\_0**;**

**{**D\_En**,** DA\_sel**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b1\_1\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** WB\_Din**;**

**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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** c**,** v**,** n**,** z**};**

state **=** WB\_mem**;**

**end**

WB\_Din**:**

**begin**

// control word assignments for R[rt] <- D\_in[M[ALUout]]

**@(negedge** sys\_clk**)**

int\_ack**=**0**;** FS**=**5'h00**;**

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b00\_0\_0\_0**;**

**{**im\_cs**,** im\_rd**,** im\_wr**}** **=** 3'b0\_0\_0**;**

**{**D\_En**,** DA\_sel**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b1\_01\_00\_0\_011**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** FETCH**;**

**end**

WB\_alu**:**

**begin**

// control word assignments for R[rd] <- ALU\_Out

**@(negedge** sys\_clk**)**

int\_ack**=**0**;** FS**=**5'h00**;**

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b00\_0\_0\_0**;**

**{**im\_cs**,** im\_rd**,** im\_wr**}** **=** 3'b0\_0\_0**;**

**{**D\_En**,** DA\_sel**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b1\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** FETCH**;**

**end**

WB\_imm**:**

**begin**

// control word assignments for R[rt] <- ALU\_Out

**@(negedge** sys\_clk**)**

int\_ack**=**0**;** FS**=**5'h00**;**

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b00\_0\_0\_0**;**

**{**im\_cs**,** im\_rd**,** im\_wr**}** **=** 3'b0\_0\_0**;**

**{**D\_En**,** DA\_sel**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b1\_01\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** FETCH**;**

**end**

WB\_mem**:**

**begin**

// control word assignments for M[ ALU\_Out(rs+se\_16)] <- RT(rt)

**@(negedge** sys\_clk**)**

int\_ack**=**0**;** FS**=**5'h00**;**

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b00\_0\_0\_0**;**

**{**im\_cs**,** im\_rd**,** im\_wr**}** **=** 3'b0\_0\_0**;**

**{**D\_En**,** DA\_sel**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b1\_0\_1**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** FETCH**;**

**end**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* E-key Instruction \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

INPUT**:**

**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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** c**,** v**,** n**,** z**};**

state **=** INPUT\_2**;**

**end**

INPUT\_2**:**

**begin**

// control word assignments for D\_in <- IOM[ALU\_Out]

**@(negedge** sys\_clk**)**

int\_ack**=**0**;** FS**=**5'h00**;**

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b00\_0\_0\_0**;**

**{**im\_cs**,** im\_rd**,** im\_wr**}** **=** 3'b0\_0\_0**;**

**{**D\_En**,** DA\_sel**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b1\_1\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** WB\_Din**;**

**end**

OUTPUT**:**

**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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** c**,** v**,** n**,** z**};**

state **=** OUTPUT\_2**;**

**end**

OUTPUT\_2**:**

**begin**

// control word assignments for IOM[ALU\_Out] <- D\_in

**@(negedge** sys\_clk**)**

int\_ack**=**0**;** FS**=**5'h00**;**

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b00\_0\_0\_0**;**

**{**im\_cs**,** im\_rd**,** im\_wr**}** **=** 3'b0\_0\_0**;**

**{**D\_En**,** DA\_sel**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b1\_0\_1**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** FETCH**;**

**end**

RETI**:**

**begin**

// Pass $sp value to ALU\_Out as an Effective Address

// control word assignments for ALU\_Out <- RS($sp)

**@(negedge** sys\_clk**)**

int\_ack**=**0**;** FS**=**5'h00**;**

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b00\_0\_0\_0**;**

**{**im\_cs**,** im\_rd**,** im\_wr**}** **=** 3'b0\_0\_0**;**

**{**D\_En**,** DA\_sel**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** RETI\_2**;**

**end**

RETI\_2**:**

**begin**

// POP flag status to D\_in

// control word assignments for D\_in <- DM[ALU\_Out($sp)]

**@(negedge** sys\_clk**)**

int\_ack**=**0**;** FS**=**5'h00**;**

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b00\_0\_0\_0**;**

**{**im\_cs**,** im\_rd**,** im\_wr**}** **=** 3'b0\_0\_0**;**

**{**D\_En**,** DA\_sel**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b1\_1\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** RETI\_3**;**

**end**

RETI\_3**:**

**begin**

// POP flag status from D\_in to current flag status

// and increment $sp

// control word assignments for flags <- D\_in[4:0]

// 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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_1**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**sp\_flags\_in**};**

state **=** RETI\_4**;**

**end**

RETI\_4**:**

**begin**

// POP Returning Address(ra) to D\_in before load it to PC

// and increment $sp

// control word assignments for D\_in <- DM[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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b1\_1\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_1**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** RETI\_5**;**

**end**

RETI\_5**:**

**begin**

// Load Return Address from Memory to PC register

// and hold the $sp

// control word assignments for PC <- D\_in(PC),

// ALU\_Out <- ALU\_Out($sp+8)

**@(negedge** sys\_clk**)**

int\_ack**=**0**;** FS**=**5'h00**;**

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b00\_1\_0\_0**;**

**{**im\_cs**,** im\_rd**,** im\_wr**}** **=** 3'b0\_0\_0**;**

**{**D\_En**,** DA\_sel**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_011**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_1**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** RETI\_6**;**

**end**

RETI\_6**:**

**begin**

// Write back the current $sp in regfile

// control word assignments for R[$sp] <- ALU\_Out($sp+8)

**@(negedge** sys\_clk**)**

int\_ack**=**0**;** FS**=**5'h00**;**

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b00\_0\_0\_0**;**

**{**im\_cs**,** im\_rd**,** im\_wr**}** **=** 3'b0\_0\_0**;**

**{**D\_En**,** DA\_sel**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b1\_11\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** FETCH**;**

**end**

ROTL**:**

**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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** c**,** v**,** n**,** z**};**

state **=** WB\_alu**;**

**end**

ROTR**:**

**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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** c**,** v**,** n**,** z**};**

state **=** WB\_alu**;**

**end**

BLT**:**

**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'b00\_0\_0\_0**;**

**{**im\_cs**,** im\_rd**,** im\_wr**}** **=** 3'b0\_0\_0**;**

**{**D\_En**,** DA\_sel**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** c**,** v**,** n**,** z**};**

state **=** BLT\_2**;**

**end**

BLT\_2**:**

**begin**

// control word assignments

// if(n==1) PC <- PC + {sign\_ext(ir[15:0]),00}

**@(negedge** sys\_clk**)**

int\_ack**=**0**;** FS**=**5'h00**;**

**if(**psn **==** 1**)**

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b10\_1\_0\_0**;**

**else**

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b00\_0\_0\_0**;**

**{**im\_cs**,** im\_rd**,** im\_wr**}** **=** 3'b0\_0\_0**;**

**{**D\_En**,** DA\_sel**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** FETCH**;**

**end**

BGE**:**

**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'b00\_0\_0\_0**;**

**{**im\_cs**,** im\_rd**,** im\_wr**}** **=** 3'b0\_0\_0**;**

**{**D\_En**,** DA\_sel**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** c**,** v**,** n**,** z**};**

state **=** BGE\_2**;**

**end**

BGE\_2**:**

**begin**

// control word assignments

// if(n==0 || z==1) PC <- PC + {sign\_ext(ir[15:0]),00}

**@(negedge** sys\_clk**)**

int\_ack**=**0**;** FS**=**5'h00**;**

**if(**psn **==** 0 **||** psz **==** 1**)**

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b10\_1\_0\_0**;**

**else**

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b00\_0\_0\_0**;**

**{**im\_cs**,** im\_rd**,** im\_wr**}** **=** 3'b0\_0\_0**;**

**{**D\_En**,** DA\_sel**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** FETCH**;**

**end**

CLR**:**

**begin**

// control word assignments ALU\_Out <- 0x0000;

**@(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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** CLR\_2**;**

**end**

CLR\_2**:**

**begin**

// control word assignments R[$rt] <- ALU\_out

**@(negedge** sys\_clk**)**

int\_ack**=**0**;** FS**=**5'h00**;**

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b00\_0\_0\_0**;**

**{**im\_cs**,** im\_rd**,** im\_wr**}** **=** 3'b0\_0\_0**;**

**{**D\_En**,** DA\_sel**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b1\_01\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** FETCH**;**

**end**

DJNZ**:**

**begin**

// DJNZ is an I-type enhanced where $rs and $rt field

// should be the same because ALU's FS Dec is using S

// to decrement, but $rt field is used for writing back

// to register files32.

// control word assignments ALU\_Out <- RS-1;

**@(negedge** sys\_clk**)**

int\_ack**=**0**;** FS**=**5'h10**;**

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b00\_0\_0\_0**;**

**{**im\_cs**,** im\_rd**,** im\_wr**}** **=** 3'b0\_0\_0**;**

**{**D\_En**,** DA\_sel**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** c**,** v**,** n**,** z**};**

state **=** DJNZ\_2**;**

**end**

DJNZ\_2**:**

**begin**

// control word assignments R[$rt] <- ALU\_Out;

// if(z==0) PC <- PC+IR[15:0]<<2

**@(negedge** sys\_clk**)**

int\_ack**=**0**;** FS**=**5'h00**;**

**if(**psz**==**0**)**

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b10\_1\_0\_0**;**

**else**

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b00\_0\_0\_0**;**

**{**im\_cs**,** im\_rd**,** im\_wr**}** **=** 3'b0\_0\_0**;**

**{**D\_En**,** DA\_sel**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b1\_01\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** FETCH**;**

**end**

MOV**:**

**begin**

// Move data from R[$rs] to R[$rt]

// control word assignments ALU\_Out <- R[$rs]

**@(negedge** sys\_clk**)**

int\_ack**=**0**;** FS**=**5'h00**;**

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b00\_0\_0\_0**;**

**{**im\_cs**,** im\_rd**,** im\_wr**}** **=** 3'b0\_0\_0**;**

**{**D\_En**,** DA\_sel**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** MOV\_2**;**

**end**

MOV\_2**:**

**begin**

// control word assignments R[$rt] <- ALU\_out

**@(negedge** sys\_clk**)**

int\_ack**=**0**;** FS**=**5'h00**;**

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b00\_0\_0\_0**;**

**{**im\_cs**,** im\_rd**,** im\_wr**}** **=** 3'b0\_0\_0**;**

**{**D\_En**,** DA\_sel**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b1\_01\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** FETCH**;**

**end**

NOP**:**

**begin**

// No Operation. Nothing change in a clock

// control word assignments ALU\_Out <- ALU\_Out

**@(negedge** sys\_clk**)**

int\_ack**=**0**;** FS**=**5'h00**;**

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b00\_0\_0\_0**;**

**{**im\_cs**,** im\_rd**,** im\_wr**}** **=** 3'b0\_0\_0**;**

**{**D\_En**,** DA\_sel**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_1**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** FETCH**;**

**end**

PUSH**:**

**begin**

// Pre-decrement PUSH DM[--$sp] <- $rt

// control word assignments RS <- R[$sp}; RT <- R[$rt]

**@(negedge** sys\_clk**)**

int\_ack**=**0**;** FS**=**5'h00**;**

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b00\_0\_0\_0**;**

**{**im\_cs**,** im\_rd**,** im\_wr**}** **=** 3'b0\_0\_0**;**

**{**D\_En**,** DA\_sel**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b1\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** c**,** v**,** n**,** z**};**

state **=** PUSH\_2**;**

**end**

PUSH\_2**:**

**begin**

// control word assignments ALU\_Out <- RS(sp)-4; RT <- R[$rt]

**@(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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** PUSH\_3**;**

**end**

PUSH\_3**:**

**begin**

// control word assignments DM[ALU\_Out] <- RT(rt);

// and save SP pointer ALU\_Out <- ALU\_Out($sp-4)

**@(negedge** sys\_clk**)**

int\_ack**=**0**;** FS**=**5'h00**;**

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b00\_0\_0\_0**;**

**{**im\_cs**,** im\_rd**,** im\_wr**}** **=** 3'b0\_0\_0**;**

**{**D\_En**,** DA\_sel**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b1\_0\_1**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_1**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** PUSH\_4**;**

**end**

PUSH\_4**:**

**begin**

// control word assignments R[$sp] <- ALU\_Out($sp-4)

**@(negedge** sys\_clk**)**

int\_ack**=**0**;** FS**=**5'h00**;**

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b00\_0\_0\_0**;**

**{**im\_cs**,** im\_rd**,** im\_wr**}** **=** 3'b0\_0\_0**;**

**{**D\_En**,** DA\_sel**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b1\_11\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** FETCH**;**

**end**

POP**:**

**begin**

// Post-increment POP R[rt] <- DM[sp++]

// control word assignments RS <- R[$sp]

**@(negedge** sys\_clk**)**

int\_ack**=**0**;** FS**=**5'h00**;**

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b00\_0\_0\_0**;**

**{**im\_cs**,** im\_rd**,** im\_wr**}** **=** 3'b0\_0\_0**;**

**{**D\_En**,** DA\_sel**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b1\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** POP\_2**;**

**end**

POP\_2**:**

**begin**

// control word assignments ALU\_Out <- RS(sp)

**@(negedge** sys\_clk**)**

int\_ack**=**0**;** FS**=**5'h00**;**

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b00\_0\_0\_0**;**

**{**im\_cs**,** im\_rd**,** im\_wr**}** **=** 3'b0\_0\_0**;**

**{**D\_En**,** DA\_sel**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** POP\_3**;**

**end**

POP\_3**:**

**begin**

// control word assignments D\_in <- DM[ALU\_Out];

// ALU\_Out <- ALU\_out(sp)

**@(negedge** sys\_clk**)**

int\_ack**=**0**;** FS**=**5'h00**;**

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b00\_0\_0\_0**;**

**{**im\_cs**,** im\_rd**,** im\_wr**}** **=** 3'b0\_0\_0**;**

**{**D\_En**,** DA\_sel**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b1\_1\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_1**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** POP\_4**;**

**end**

POP\_4**:**

**begin**

// control word assignments R[rt] <- 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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b1\_01\_00\_0\_011**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_1**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** c**,** v**,** n**,** z**};**

state **=** POP\_5**;**

**end**

POP\_5**:**

**begin**

// control word assignments R[sp] <- ALU\_Out(sp+4)

**@(negedge** sys\_clk**)**

int\_ack**=**0**;** FS**=**5'h00**;**

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b00\_0\_0\_0**;**

**{**im\_cs**,** im\_rd**,** im\_wr**}** **=** 3'b0\_0\_0**;**

**{**D\_En**,** DA\_sel**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b1\_11\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** FETCH**;**

**end**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Other types Instruction \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

ILLEGAL\_OP**:**

**begin**

$display**(**"ILLEGAL OPCODE FETCHED %t"**,**$time**);**

// control word assignments for "deasserting" everything

**@(negedge** sys\_clk**)**

int\_ack**=**0**;** FS**=**5'h00**;**

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b00\_0\_0\_0**;**

**{**im\_cs**,** im\_rd**,** im\_wr**}** **=** 3'b0\_0\_0**;**

**{**D\_En**,** DA\_sel**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

$display**(**" "**);**

$display**(**"Memory:"**);**

$display**(**" "**);**

Dump\_Registers**;**

$display**(**" "**);**

Dump\_PC\_and\_IR**;**

$finish**;**

**end**

INTR\_1**:**

**begin**

// PC gets address of interrupt vector; Save PC in $ra

// control word assignments for RS <- R[$sp],

// R[$ra] <- PC

**@(negedge** sys\_clk**)**

int\_ack**=**0**;** FS**=**5'h00**;**

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b00\_0\_0\_0**;**

**{**im\_cs**,** im\_rd**,** im\_wr**}** **=** 3'b0\_0\_0**;**

**{**D\_En**,** DA\_sel**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b1\_10\_00\_0\_100**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b1\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** INTR\_2**;**

**end**

INTR\_2**:**

**begin**

// Pass $sp from RS to ALU\_Out

// control word assignments for ALU\_Out <- RS($sp)

**@(negedge** sys\_clk**)**

int\_ack**=**0**;** FS**=**5'h00**;**

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b00\_0\_0\_0**;**

**{**im\_cs**,** im\_rd**,** im\_wr**}** **=** 3'b0\_0\_0**;**

**{**D\_En**,** DA\_sel**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** INTR\_3**;**

**end**

INTR\_3**:**

**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'h00**;**

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b00\_0\_0\_0**;**

**{**im\_cs**,** im\_rd**,** im\_wr**}** **=** 3'b0\_0\_0**;**

**{**D\_En**,** DA\_sel**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b1\_1\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** INTR\_4**;**

**end**

INTR\_4**:**

**begin**

// Load PC with M[$sp], Pre-decrement $sp,

// and prepare PUSH current PC on stack.

// control word assignments for 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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_10\_0\_011**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b0\_0\_0**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_1**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** INTR\_5**;**

**end**

INTR\_5**:**

**begin**

// PUSH the current PC on stack, decrement $sp,

// and prepare PUSH current flags on stack.

// control word assignments for DM[$sp-4] <- RT(PC),

// ALU\_Out <- ALU\_Out($sp-4)-4,

// RT <- 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**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b0\_00\_11\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b1\_0\_1**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_1**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**psi**,** psc**,** psv**,** psn**,** psz**};**

state **=** INTR\_6**;**

**end**

INTR\_6**:**

**begin**

// PUSH flags on stack, write current $sp in regfile,

// and set interrupt acknowledge

// control word assignments for DM[ALU\_Out($sp-8)] <- RT(flags)

// R[$sp] <- ALU\_Out($sp-8)

// int\_ack <- 1'b1;

**@(negedge** sys\_clk**)**

int\_ack**=**1**;** FS**=**5'h00**;**

**{**pc\_sel**,** pc\_ld**,** pc\_inc**,** ir\_ld**}** **=** 5'b00\_0\_0\_0**;**

**{**im\_cs**,** im\_rd**,** im\_wr**}** **=** 3'b0\_0\_0**;**

**{**D\_En**,** DA\_sel**,** T\_Sel**,** HILO\_ld**,** Y\_Sel**}** **=** 9'b1\_11\_00\_0\_000**;**

**{**dm\_cs**,** dm\_rd**,** dm\_wr**}** **=** 3'b1\_0\_1**;**

**{**io\_cs**,** io\_rd**,** io\_wr**}** **=** 3'b0\_0\_0**;**

**{**SP\_Sel**,** S\_Sel**}** **=** 2'b0\_0**;**

**#**1 **{**nsi**,** nsc**,** nsv**,** nsn**,** nsz**}** **=** **{**1'b0**,** psc**,** psv**,** psn**,** psz**};**

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**,** CPU\_Test\_Module**.**cpu**.**idp**.**regfile**.**reg\_array**[**i**],**

$time**,** i**+**16**,** CPU\_Test\_Module**.**cpu**.**idp**.**regfile**.**reg\_array**[**i**+**16**]);**

**end**

**end**

**endtask**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// PC and IR Register Dump Task \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**task** Dump\_PC\_and\_IR**;**

**begin**

$display**(**" "**);** $display**(**"PC Register:"**);**

$display**(**"t=%t PC=%h"**,** $time**,** CPU\_Test\_Module**.**cpu**.**iu**.**PC**.**pc\_out**);**

$display**(**" "**);**

$display**(**"IR Register:"**);**

$display**(**"t=%t IR=%h"**,** $time**,** CPU\_Test\_Module**.**cpu**.**iu**.**IR**.**q**);**

$display**(**" "**);** $display**(**" "**);**

**end**

**endtask**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Data Memory and I/O Memory Dump Task \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**task** Dump\_Mem**;**

**begin**

$display**(**"\_\_\_\_\_\_\_Data Memory(DM)\_\_\_\_\_\_\_\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_IO Memory(IOM)\_\_\_\_\_"**);**

**for(**i **=** 9'h0C0**;** i **<** 9'h100**;** i **=** i **+** 4**)** **begin**

X **=** **{**CPU\_Test\_Module**.**DM**.**Mem**[**i**],**

CPU\_Test\_Module**.**DM**.**Mem**[**i**+**1**],**

CPU\_Test\_Module**.**DM**.**Mem**[**i**+**2**],**

CPU\_Test\_Module**.**DM**.**Mem**[**i**+**3**]};**

Y **=** **{**CPU\_Test\_Module**.**IO**.**Mem**[**i**],**

CPU\_Test\_Module**.**IO**.**Mem**[**i**+**1**],**

CPU\_Test\_Module**.**IO**.**Mem**[**i**+**2**],**

CPU\_Test\_Module**.**IO**.**Mem**[**i**+**3**]};**

$display**(**"t=%t DM[%h] = %h || t=%t IOM[%h] = %h"**,**

$time**,** i**,** X**,** $time**,** i**,** Y**);**

**end**

**end**

**endtask**

**endmodule**

## ALU

`timescale 1ns **/** 1ps

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* C E C S 4 4 0 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* File Name: ALU\_32.v

\* Project: Lab\_Assignment\_1

\* Designer: Chanartip Soonthornwan, Jonathan Shihata

\* Email: Chanartip.Soonthornwan@gmail.com, JonnyShihata@gmail.com

\* Rev. No.: Version 1.0

\* Rev. Date: Date 9/3/2017

\*

\* Rev. No.1: Version 1.1

\* Rev. Date: Date 11/19/2017

\* update: Add BarrialShifter32 to perform 0 to 31 bit shifting using

\* shamt bits from Instruction.

\* Update flag status for BarrialShifting.

\*

\* Rev. No.2: Version 1.2

\* Rev. Date: Current Rev. Date 11/26/2017

\* update: Added Rotate Left (ROTL) and Rotate Right (ROTR)

\*

\* Purpose: Top View of 32-bit Arithmatics Logic Unit(ALU). Used for

\* computing Arithmatic of two 32-bit inputs, register S and T.

\* Performs 30 different operations where 28 operations in

\* MIPS\_32.v, multiplication in MPY\_32.v, and division in DIV\_32.v

\*

\* Notes: ~|{Y1, Y2} Nor all bits in side the concatenated Y1 and Y2

\* if all bits are zero, the result is one (or true).

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**module** ALU\_32**(**S**,** T**,** FS**,** shamt**,** Y\_hi**,** Y\_lo**,** C**,** V**,** N**,** Z**);**

**input** **[**31**:**0**]** S**,** T**;** // 32-bit inputs

**input** **[** 4**:**0**]** FS**;** // Function Select

**input** **[** 4**:**0**]** shamt**;** // shifting amount

**output** **[**31**:**0**]** Y\_hi**;** // Upper 32-bit of Output

**output** **[**31**:**0**]** Y\_lo**;** // Lower 32-bit of Output

**output** C**,** V**,** N**,** Z**;** // Flag Status

**wire** mips\_v**,** mips\_c**,** mul\_z**,** mip\_z**,** div\_z**;** // Temporary flag status

**wire** **[**31**:**0**]** mips\_y **;** // Y Output from MIPS\_32

**wire** **[**31**:**0**]** mul\_hi**,** mul\_lo **;** // Y Output from MPY\_32

**wire** **[**31**:**0**]** div\_rem**,** div\_qout **;** // Y Output from DIV\_32

**wire** **[**31**:**0**]** shifted\_T **;** // Y Output from BrrialShifter

**wire** shifted\_C**,** shifted\_z**;** // Temporary shifting flags

**parameter** ADDU **=** 5'h03**,**

SUBU **=** 5'h05**,**

MUL **=** 5'h1E**,**

DIV **=** 5'h1F**,**

SLL **=** 5'h0C**,**

SRL **=** 5'h0D**,**

SRA **=** 5'h0E**,**

ROTL **=** 5'h1A**,**

ROTR **=** 5'h1B**;**

// Multiplication

// return a64-bit result of multiplying S and T

// (S, T, \_\_\_\_\_\_\_Y\_\_\_\_\_\_\_ )

MPY\_32 alu\_mul **(**S**,** T**,** **{**mul\_hi**,** mul\_lo**});**

// Division

// return 32-bit results where Y\_hi = remainder and Y\_lo = qoutian

// (S, T, \_\_rem\_\_, \_\_qout\_\_)

DIV\_32 alu\_div **(**S**,** T**,** div\_rem**,** div\_qout**);**

// Million Instructions Per Second (MIPS)

// return a 32-bit result(Y), overflow flag(v), and carry flag(c)

// (S, T, FS, \_\_Y\_\_\_, \_\_V\_\_\_, \_\_C\_\_\_)

MIPS\_32 alu\_mips**(**S**,** T**,** FS**,** mips\_y**,** mips\_v**,** mips\_c**);**

// Barrial Shifter

// shifting 32-bit input with shift amount

// (T, shamt, type, Y, C)

BarrialShifter32 bs**(**T**,** shamt**,** FS**,** shifted\_T**,** shifted\_C**);**

// Multiplexer to assign 64-bit outputs

// If MULtiplication, return a 64-bit output.

// Else if DIVision, return a remainder and qoutian.

// Else return 32-bit zeros and 32-bit result.

**assign** **{**Y\_hi**,** Y\_lo**}** **=** **(**FS **==** MUL**)?** **{** mul\_hi **,** mul\_lo **}:**

**(**FS **==** DIV**)?** **{** div\_rem**,** div\_qout**}:**

**(**FS **==** SLL **||**

FS **==** SRL **||**

FS **==** SRA **||**

FS **==** ROTL**||**

FS **==** ROTR **)?{**32'b0**,** shifted\_T**}:**

**{**32'b0**,** mips\_y **};**

// Checking and assigning temporary the Zero flag

// ~| is nor reduction operand where

// the all output bits will be Nor

// and return 1 if all bits are zero.

// Return 0 if any bit is one.

**assign** mul\_z **=** **~|{**Y\_hi**,** Y\_lo**},**

div\_z **=** **(**T **==** 32'b0**)?** 1'bz **:** **~|{**Y\_lo**},**

mip\_z **=** **~|{**Y\_lo**},**

shift\_z **=** **~|{**Y\_lo**};**

// Multiplexer to assign the Zero flag output

// assign z flag according to Function Select(FS)

**assign** Z **=** **(**FS **==** MUL**)?** mul\_z**:**

**(**FS **==** DIV**)?** div\_z**:**

**(**FS **==** SLL **||**

FS **==** SRL **||**

FS **==** SRA **||**

FS **==** ROTL**||**

FS **==** ROTR **)?** shift\_z**:**

mip\_z**;**

// Multiplexer to assign Negative flag

// Assign n flag according to the Most Significant Bit(MSB)

**assign** N **=** **(**FS **==** MUL**)** **?** Y\_hi**[**31**]:**

**(**FS **==** DIV**)** **?** Y\_lo**[**31**]:**

**(**FS **==** ADDU**)?** 1'b0**:**

**(**FS **==** SUBU**)?** 1'b0**:**

Y\_lo**[**31**];**

// Multiplexer to assign Overflow flag

// \*Note: Multiplication and Division

// are not affected over flow flag

**assign** V **=** **(**FS **==** MUL **||**

FS **==** DIV **||**

FS **==** SLL **||**

FS **==** SRL **||**

FS **==** SRA **||**

FS **==** ROTL**||**

FS **==** ROTR **)?** 1'bx**:** mips\_v**;**

// Multiplexer to assign Carry flag

// \*Note: Multiplication and Division

// are not affected carry out flag

**assign** C **=** **(**FS **==** MUL**)?** 1'bx**:**

**(**FS **==** DIV**)?** 1'bx**:**

**(**FS **==** SLL **||**

FS **==** SRL **||**

FS **==** SRA **||**

FS **==** ROTL**||**

FS **==** ROTR **)?** shifted\_C**:** mips\_c**;**

**endmodule**

## Barrell Shifter

`timescale 1ns **/** 1ps

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* C E C S 4 4 0 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* File Name: BarrialShifter32.v

\* Project: MIPS\_Final\_project

\* Designer: Chanartip Soonthornwan, Jonathan Shihata

\* Email: Chanartip.Soonthornwan@gmail.com, JonnyShihata@gmail.com

\* Rev. No.: Version 1.0

\* Rev. Date: Date 11/13/2017

\*

\* Rev. No.1: Version 1.1

\* Rev. Date: Current Rev. Date 11/19/2017

\* update: Correct case statement for type and shamt

\*

\* Purpose: Perform Shifting 32-bit input with shifting amount(shamt)

\* in different type of shifting method(type) and output back

\* to ALU\_32

\*

\* Notes: There are 5 modes but only 3 in used (SLL,SRL,SRA)

\* SLL: type = 5'h0C

\* SRL: type = 5'h0D

\* SRA: type = 5'h0E

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**module** BarrialShifter32**(**T**,** shamt**,** **type,** Y**,** C**);**

**input** **[**4**:**0**]** **type;** // Function Select from MCU

**input** **[**4**:**0**]** shamt**;** // Shifting amount from IR[10:6]

**input** **[**31**:**0**]** T**;** // Data input for shifiting

**output** **reg** C**;** // Carry flag

**output** **reg** **[**31**:**0**]** Y**;** // Shifted output

**always@(\*)**

**case(type)**

5'h0C**:** // SLL

**case(**shamt**)**

5'd0**:** **{**C**,**Y**}** **=** **{**1'b0**,** T**};**

5'd1**:** **{**C**,**Y**}** **=** **{**T**[**31**],** T**[**30**:**0**],** 1'b0**};**

5'd2**:** **{**C**,**Y**}** **=** **{**T**[**30**],** T**[**29**:**0**],** 2'b0**};**

5'd3**:** **{**C**,**Y**}** **=** **{**T**[**29**],** T**[**28**:**0**],** 3'b0**};**

5'd4**:** **{**C**,**Y**}** **=** **{**T**[**28**],** T**[**27**:**0**],** 4'b0**};**

5'd5**:** **{**C**,**Y**}** **=** **{**T**[**27**],** T**[**26**:**0**],** 5'b0**};**

5'd6**:** **{**C**,**Y**}** **=** **{**T**[**26**],** T**[**25**:**0**],** 6'b0**};**

5'd7**:** **{**C**,**Y**}** **=** **{**T**[**25**],** T**[**24**:**0**],** 7'b0**};**

5'd8**:** **{**C**,**Y**}** **=** **{**T**[**24**],** T**[**23**:**0**],** 8'b0**};**

5'd9**:** **{**C**,**Y**}** **=** **{**T**[**23**],** T**[**22**:**0**],** 9'b0**};**

5'd10**:** **{**C**,**Y**}** **=** **{**T**[**22**],** T**[**21**:**0**],** 10'b0**};**

5'd11**:** **{**C**,**Y**}** **=** **{**T**[**21**],** T**[**20**:**0**],** 11'b0**};**

5'd12**:** **{**C**,**Y**}** **=** **{**T**[**20**],** T**[**19**:**0**],** 12'b0**};**

5'd13**:** **{**C**,**Y**}** **=** **{**T**[**19**],** T**[**18**:**0**],** 13'b0**};**

5'd14**:** **{**C**,**Y**}** **=** **{**T**[**18**],** T**[**17**:**0**],** 14'b0**};**

5'd15**:** **{**C**,**Y**}** **=** **{**T**[**17**],** T**[**16**:**0**],** 15'b0**};**

5'd16**:** **{**C**,**Y**}** **=** **{**T**[**16**],** T**[**15**:**0**],** 16'b0**};**

5'd17**:** **{**C**,**Y**}** **=** **{**T**[**15**],** T**[**14**:**0**],** 17'b0**};**

5'd18**:** **{**C**,**Y**}** **=** **{**T**[**14**],** T**[**13**:**0**],** 18'b0**};**

5'd19**:** **{**C**,**Y**}** **=** **{**T**[**13**],** T**[**12**:**0**],** 19'b0**};**

5'd20**:** **{**C**,**Y**}** **=** **{**T**[**12**],** T**[**11**:**0**],** 20'b0**};**

5'd21**:** **{**C**,**Y**}** **=** **{**T**[**11**],** T**[**10**:**0**],** 21'b0**};**

5'd22**:** **{**C**,**Y**}** **=** **{**T**[**10**],** T**[** 9**:**0**],** 22'b0**};**

5'd23**:** **{**C**,**Y**}** **=** **{**T**[** 9**],** T**[** 8**:**0**],** 23'b0**};**

5'd24**:** **{**C**,**Y**}** **=** **{**T**[** 8**],** T**[** 7**:**0**],** 24'b0**};**

5'd25**:** **{**C**,**Y**}** **=** **{**T**[** 7**],** T**[** 6**:**0**],** 25'b0**};**

5'd26**:** **{**C**,**Y**}** **=** **{**T**[** 6**],** T**[** 5**:**0**],** 26'b0**};**

5'd27**:** **{**C**,**Y**}** **=** **{**T**[** 5**],** T**[** 4**:**0**],** 27'b0**};**

5'd28**:** **{**C**,**Y**}** **=** **{**T**[** 4**],** T**[** 3**:**0**],** 28'b0**};**

5'd29**:** **{**C**,**Y**}** **=** **{**T**[** 3**],** T**[** 2**:**0**],** 29'b0**};**

5'd30**:** **{**C**,**Y**}** **=** **{**T**[** 2**],** T**[** 1**:**0**],** 30'b0**};**

5'd31**:** **{**C**,**Y**}** **=** **{**T**[** 1**],** T**[**0**],** 31'b0**};**

**endcase**

5'h0D**:** // SRL

**case(**shamt**)**

5'd0**:** **{**C**,**Y**}** **=** **{**1'b0**,** T**};**

5'd1**:** **{**C**,**Y**}** **=** **{**T**[** 0**],** 1'b0**,** T**[**31**:** 1**]};**

5'd2**:** **{**C**,**Y**}** **=** **{**T**[** 1**],** 2'b0**,** T**[**31**:** 2**]};**

5'd3**:** **{**C**,**Y**}** **=** **{**T**[** 2**],** 3'b0**,** T**[**31**:** 3**]};**

5'd4**:** **{**C**,**Y**}** **=** **{**T**[** 3**],** 4'b0**,** T**[**31**:** 4**]};**

5'd5**:** **{**C**,**Y**}** **=** **{**T**[** 4**],** 5'b0**,** T**[**31**:** 5**]};**

5'd6**:** **{**C**,**Y**}** **=** **{**T**[** 5**],** 6'b0**,** T**[**31**:** 6**]};**

5'd7**:** **{**C**,**Y**}** **=** **{**T**[** 6**],** 7'b0**,** T**[**31**:** 7**]};**

5'd8**:** **{**C**,**Y**}** **=** **{**T**[** 7**],** 8'b0**,** T**[**31**:** 8**]};**

5'd9**:** **{**C**,**Y**}** **=** **{**T**[** 8**],** 9'b0**,** T**[**31**:** 9**]};**

5'd10**:** **{**C**,**Y**}** **=** **{**T**[** 9**],** 10'b0**,** T**[**31**:**10**]};**

5'd11**:** **{**C**,**Y**}** **=** **{**T**[**10**],** 11'b0**,** T**[**31**:**11**]};**

5'd12**:** **{**C**,**Y**}** **=** **{**T**[**11**],** 12'b0**,** T**[**31**:**12**]};**

5'd13**:** **{**C**,**Y**}** **=** **{**T**[**12**],** 13'b0**,** T**[**31**:**13**]};**

5'd14**:** **{**C**,**Y**}** **=** **{**T**[**13**],** 14'b0**,** T**[**31**:**14**]};**

5'd15**:** **{**C**,**Y**}** **=** **{**T**[**14**],** 15'b0**,** T**[**31**:**15**]};**

5'd16**:** **{**C**,**Y**}** **=** **{**T**[**15**],** 16'b0**,** T**[**31**:**16**]};**

5'd17**:** **{**C**,**Y**}** **=** **{**T**[**16**],** 17'b0**,** T**[**31**:**17**]};**

5'd18**:** **{**C**,**Y**}** **=** **{**T**[**17**],** 18'b0**,** T**[**31**:**18**]};**

5'd19**:** **{**C**,**Y**}** **=** **{**T**[**18**],** 19'b0**,** T**[**31**:**19**]};**

5'd20**:** **{**C**,**Y**}** **=** **{**T**[**19**],** 20'b0**,** T**[**31**:**20**]};**

5'd21**:** **{**C**,**Y**}** **=** **{**T**[**20**],** 21'b0**,** T**[**31**:**21**]};**

5'd22**:** **{**C**,**Y**}** **=** **{**T**[**21**],** 22'b0**,** T**[**31**:**22**]};**

5'd23**:** **{**C**,**Y**}** **=** **{**T**[**22**],** 23'b0**,** T**[**31**:**23**]};**

5'd24**:** **{**C**,**Y**}** **=** **{**T**[**23**],** 24'b0**,** T**[**31**:**24**]};**

5'd25**:** **{**C**,**Y**}** **=** **{**T**[**24**],** 25'b0**,** T**[**31**:**25**]};**

5'd26**:** **{**C**,**Y**}** **=** **{**T**[**25**],** 26'b0**,** T**[**31**:**26**]};**

5'd27**:** **{**C**,**Y**}** **=** **{**T**[**26**],** 27'b0**,** T**[**31**:**27**]};**

5'd28**:** **{**C**,**Y**}** **=** **{**T**[**27**],** 28'b0**,** T**[**31**:**28**]};**

5'd29**:** **{**C**,**Y**}** **=** **{**T**[**28**],** 29'b0**,** T**[**31**:**29**]};**

5'd30**:** **{**C**,**Y**}** **=** **{**T**[**29**],** 30'b0**,** T**[**31**:**30**]};**

5'd31**:** **{**C**,**Y**}** **=** **{**T**[**30**],** 31'b0**,** T**[**31**]** **};**

**endcase**

5'h0E**:** // SRA

**case(**shamt**)**

5'd0**:** **{**C**,**Y**}** **=** **{**1'b0**,** T**};**

5'd1**:** **{**C**,**Y**}** **=** **{**T**[** 0**],** T**[**31**],** **{** 1**{**T**[**31**]}},** T**[**30**:** 1**]};**

5'd2**:** **{**C**,**Y**}** **=** **{**T**[** 1**],** T**[**31**],** **{** 2**{**T**[**31**]}},** T**[**30**:** 2**]};**

5'd3**:** **{**C**,**Y**}** **=** **{**T**[** 2**],** T**[**31**],** **{** 3**{**T**[**31**]}},** T**[**30**:** 3**]};**

5'd4**:** **{**C**,**Y**}** **=** **{**T**[** 3**],** T**[**31**],** **{** 4**{**T**[**31**]}},** T**[**30**:** 4**]};**

5'd5**:** **{**C**,**Y**}** **=** **{**T**[** 4**],** T**[**31**],** **{** 5**{**T**[**31**]}},** T**[**30**:** 5**]};**

5'd6**:** **{**C**,**Y**}** **=** **{**T**[** 5**],** T**[**31**],** **{** 6**{**T**[**31**]}},** T**[**30**:** 6**]};**

5'd7**:** **{**C**,**Y**}** **=** **{**T**[** 6**],** T**[**31**],** **{** 7**{**T**[**31**]}},** T**[**30**:** 7**]};**

5'd8**:** **{**C**,**Y**}** **=** **{**T**[** 7**],** T**[**31**],** **{** 8**{**T**[**31**]}},** T**[**30**:** 8**]};**

5'd9**:** **{**C**,**Y**}** **=** **{**T**[** 8**],** T**[**31**],** **{** 9**{**T**[**31**]}},** T**[**30**:** 9**]};**

5'd10**:** **{**C**,**Y**}** **=** **{**T**[** 9**],** T**[**31**],** **{**10**{**T**[**31**]}},** T**[**30**:**10**]};**

5'd11**:** **{**C**,**Y**}** **=** **{**T**[**10**],** T**[**31**],** **{**11**{**T**[**31**]}},** T**[**30**:**11**]};**

5'd12**:** **{**C**,**Y**}** **=** **{**T**[**11**],** T**[**31**],** **{**12**{**T**[**31**]}},** T**[**30**:**12**]};**

5'd13**:** **{**C**,**Y**}** **=** **{**T**[**12**],** T**[**31**],** **{**13**{**T**[**31**]}},** T**[**30**:**13**]};**

5'd14**:** **{**C**,**Y**}** **=** **{**T**[**13**],** T**[**31**],** **{**14**{**T**[**31**]}},** T**[**30**:**14**]};**

5'd15**:** **{**C**,**Y**}** **=** **{**T**[**14**],** T**[**31**],** **{**15**{**T**[**31**]}},** T**[**30**:**15**]};**

5'd16**:** **{**C**,**Y**}** **=** **{**T**[**15**],** T**[**31**],** **{**16**{**T**[**31**]}},** T**[**30**:**16**]};**

5'd17**:** **{**C**,**Y**}** **=** **{**T**[**16**],** T**[**31**],** **{**17**{**T**[**31**]}},** T**[**30**:**17**]};**

5'd18**:** **{**C**,**Y**}** **=** **{**T**[**17**],** T**[**31**],** **{**18**{**T**[**31**]}},** T**[**30**:**18**]};**

5'd19**:** **{**C**,**Y**}** **=** **{**T**[**18**],** T**[**31**],** **{**19**{**T**[**31**]}},** T**[**30**:**19**]};**

5'd20**:** **{**C**,**Y**}** **=** **{**T**[**19**],** T**[**31**],** **{**20**{**T**[**31**]}},** T**[**30**:**20**]};**

5'd21**:** **{**C**,**Y**}** **=** **{**T**[**20**],** T**[**31**],** **{**21**{**T**[**31**]}},** T**[**30**:**21**]};**

5'd22**:** **{**C**,**Y**}** **=** **{**T**[**21**],** T**[**31**],** **{**22**{**T**[**31**]}},** T**[**30**:**22**]};**

5'd23**:** **{**C**,**Y**}** **=** **{**T**[**22**],** T**[**31**],** **{**23**{**T**[**31**]}},** T**[**30**:**23**]};**

5'd24**:** **{**C**,**Y**}** **=** **{**T**[**23**],** T**[**31**],** **{**24**{**T**[**31**]}},** T**[**30**:**24**]};**

5'd25**:** **{**C**,**Y**}** **=** **{**T**[**24**],** T**[**31**],** **{**25**{**T**[**31**]}},** T**[**30**:**25**]};**

5'd26**:** **{**C**,**Y**}** **=** **{**T**[**25**],** T**[**31**],** **{**26**{**T**[**31**]}},** T**[**30**:**26**]};**

5'd27**:** **{**C**,**Y**}** **=** **{**T**[**26**],** T**[**31**],** **{**27**{**T**[**31**]}},** T**[**30**:**27**]};**

5'd28**:** **{**C**,**Y**}** **=** **{**T**[**27**],** T**[**31**],** **{**28**{**T**[**31**]}},** T**[**30**:**28**]};**

5'd29**:** **{**C**,**Y**}** **=** **{**T**[**28**],** T**[**31**],** **{**29**{**T**[**31**]}},** T**[**30**:**29**]};**

5'd30**:** **{**C**,**Y**}** **=** **{**T**[**29**],** T**[**31**],** **{**30**{**T**[**31**]}},** T**[**30**]** **};**

5'd31**:** **{**C**,**Y**}** **=** **{**T**[**30**],** T**[**31**],** **{**31**{**T**[**31**]}}** **};**

**endcase**

5'h1A**:** // Rotate Left

**case(**shamt**)**

5'd0**:** **{**C**,**Y**}** **=** T**;**

5'd1**:** **{**C**,**Y**}** **=** **{**T**[**30**:**0**],** T**[**31**]};**

5'd2**:** **{**C**,**Y**}** **=** **{**T**[**29**:**0**],** T**[**31**:**30**]};**

5'd3**:** **{**C**,**Y**}** **=** **{**T**[**28**:**0**],** T**[**31**:**29**]};**

5'd4**:** **{**C**,**Y**}** **=** **{**T**[**27**:**0**],** T**[**31**:**28**]};**

5'd5**:** **{**C**,**Y**}** **=** **{**T**[**26**:**0**],** T**[**31**:**27**]};**

5'd6**:** **{**C**,**Y**}** **=** **{**T**[**25**:**0**],** T**[**31**:**26**]};**

5'd7**:** **{**C**,**Y**}** **=** **{**T**[**24**:**0**],** T**[**31**:**25**]};**

5'd8**:** **{**C**,**Y**}** **=** **{**T**[**23**:**0**],** T**[**31**:**24**]};**

5'd9**:** **{**C**,**Y**}** **=** **{**T**[**22**:**0**],** T**[**31**:**23**]};**

5'd10**:** **{**C**,**Y**}** **=** **{**T**[**21**:**0**],** T**[**31**:**22**]};**

5'd11**:** **{**C**,**Y**}** **=** **{**T**[**20**:**0**],** T**[**31**:**21**]};**

5'd12**:** **{**C**,**Y**}** **=** **{**T**[**19**:**0**],** T**[**31**:**20**]};**

5'd13**:** **{**C**,**Y**}** **=** **{**T**[**18**:**0**],** T**[**31**:**19**]};**

5'd14**:** **{**C**,**Y**}** **=** **{**T**[**17**:**0**],** T**[**31**:**18**]};**

5'd15**:** **{**C**,**Y**}** **=** **{**T**[**16**:**0**],** T**[**31**:**17**]};**

5'd16**:** **{**C**,**Y**}** **=** **{**T**[**15**:**0**],** T**[**31**:**16**]};**

5'd17**:** **{**C**,**Y**}** **=** **{**T**[**14**:**0**],** T**[**31**:**15**]};**

5'd18**:** **{**C**,**Y**}** **=** **{**T**[**13**:**0**],** T**[**31**:**14**]};**

5'd19**:** **{**C**,**Y**}** **=** **{**T**[**12**:**0**],** T**[**31**:**13**]};**

5'd20**:** **{**C**,**Y**}** **=** **{**T**[**11**:**0**],** T**[**31**:**12**]};**

5'd21**:** **{**C**,**Y**}** **=** **{**T**[**10**:**0**],** T**[**31**:**11**]};**

5'd22**:** **{**C**,**Y**}** **=** **{**T**[**9**:**0**],** T**[**31**:**10**]};**

5'd23**:** **{**C**,**Y**}** **=** **{**T**[**8**:**0**],** T**[**31**:**9**]};**

5'd24**:** **{**C**,**Y**}** **=** **{**T**[**7**:**0**],** T**[**31**:**8**]};**

5'd25**:** **{**C**,**Y**}** **=** **{**T**[**6**:**0**],** T**[**31**:**7**]};**

5'd26**:** **{**C**,**Y**}** **=** **{**T**[**5**:**0**],** T**[**31**:**6**]};**

5'd27**:** **{**C**,**Y**}** **=** **{**T**[**4**:**0**],** T**[**31**:**5**]};**

5'd28**:** **{**C**,**Y**}** **=** **{**T**[**3**:**0**],** T**[**31**:**4**]};**

5'd29**:** **{**C**,**Y**}** **=** **{**T**[**2**:**0**],** T**[**31**:**3**]};**

5'd30**:** **{**C**,**Y**}** **=** **{**T**[**1**:**0**],** T**[**31**:**2**]};**

5'd31**:** **{**C**,**Y**}** **=** **{**T**[**0**],** T**[**31**:**1**]};**

**endcase**

5'h1B**:** // Rotate Right

**case(**shamt**)**

5'd0**:** **{**C**,**Y**}** **=** T**;**

5'd1**:** **{**C**,**Y**}** **=** **{**T**[**0**],** T**[**31**:**1**]};**

5'd2**:** **{**C**,**Y**}** **=** **{**T**[**1**:**0**],** T**[**31**:**2**]};**

5'd3**:** **{**C**,**Y**}** **=** **{**T**[**2**:**0**],** T**[**31**:**3**]};**

5'd4**:** **{**C**,**Y**}** **=** **{**T**[**3**:**0**],** T**[**31**:**4**]};**

5'd5**:** **{**C**,**Y**}** **=** **{**T**[**4**:**0**],** T**[**31**:**5**]};**

5'd6**:** **{**C**,**Y**}** **=** **{**T**[**5**:**0**],** T**[**31**:**6**]};**

5'd7**:** **{**C**,**Y**}** **=** **{**T**[**6**:**0**],** T**[**31**:**7**]};**

5'd8**:** **{**C**,**Y**}** **=** **{**T**[**7**:**0**],** T**[**31**:**8**]};**

5'd9**:** **{**C**,**Y**}** **=** **{**T**[**8**:**0**],** T**[**31**:**9**]};**

5'd10**:** **{**C**,**Y**}** **=** **{**T**[**9**:**0**],** T**[**31**:**10**]};**

5'd11**:** **{**C**,**Y**}** **=** **{**T**[**10**:**0**],** T**[**31**:**11**]};**

5'd12**:** **{**C**,**Y**}** **=** **{**T**[**11**:**0**],** T**[**31**:**12**]};**

5'd13**:** **{**C**,**Y**}** **=** **{**T**[**12**:**0**],** T**[**31**:**13**]};**

5'd14**:** **{**C**,**Y**}** **=** **{**T**[**13**:**0**],** T**[**31**:**14**]};**

5'd15**:** **{**C**,**Y**}** **=** **{**T**[**14**:**0**],** T**[**31**:**15**]};**

5'd16**:** **{**C**,**Y**}** **=** **{**T**[**15**:**0**],** T**[**31**:**16**]};**

5'd17**:** **{**C**,**Y**}** **=** **{**T**[**16**:**0**],** T**[**31**:**17**]};**

5'd18**:** **{**C**,**Y**}** **=** **{**T**[**17**:**0**],** T**[**31**:**18**]};**

5'd19**:** **{**C**,**Y**}** **=** **{**T**[**18**:**0**],** T**[**31**:**19**]};**

5'd20**:** **{**C**,**Y**}** **=** **{**T**[**19**:**0**],** T**[**31**:**20**]};**

5'd21**:** **{**C**,**Y**}** **=** **{**T**[**20**:**0**],** T**[**31**:**21**]};**

5'd22**:** **{**C**,**Y**}** **=** **{**T**[**21**:**0**],** T**[**31**:**22**]};**

5'd23**:** **{**C**,**Y**}** **=** **{**T**[**22**:**0**],** T**[**31**:**23**]};**

5'd24**:** **{**C**,**Y**}** **=** **{**T**[**23**:**0**],** T**[**31**:**24**]};**

5'd25**:** **{**C**,**Y**}** **=** **{**T**[**24**:**0**],** T**[**31**:**25**]};**

5'd26**:** **{**C**,**Y**}** **=** **{**T**[**25**:**0**],** T**[**31**:**26**]};**

5'd27**:** **{**C**,**Y**}** **=** **{**T**[**26**:**0**],** T**[**31**:**27**]};**

5'd28**:** **{**C**,**Y**}** **=** **{**T**[**27**:**0**],** T**[**31**:**28**]};**

5'd29**:** **{**C**,**Y**}** **=** **{**T**[**28**:**0**],** T**[**31**:**29**]};**

5'd30**:** **{**C**,**Y**}** **=** **{**T**[**29**:**0**],** T**[**31**:**30**]};**

5'd31**:** **{**C**,**Y**}** **=** **{**T**[**30**:**0**],** T**[**31**]};**

**endcase**

**endcase**

**endmodule**

## ALU MIPS

`timescale 1ns **/** 1ps

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* C E C S 4 4 0 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* File Name: MIPS\_32.v

\* Project: Lab\_Assignment\_1

\* Designer: Chanartip Soonthornwan, Jonathan Shihata

\* Email: Chanartip.Soonthornwan@gmail.com, JonnyShihata@gmail.com

\* Rev. No.: Version 1.0

\* Rev. Date: Current Rev. Date 9/3/2017

\*

\* Purpose: Processes two 32-bit input(S and T) according to

\* Function Select(FS) and then return a 32-bit result,

\* Overflow flag(V), and Carry flag(C).

\*

\* Notes: For an OP code that does not affect overflow flag (V) or

\* Carry flag (C) will output as 'x'

\* U - unsigned

\* I - Immediate

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**module** MIPS\_32**(**S**,** T**,** FS**,** Y**,** V**,** C**);**

**input** **[**31**:**0**]** S**,** T**;** // Input Registers

**input** **[** 4**:**0**]** FS**;** // Function Select(OP code)

**output** **reg** **[**31**:**0**]** Y**;** // Output below 32th bit

**output** **reg** V**,** C**;** // Overflow and Carry Flags

**integer** int\_s**,** int\_t**;** // parse interger of S and T

// Look-up table for Function Select(FS)

**parameter** PASS\_S **=** 5'h00**,** AND **=** 5'h08**,** INC **=** 5'h0F**,**

PASS\_T **=** 5'h01**,** OR **=** 5'h09**,** DEC **=** 5'h10**,**

ADD **=** 5'h02**,** XOR **=** 5'h0A**,** INC4 **=** 5'h11**,**

ADDU **=** 5'h03**,** NOR **=** 5'h0B**,** DEC4 **=** 5'h12**,**

SUB **=** 5'h04**,** SLL **=** 5'h0C**,** ZEROS **=** 5'h13**,**

SUBU **=** 5'h05**,** SRL **=** 5'h0D**,** ONES **=** 5'h14**,**

SLT **=** 5'h06**,** SRA **=** 5'h0E**,** SP\_INIT **=** 5'h15**,**

SLTU **=** 5'h07**,** ANDI **=** 5'h16**,**

MUL **=** 5'h1E**,** ORI **=** 5'h17**,**

DIV **=** 5'h1F**,** XORI **=** 5'h18**,**

LUI **=** 5'h19**;**

// Parse S and T to integer

**always@(\*)** **begin**

int\_s **<=** S**;**

int\_t **<=** T**;**

**end**

// Combination Logic for both Arithmetic and Logic Operations

**always@(\*)** **begin**

**case(**FS**)**

PASS\_S**:** **{**V**,**C**,**Y**}** **=** **{**2'bxx**,** S**};** // Pass S

PASS\_T**:** **{**V**,**C**,**Y**}** **=** **{**2'bxx**,** T**};** // Pass T

ADD **:** **begin** // Adding

**{**C**,**Y**}** **=** S **+** T**;**

V **=** **{** S**[**31**]** **&** T**[**31**]** **&** **~**Y**[**31**]** **}|** // (+) + (+) = (-)

**{** **~**S**[**31**]** **&** **~**T**[**31**]** **&** Y**[**31**]** **};** // (-) + (-) = (+)

**end**

ADDU **:** **begin** // Adding Unsigned

**{**C**,**Y**}** **=** S **+** T**;**

V **=** C**;**

**end**

SUB **:** **begin** // Subtracting

**{**C**,**Y**}** **=** S **-** T**;**

V **=** **{** S**[**31**]** **&** **~**T**[**31**]** **&** **~**Y**[**31**]** **}|** // (+) - (-) = (-)

**{** **~**S**[**31**]** **&** T**[**31**]** **&** Y**[**31**]** **};** // (-) - (+) = (+)

**end**

SUBU **:** **begin** // Subtracting Unsigned

**{**C**,**Y**}** **=** S **-** T**;**

V **=** C**;**

**end**

SLT **:** **begin** // Set Less Than

**{**V**,**C**}** **=** 2'bxx**;**

Y **=** int\_s **<** int\_t**;** // comparing signed integers

**end**

SLTU **:** **begin** // Set Less Than Unsigned

**{**V**,**C**}** **=** 2'bxx**;**

Y **=** S **<** T**;**  // comparing bits

**end**

AND **:** **{**V**,**C**,**Y**}** **=** **{**2'bxx**,** S **&** T**};** // And

OR **:** **{**V**,**C**,**Y**}** **=** **{**2'bxx**,** S **|** T**};** // Or

XOR **:** **{**V**,**C**,**Y**}** **=** **{**2'bxx**,** S **^** T**};** // Xor

NOR **:** **{**V**,**C**,**Y**}** **=** **{**2'bxx**,** **~(**S **|** T**)};** // Nor

SLL **:** **begin** // Shift Left Logic

**{**V**,**C**}** **=** **{**1'bx**,** T**[**31**]};**

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

**end**

SRL **:** **begin** // Shift Right Logic

**{**V**,**C**}** **=** **{**1'bx**,** T**[**0**]};**

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

**end**

SRA **:** **begin** // Shift Right Arithmetic

**{**V**,**C**}** **=** **{**1'bx**,** T**[**0**]};**

Y **=** **{**T**[**31**],** T**[**31**:**1**]};**

**end**

ANDI **:** **{**V**,**C**,**Y**}** **=** **{**2'bxx**,** S **&** **{**16'b0**,** T**[**15**:**0**]}};**

ORI **:** **{**V**,**C**,**Y**}** **=** **{**2'bxx**,** S **|** **{**16'b0**,** T**[**15**:**0**]}};**

XORI **:** **{**V**,**C**,**Y**}** **=** **{**2'bxx**,** S **^** **{**16'b0**,** T**[**15**:**0**]}};**

LUI **:** **{**V**,**C**,**Y**}** **=** **{**2'bxx**,** T**[**15**:**0**],** 16'b0**};** // Load Upper Immediate

INC **:** **begin** // Increment (by one)

**{**C**,**Y**}** **=** S **+** 1**;**

V **=** **{** S**[**31**]** **&** **~**Y**[**31**]** **}|** // (+) + 1 = (-)

**{** **~**S**[**31**]** **&** Y**[**31**]** **};** // (-) + 1 = (+)

**end**

DEC **:** **begin** // Decrement (by one)

**{**C**,**Y**}** **=** S **-** 1**;**

V **=** **{** S**[**31**]** **&** **~**Y**[**31**]** **}|** // (+) - 1 = (-)

**{** **~**S**[**31**]** **&** Y**[**31**]** **};** // (-) - 1 = (+)

**end**

INC4 **:** **begin** // Increment (by four)

**{**C**,**Y**}** **=** S **+** 4**;**

V **=** **{** S**[**31**]** **&** **~**Y**[**31**]** **}|** // (+) + 4 = (-)

**{** **~**S**[**31**]** **&** Y**[**31**]** **};** // (-) + 4 = (+)

**end**

DEC4 **:** **begin** // Decrement (by four)

**{**C**,**Y**}** **=** S **-** 4**;**

V **=** **{** S**[**31**]** **&** **~**Y**[**31**]** **}|** // (+) - 4 = (-)

**{** **~**S**[**31**]** **&** Y**[**31**]** **};** // (-) - 4 = (+)

**end**

ZEROS **:** **{**V**,**C**,**Y**}** **=** **{**2'bxx**,** 32'h0**};** // Set to Zero

ONES **:** **{**V**,**C**,**Y**}** **=** **{**2'bxx**,** 32'hFFFFFFFF**};** // Set to One

SP\_INIT**:** **{**V**,**C**,**Y**}** **=** **{**2'bxx**,** 32'h3FC**};** // Stack Pointer Init

**default:** **{**V**,**C**,**Y**}** **=** **{**2'bxx**,** S**};** // Pass S

**endcase**

**end**

**endmodule**

## ALU Multiplier

`timescale 1ns **/** 1ps

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* C E C S 4 4 0 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* File Name: MPY\_32.v

\* Project: Lab\_Assignment\_1

\* Designer: Chanartip Soonthornwan, Jonathan Shihata

\* Email: Chanartip.Soonthornwan@gmail.com, JonnyShihata@gmail.com

\* Rev. No.: Version 1.0

\* Rev. Date: Current Rev. Date 9/3/2017

\*

\* Purpose: Multiply two 32-bit inputs and return a 64-bit results.

\*

\* Notes: Two inputs are needed to be casted as integers before

\* being operated.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**module** MPY\_32**(**S**,** T**,** Y**);**

**input** **[**31**:**0**]** S**,** T**;** // 32-bit inputs

**output** **reg** **[**63**:**0**]** Y**;** // 64-bit result

**integer** int\_s**,** int\_t**;** // integer for each inputs

// Cast S and T into integers

**always@(\*)** **begin**

int\_s **<=** S**;**

int\_t **<=** T**;**

**end**

// Multiply the two integers

**always@(\*)**

Y **=** int\_s **\*** int\_t**;**

**endmodule**

## ALU Divider

`timescale 1ns **/** 1ps

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* C E C S 4 4 0 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* File Name: DIV\_32.v

\* Project: Lab\_Assignment\_1

\* Designer: Chanartip Soonthornwan, Jonathan Shihata

\* Email: Chanartip.Soonthornwan@gmail.com, JonnyShihata@gmail.com

\* Rev. No.: Version 1.0

\* Rev. Date: Current Rev. Date 9/3/2017

\*

\* Purpose: Divide two 32-bit inputs and return remainder and qoutian

\*

\* Notes: Two inputs are needed to be casted as integers.

\* division is qoutationg, or p reslt.

\* modulus has

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**module** DIV\_32**(**S**,** T**,** rem**,** qout**);**

**input** **[**31**:**0**]** S**,** T**;** // 32-bit inputs

**output** **reg** **[**31**:**0**]** rem**,** qout**;** // Remainder and Qoutian outputs

**integer** int\_s**,** int\_t**;** // Integer of each inputs

// Parse S and T into integers

**always@(\*)** **begin**

int\_s **<=** S**;**

int\_t **<=** T**;**

**end**

// Divide the two integers

**always@(\*)** **begin**

qout **=** int\_s **/** int\_t**;** // Quotient

rem **=** int\_s **%** int\_t**;** // Remainder

**end**

**endmodule**

## Instruction Unit

`timescale 1ns **/** 1ps

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* C E C S 4 4 0 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* File Name: Instruction\_Unit.v

\* Project: Final Project

\* Designer: Chanartip Soonthornwan, Jonathan Shihata

\* Email: Chanartip.Soonthornwan@gmail.com, JonnyShihata@gmail.com

\* Rev. No.: Version 1.0

\* Rev. Date: Date 10/10/2017

\*

\* Rev. No.1: Version 1.1

\* Rev. Date: Date 10/24/2017

\* update: Get rid of [31:0] D\_in, and Add pc\_sel for PC\_MUX

\*

\* Rev. No.2: Version 1.2

\* Rev. Date: Current Rev. Date 11/19/2017

\* update: Adjusts Instruction Memory address pin to only

\* get 12 bits from PC\_out since the project's memory is small

\* and there is no need for using more than 12 bits.

\*

\* Purpose: Instruction Unit(CPU\_IU) is a memory holding instructions

\* could be accessed by an address from the Program Counter(PC),

\* then sending the called instruction through data output(D\_OUT).

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**module** Instruction\_Unit**(**

**input** clk**,** reset**,** // On-board clock and reset signal

**input** pc\_ld**,** pc\_inc**,** ir\_ld**,** // PC and IR load, and PC increment

**input** im\_cs**,** im\_wr**,** im\_rd**,** // Instruction Memory (im)

// Chip Select, Write, and Read

**input** **[**1**:**0**]** pc\_sel**,** // pc select for PC\_MUX

**input** **[**31**:**0**]** PC\_in**,** // PC Input

**output** **[**31**:**0**]** PC\_out**,** // PC Output

**output** **[**31**:**0**]** IR\_out**,** SE\_16 // Instruction Output

// and Sign-Extension output

**);**

**wire** **[**31**:**0**]** IM\_out**;** // Instruction Memory Output

**wire** **[**31**:**0**]** pc\_mux**;** // PC\_MUX for pc\_in

//PC\_MUX

//if pc\_sel = 0, select PC\_in (pc+4)

//pc\_sel = 1, select Jump instruction PC <- {pc[31:28], 26bit-imme, 00}

//pc\_sel = 2, select Branch instruction PC <- PC + {sign\_ext(ir[15:0]),00}

**assign** pc\_mux **=** **(**pc\_sel **==** 2'b01**)** **?** **{**PC\_out**[**31**:**28**],**IR\_out**[**25**:**0**],**2'b00**}** **:**// Jump

**(**pc\_sel **==** 2'b10**)** **?** **{**PC\_out **+** **{**SE\_16**[**29**:**0**],**2'b00**}}**

**:**// Branch

PC\_in**;**

// PC+4

// Program Counter(PC)

pc PC **(.**clk**(**clk**),** // On-board clock

**.**reset**(**reset**),** // Reset

**.**pc\_ld**(**pc\_ld**),** // Load

**.**pc\_inc**(**pc\_inc**),** // Increment

**.**pc\_in**(**pc\_mux**),** // Input

**.**pc\_out**(**PC\_out**)** // Output

**);**

// Instrcution Memory

Memory IM **(.**clk**(**clk**),** // On-board clock

**.**cs**(**im\_cs**),** // Chip-select

**.**wr**(**im\_wr**),** // Write Enable

**.**rd**(**im\_rd**),** // Read Enable

**.**Addr**({**20'b0**,**PC\_out**[**11**:**0**]}),**// Address from PC

**.**D\_In**(**32'b0**),** // Deasserted

**.**D\_Out**(**IM\_out**)** // Instruction output

**);**

// Instruction Register

//

reg32\_w\_ld IR **(.**clk**(**clk**),** // On-board clock

**.**reset**(**reset**),** // Reset

**.**load**(**ir\_ld**),** // Load

**.**d**(**IM\_out**),** // Data input

**.**q**(**IR\_out**)** // Data output

**);**

// Sign Extend 16 bits

// Takes 15th bit of IR\_out, creates 16-bit of the 15th bit,

// then concatenates with lower bits of IR\_out

**assign** SE\_16 **=** **{** **{**16**{**IR\_out**[**15**]}},** IR\_out**[**15**:**0**]};**

**endmodule**

## IDP

`timescale 1ns **/** 1ps

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* C E C S 4 4 0 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* File Name: IDP.v

\* Project: Final Project

\* Designer: Chanartip Soonthornwan, Jonathan Shihata

\* Email: Chanartip.Soonthornwan@gmail.com, JonnyShihata@gmail.com

\* Rev. No.: Version 1.1

\* Rev. Date: Date 10/6/2017

\* Update: Add four registers as scratch register;

\* RS, RT, ALU\_out\_r, D\_in.

\*

\* Rev. No.: Version 1.2

\* Rev. Date: Rev. 1.2 Date 10/16/2017

\* Update: Added DA\_Sel to select either D\_Addr or T\_Addr

\*

\* Rev. No.: Version 1.3

\* Rev. Date: 1.3 Date 10/24/2017

\* Update: Added another bit for DA\_Sel to select

\* D\_Addr, T\_Addr, $ra(return Address), or $sp(stack Pointer)

\*

\* Rev. No.: Version 1.4

\* Rev. Date: Current Rev. 1.3 Date 11/21/2017

\* Update: Added SP\_Sel, S\_Sel, sp\_flags\_in and sp\_flags\_out for

\* performing Stack Memory

\*

\* Purpose: Integer Data Path (IDP) performs arithmetics calculation

\* from 32-bit data inside a register file and output the result.

\*

\* Notes: IDP receives control from a control unit(CU) to control

\* the data flow (data path) in IDP by sending source

\* register addresses to the register file and execute the

\* arithmetic operation in Arithmetic Logic Unit (ALU), then

\* selects output by Y-MUX.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**module** IDP**(**

**input** clk**,** reset**,** // On-board clock and reset signal

**input** D\_En**,** HILO\_ld**,** // Load enable for regfiles

**input** **[**1**:**0**]** DA\_Sel**,** // DA\_MUX

**input** **[**1**:**0**]** T\_Sel**,** // T-MUX select

**input** **[**2**:**0**]** Y\_Sel**,** // Y-MUX select

**input** **[**4**:**0**]** FS**,** // ALU function select

**input** **[**4**:**0**]** shamt**,** // Shifting amount

**input** **[**4**:**0**]** D\_Addr**,** // regfile Write address

**input** **[**4**:**0**]** S\_Addr**,** T\_Addr**,** // regfile Source addresses

**input** **[**31**:**0**]** DT**,** DY**,** // Data inputs

**input** **[**31**:**0**]** pc\_in**,** // Program Counter input

**input** S\_Sel**,** SP\_Sel**,** // S\_MUX and Stack Pointer Select

**input** **[**4**:**0**]** sp\_flags\_in**,** // input stack flags

**output** **[**4**:**0**]** sp\_flags\_out**,** // output stack flags

**output** C**,** V**,** N**,** Z**,** // Status flags from ALU

**output** **[**31**:**0**]** ALU\_OUT**,** D\_OUT // Outputs

**);**

**wire** **[**31**:**0**]** S**,** T**;** // Wires S to ALU and T to T-MUX

**wire** **[**31**:**0**]** rt\_in**;** // A wire from T-MUX to RT

**wire** **[**31**:**0**]** Y\_hi**,** Y\_lo**;** // Wires to HILO regfile and Y-MUX

**wire** **[**31**:**0**]** hi**,** lo**;** // Outputs from HILO regfile

**wire** **[**31**:**0**]** rs**,** rt**;** // Scratch

**wire** **[**31**:**0**]** alu\_out**,** d\_in**;** // registers wires

**wire** **[**4**:**0**]** DA\_Out**;** // Wire to D\_Addr

**wire** **[**4**:**0**]** SP\_MUX\_Out**;** // SP\_MUX output to regfile S\_Addr

**wire** **[**31**:**0**]** S\_MUX\_Out**;** // S\_MUX output to ALU S input

// Register File

// a memory unit for CPU

regfile32 regfile **(.**clk**(**clk**),** // On-board clock

**.**reset**(**reset**),** // Reset signal

**.**D\_En**(**D\_En**),** // Load Enable

**.**D\_Addr**(**DA\_Out**),** // Writing address

**.**S\_Addr**(**SP\_MUX\_Out**),**// Source S address

**.**T\_Addr**(**T\_Addr**),** // Source T address

**.**D\_in**(**ALU\_OUT**),** // Data input

**.**S**(**S**),** // S output

**.**T**(**T**)** // T output

**);**

// DA\_MUX

// if DA\_Sel = 1, choose T\_Addr(rt)

// if DA\_Sel = 2, choose $ra (register 31)

// if DA\_Sel = 3, choose $sp (register 29)

// otherwise, choose D\_Addr(rd) by default

**assign** DA\_Out **=** **(**DA\_Sel **==** 2'b01**)?** T\_Addr**:** // $rt

**(**DA\_Sel **==** 2'b10**)?** 5'h1F**:** // 31 for $ra

**(**DA\_Sel **==** 2'b11**)?** 5'h1D**:** // 29 for $sp

D\_Addr**;** // $rd default

// SP\_MUX

// selecting either S\_Addr from Instuction or

// Stack Pointer's address($sp at 5'h1D)

**assign** SP\_MUX\_Out **=** **(**SP\_Sel**)?** 5'h1D**:** S\_Addr**;**

// S\_MUX

// selecting either rs or alu\_out

// and load it into S input of ALU

**assign** S\_MUX\_Out **=** **(**S\_Sel**)?** alu\_out**:** rs**;**

// T-MUX

// selecting an output to input T of ALU

// if T\_sel is 1, assign DT (SE\_16 from Instruction Unit)

// 2, PC\_in

// 3, flags from Data Memory before Interrupted

// otherwise assign T from regfile32

**assign** rt\_in **=** **(**T\_Sel**==**2'b01**)** **?** DT**:**

**(**T\_Sel**==**2'b10**)** **?** pc\_in**:**

**(**T\_Sel**==**2'b11**)** **?** **{**27'b0**,** sp\_flags\_in**}:** // status flags from mem

T**;**

// Arithmetic Logic Unit (ALU)

// performs Arithmetic and logic calculation

// base on Function Select(FS)

ALU\_32 alu **(.**S**(**S\_MUX\_Out**),** // S input

**.**T**(**rt**),** // T input

**.**FS**(**FS**),** // Function Select(OPcode)

**.**shamt**(**shamt**),** // Shift Amount

**.**Y\_hi**(**Y\_hi**),** // upper part of 64-bit result

**.**Y\_lo**(**Y\_lo**),** // lower part of 64-bit resutl

**.**C**(**C**),** // Carry flag

**.**V**(**V**),** // Overflow flag

**.**N**(**N**),** // Negative flag

**.**Z**(**Z**)** // Zero flag

**);**

// Hi-Lo register

// holds result from multiplication

// and division.

// clk - On-board clock

// reset - Reset signal

// load - Load enable

// d - upper part of 64-bit input

// q - lower part of 64-bit input

// clk, reset, load, d, q

reg32\_w\_ld HI **(**clk**,** reset**,** HILO\_ld**,** Y\_hi**,** hi**),**

LO **(**clk**,** reset**,** HILO\_ld**,** Y\_lo**,** lo**);**

// Scratch register

// holding value for the next clock

// clk - On-board clock

// reset - Reset signal

// d - 32-bit input

// q - 32-bit output

// clk, reset, d, q

reg32\_no\_ld RS **(**clk**,** reset**,** S**,** rs**),**

RT **(**clk**,** reset**,** rt\_in**,** rt**),**

ALU\_out **(**clk**,** reset**,** Y\_lo**,** alu\_out**),**

D\_in **(**clk**,** reset**,** DY**,** d\_in**);**

// Wire flags status to MCU

// when IDP got flags from Memory(D\_in)

**assign** sp\_flags\_out **=** d\_in**[**4**:**0**];**

// Y-MUX

// selecting output (ALU\_OUT) from Y\_Sel

**assign** ALU\_OUT **=** **(**Y\_Sel **==** 3'b001**)** **?** hi**:** // upper part of 64-bit result

**(**Y\_Sel **==** 3'b010**)** **?** lo**:** // lower part of 64-bit result

**(**Y\_Sel **==** 3'b011**)** **?** d\_in**:** // Data Input

**(**Y\_Sel **==** 3'b100**)** **?** pc\_in**:** // Program Counter Input

alu\_out**;** // Pass alu\_out by default

// Assign D\_OUT from the output of RT register

**assign** D\_OUT **=** rt**;**

**endmodule**

## Data Memory

`timescale 1ns **/** 1ps

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* C E C S 4 4 0 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* File Name: Data\_Memory.v

\* Project: Final Project

\* Designer: Chanartip Soonthornwan, Jonathan Shihata

\* Email: Chanartip.Soonthornwan@gmail.com, JonnyShihata@gmail.com

\* Rev. No.: Version 1.0

\* Rev. Date: Current Rev. Date 10/7/2017

\*

\* Purpose: a 4096x8 byte addressable memory in big endian format

\* storing data and instructions that could be accessed

\* by addressing Address(Addr) as a base of the location

\* of the memory and the add 1,2,3 for the rest of the base

\* (see details at Read section).

\*

\* Notes: For the purposes of this lab Addr will be used only

\* 12 bits out of 32 bits.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**module** Data\_Memory**(**clk**,** dm\_cs**,** dm\_wr**,** dm\_rd**,** Addr**,** DM\_In**,** DM\_Out**);**

**input** clk**;** // clock signal

**input** dm\_cs**,** dm\_wr**,** dm\_rd**;** // Data memory "ChipSelect", "Write", "Read"

**input** **[**31**:**0**]** Addr**;** // Address to the memory

**input** **[**31**:**0**]** DM\_In**;** // Data input

**output** **[**31**:**0**]** DM\_Out**;** // Data output

**reg** **[**7**:**0**]** Mem **[**0**:**4095**];** // 4Kx8 array of registers

// Read data from Memory

// Addr is the ALU\_out which contained

// register in-direct address.

// for instance, Addr <- ALU\_out\_r(0x0000\_000F) // register 15

// Addr is located at register 15 in this memory

// and then read the value of base(reg15)

// which are 0x03C to 0x03F [4 addresses at the time.]

// If the memory is not being read, D\_Out will output HighImpedance(z).

**assign** DM\_Out **=** **(**dm\_cs **&** dm\_rd **&** **!**dm\_wr**)?** **{**Mem**[**Addr**+**0**],**

Mem**[**Addr**+**1**],**

Mem**[**Addr**+**2**],**

Mem**[**Addr**+**3**]}** **:** 32'hz**;**

// Write data on Memory

// Writing data on memory is synchronous with the clock(clk)

// and only if Chip Select(cs) and Write Enable(wr) are HIGH.

// Otherwise, the memory cannot be written.

**always@(posedge** clk**)**

**if(**dm\_cs **&** dm\_wr **&** **!**dm\_rd**)** // Write Data Input into the Memory

**{**Mem**[**Addr**+**0**],**

Mem**[**Addr**+**1**],**

Mem**[**Addr**+**2**],**

Mem**[**Addr**+**3**]}** **<=** DM\_In**;**

**else** **begin** // Keep the same value

Mem**[**Addr**+**0**]** **<=** Mem**[**Addr**+**0**];**

Mem**[**Addr**+**1**]** **<=** Mem**[**Addr**+**1**];**

Mem**[**Addr**+**2**]** **<=** Mem**[**Addr**+**2**];**

Mem**[**Addr**+**3**]** **<=** Mem**[**Addr**+**3**];**

**end**

**endmodule**

## IO Memory

`timescale 1ns **/** 1ps

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* C E C S 4 4 0 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* File Name: IO\_Memory.v

\* Project: Final Project

\* Designer: Chanartip Soonthornwan, Jonathan Shihata

\* Email: Chanartip.Soonthornwan@gmail.com, JonnyShihata@gmail.com

\* Rev. No.: Version 1.0

\* Rev. Date: Current Rev. Date 10/7/2017

\*

\* Purpose: a 4096x8 byte addressable memory in big endian format

\* storing data and instructions that could be accessed

\* by addressing Address(Addr) as a base of the location

\* of the memory and the add 1,2,3 for the rest of the base

\* (see details at Read section).

\*

\* Notes: For the purposes of this lab Addr will be used only

\* 12 bits out of 32 bits.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**module** IO\_Memory**(**clk**,** cs**,** wr**,** rd**,** int\_r**,** int\_ack**,** Addr**,** IO\_In**,** IO\_Out**);**

**input** clk**;** // clock signal

**input** cs**,** wr**,** rd**;** // "ChipSelect", "Write", "Read"

**input** int\_ack**;** // interrupt acknowledge output

**output** **reg** int\_r**;** // interrupt request input

**input** **[**31**:**0**]** Addr**;** // Address to the memory

**input** **[**31**:**0**]** IO\_In**;** // Data input

**output** **[**31**:**0**]** IO\_Out**;** // Data output

**reg** **[**7**:**0**]** Mem **[**0**:**4095**];** // 4Kx8 array of registers

// Send interrupt request to CPU

// after 1000/10 ns = 100 clock cycles

// Even though interrupt request is set,

// if instruction SETIE is not called,

// CPU will not be interrupted.

**initial** **begin**

int\_r**=**0**;**

**#**1000 int\_r**=**1**;**

**@(posedge** int\_ack**)** int\_r**=**0**;**

**end**

// Read data from Memory

// Addr is the ALU\_out which contained

// register in-direct address.

// for instance, Addr <- ALU\_out\_r(0x0000\_000F) // register 15

// Addr is located at register 15 in this memory

// and then read the value of base(reg15)

// which are 0x03C to 0x03F [4 addresses at the time.]

// If the memory is not being read, D\_Out will output HighImpedance(z).

**assign** IO\_Out **=** **(**cs **&** rd **&** **!**wr**)?** **{**Mem**[**Addr**+**0**],**

Mem**[**Addr**+**1**],**

Mem**[**Addr**+**2**],**

Mem**[**Addr**+**3**]}** **:** 32'hz**;**

// Write data on Memory

// Writing data on memory is synchronous with the clock(clk)

// and only if Chip Select(cs) and Write Enable(wr) are HIGH.

// Otherwise, the memory cannot be written.

**always@(posedge** clk**)**

**if(**cs **&** wr **&** **!**rd**)** // Write Data Input into the Memory

**{**Mem**[**Addr**+**0**],**

Mem**[**Addr**+**1**],**

Mem**[**Addr**+**2**],**

Mem**[**Addr**+**3**]}** **<=** IO\_In**;**

**else** **begin** // Keep the same value

Mem**[**Addr**+**0**]** **<=** Mem**[**Addr**+**0**];**

Mem**[**Addr**+**1**]** **<=** Mem**[**Addr**+**1**];**

Mem**[**Addr**+**2**]** **<=** Mem**[**Addr**+**2**];**

Mem**[**Addr**+**3**]** **<=** Mem**[**Addr**+**3**];**

**end**

**endmodule**

## PC

`timescale 1ns **/** 1ps

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* C E C S 4 4 0 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* File Name: pc.v

\* Project: Lab\_Assignment\_5

\* Designer: Chanartip Soonthornwan, Jonathan Shihata

\* Email: Chanartip.Soonthornwan@gmail.com, JonnyShihata@gmail.com

\* Rev. No.: Version 1.0

\* Rev. Date: Current Rev. Date 10/10/2017

\*

\* Purpose: Program Counter(PC) register holds an address of Instruction

\* Memory(CPU\_IU). As pc\_ld is high, the address is loaded

\* with data input(pc\_in), and the address is increased by 4

\* (or move to the next memory location) as pc\_inc is high.

\* Otherwise, there is no changed.

\*

\* Notes: If both pc\_ld and pc\_inc are high, there is no changed.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**module** pc**(**clk**,** reset**,** pc\_ld**,** pc\_inc**,** pc\_in**,** pc\_out**);**

**input** clk**,** reset**;** // on-board clock and reset

**input** pc\_ld**,** pc\_inc**;** // Load and Increment signals

**input** **[**31**:**0**]** pc\_in**;** // Address in

**output** **reg** **[**31**:**0**]** pc\_out**;** // Address out

**always** **@** **(posedge** clk**,** **posedge** reset**)**

**if(**reset**)** pc\_out **<=** 32'b0**;** // reset

**else** // Update pc\_out on active edge clk

**case({**pc\_ld**,** pc\_inc**})**

2'b01**:** pc\_out **<=** pc\_out **+** 4**;** // increment

2'b10**:** pc\_out **<=** pc\_in**;** // load

**default:** pc\_out **<=** pc\_out**;** // default

**endcase**

**endmodule**

## Memory

`timescale 1ns **/** 1ps

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* C E C S 4 4 0 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* File Name: Memory.v

\* Project: Lab\_Assignment\_5

\* Designer: Chanartip Soonthornwan, Jonathan Shihata

\* Email: Chanartip.Soonthornwan@gmail.com, JonnyShihata@gmail.com

\* Rev. No.: Version 1.0

\* Rev. Date: Current Rev. Date 10/11/2017

\*

\* Purpose: a 4096x8 byte addressable memory in big endian format

\* storing data and instructions that could be accessed

\* by addressing Address(Addr) as a base of the location

\* of the memory and the add 1,2,3 for the rest of the base

\* (see details at Read section).

\*

\* Notes: For the purposes of this lab Addr will be used only

\* 12 bits out of 32 bits.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**module** Memory**(**clk**,** cs**,** wr**,** rd**,** Addr**,** D\_In**,** D\_Out**);**

**input** clk**;** // clock signal

**input** cs**,** wr**,** rd**;** // Data memory "ChipSelect", "Write", "Read"

**input** **[**31**:**0**]** Addr**;** // Address to the memory

**input** **[**31**:**0**]** D\_In**;** // Data input

**output** **[**31**:**0**]** D\_Out**;** // Data output

**reg** **[**7**:**0**]** Mem **[**0**:**4095**];** // 4Kx8 array of registers

// Read data from Memory

// Addr is the ALU\_out which contained

// register in-direct address.

// for instance, Addr <- ALU\_out\_r(0x0000\_000F) // register 15

// Addr is located at register 15 in this memory

// and then read the value of base(reg15)

// which are 0x03C to 0x03F [4 addresses at the time.]

// If the memory is not being read, D\_Out will output HighImpedance(z).

**assign** D\_Out **=** **(**cs **&** rd **&** **!**wr**)?** **{**Mem**[**Addr**+**0**],**

Mem**[**Addr**+**1**],**

Mem**[**Addr**+**2**],**

Mem**[**Addr**+**3**]}** **:** 32'bz**;**

// Write data on Memory

// Writing data on memory is synchronous with the clock(clk)

// and only if Chip Select(cs) and Write Enable(wr) are HIGH.

// Otherwise, the memory cannot be written.

**always@(posedge** clk**)**

**if(**cs **&** wr **&** **!**rd**)** // Write Data Input into the Memory

**{**Mem**[**Addr**+**0**],**

Mem**[**Addr**+**1**],**

Mem**[**Addr**+**2**],**

Mem**[**Addr**+**3**]}** **=** D\_In**;**

**else** **begin** // Keep the same value

Mem**[**Addr**+**0**]** **=** Mem**[**Addr**+**0**];**

Mem**[**Addr**+**1**]** **=** Mem**[**Addr**+**1**];**

Mem**[**Addr**+**2**]** **=** Mem**[**Addr**+**2**];**

Mem**[**Addr**+**3**]** **=** Mem**[**Addr**+**3**];**

**end**

**endmodule**

## 32-bit loadable register

`timescale 1ns **/** 1ps

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* C E C S 4 4 0 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* File Name: reg32\_w\_ld.v

\* Project: Lab\_Assignment\_1

\* Designer: Chanartip Soonthornwan, Jonathan Shihata

\* Email: Chanartip.Soonthornwan@gmail.com, JonnyShihata@gmail.com

\* Rev. No.: Version 1.0

\* Rev. Date: Date 9/29/2017

\*

\* Rev. No.: Version 2.0

\* Rev. Date: Current Rev. Date 10/10/2017

\* Notes: - change port list to be reuseable as 32-bit register module

\*

\* Purpose: A register to hold 32-bit result until load enable(load)

\* allows data input(d) to overwritten previous value.

\*

\* Notes: This register operates on the active edge of the clock

\* signal input (clk). If the load enable (load) is HIGH,

\* the register will hold the new data input instead.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**module** reg32\_w\_ld **(**clk**,** reset**,** load**,** d**,** q**);**

**input** clk**,** reset**;** // on-board clock, and reset signal

**input** load**;** // load enable

**input** **[**31**:**0**]** d**;** // data inputs

**output** **reg** **[**31**:**0**]** q**;** // data outputs

**always** **@** **(posedge** clk**,** **posedge** reset**)**

**if(**reset**)** q **<=** 32'b0**;** **else** // reset

**if(**load**)** q **<=** d**;** // assign new value

**else** q **<=** q**;** // keep previous value

**endmodule**

## 32-bit Non-Loadable Regiter

`timescale 1ns **/** 1ps

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* C E C S 4 4 0 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* File Name: reg32\_no\_ld.v

\* Project: Lab\_Assignment\_4

\* Designer: Chanartip Soonthornwan, Jonathan Shihata

\* Email: Chanartip.Soonthornwan@gmail.com, JonnyShihata@gmail.com

\* Rev. No.: Version 1.0

\* Rev. Date: 10/7/2017

\*

\* Rev. No.: Version 1.1

\* Rev. Date: Current Rev. 10/10/2017

\* Notes: - filename changed

\* - comments added

\*

\* Purpose: A 32-bit memory holding 32-bit value and output it

\* on next active clock.

\*

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**module** reg32\_no\_ld**(**clk**,** reset**,** d**,** q**);**

**input** clk**,** reset**;** // on-board clock, and reset signal

**input** **[**31**:**0**]** d**;** // data inputs

**output** **reg** **[**31**:**0**]** q**;** // data outputs

**always@(posedge** clk**,** **posedge** reset**)**

**if(**reset**)** q **<=** 32'b0**;** // reset

**else** q **<=** d**;** // assign new value

**endmodule**

## Register File

`timescale 1ns **/** 1ps

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* C E C S 4 4 0 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* File Name: regfile32.v

\* Project: Lab\_Assignment\_2

\* Designer: Chanartip Soonthornwan, Jonathan Shihata

\* Email: Chanartip.Soonthornwan@gmail.com, JonnyShihata@gmail.com

\* Rev. No.: Version 1.0

\* Rev. Date: Current Rev. Date 9/14/2017

\*

\* Purpose: Represents an array of register that has 32x32 Width

\* (32 registers) and Depth(32 bits each). Being able to save

\* 32-bit input(D\_in) and any register(specified by D\_Addr) when

\* Data Write Enable(D\_En) is HIGH. Also being able to output

\* two 32-bit data through S and T paths where S and T will

\* get the data from S\_Addr and T\_Addrrespectively.

\*

\* Notes: (\*)At Write Section, if Data Write Enable(D\_En) is HIGH,

\* it will write the input data into the register at D\_Addr,

\* but it will notwrite on $r0 because Address $zero is restrict

\* for READ-ONLY.Therefore, when D\_En is HIGH and D\_Addr is

\* at $r0, the program will fall into Not changing content.

\*

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**module** regfile32**(**clk**,** reset**,** D\_En**,** D\_Addr**,** S\_Addr**,** T\_Addr**,** D\_in**,** S**,** T**);**

**input** clk**,** reset**;** // Clock and Reset Signal

**input** D\_En**;** // Data Write Enable

**input** **[** 4**:**0**]** D\_Addr**;** // Address of register to write

**input** **[** 4**:**0**]** S\_Addr**,** T\_Addr**;** // Address of register to read

**input** **[**31**:**0**]** D\_in**;** // 32-bit input data

**output** **[**31**:**0**]** S**,** T**;** // 32-bit read data

**reg** **[**31**:**0**]** reg\_array **[**31**:**0**];** // 32 of 32-bit registers

// Write Section

**always@(posedge** clk**,** **posedge** reset**)** **begin**

**if(**reset**)** // Get reset signal

reg\_array**[**0**]** **<=** 32'h0**;** // assign $r0 to zeroes

**else** **if(**D\_En **&&** D\_Addr **!=** 5'h0**)** // Load Input data

reg\_array**[**D\_Addr**]** **<=** D\_in**;** // to a register that is not $r0

**else** // Not changing content

reg\_array**[**D\_Addr**]** **<=** reg\_array**[**D\_Addr**];** // (\*)

**end**

// Read Section

// S and T read get 32-bit data from registers

// at S\_Addr and T\_addr respectively

**assign** S **=** reg\_array**[**S\_Addr**];**

**assign** T **=** reg\_array**[**T\_Addr**];**

**endmodule**

# Verification

/\* Instruction Memory Module 1 \*/

R1 <- 0x012345678

R2 <- 0x876543210

@0

3c 01 12 34

34 21 56 78

3c 02 87 65

R3 <- R1

34 42 43 21

00 01 18 20

BEQ R2, R1. Should not branch to no\_eq

10 22 00 01

10 23 00 03

BEQ R3, R1. should branch to yes\_eq.

no\_eq:

3c 0e ff ff

Set fail flag.  
R14 <- 0xFFFFFFFF  
break

35 ce ff ff

00 00 00 0d

yes\_eq:

Set pass flag  
R14 <-00000000

00 00 70 20

14 23 00 01

14 22 00 03

BNE R3, R1. Should not branch to no\_ne.  
BNE R2, R1. Should branch to yest\_ne.

no\_ne:

3c 0f ff ff

35 ef ff ff

00 00 00 0d

Set fail flag.  
R15 <- 0xFFFFFFFF  
break.

yes\_ne:

00 00 78 20

3c 0d 10 01

Set pass flag  
R15 <- 00000000

35 ad 00 c0

ad a1 00 00

00 00 00 0d

Set DM pointer  
R13 <- 0x100100c0  
DM[R13] <- R1  
break

/\* Verification 1 \*/

BREAK INSTRUCTION FETCHED 625.0 ns

Loaded R1-R3

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

t= 631.0 ns $r0 = 00000000 || t= 631.0 ns $r16 = xxxxxxxx

t= 631.0 ns $r1 = 12345678 || t= 631.0 ns $r17 = xxxxxxxx

t= 631.0 ns $r2 = 87654321 || t= 631.0 ns $r18 = xxxxxxxx

t= 631.0 ns $r3 = 12345678 || t= 631.0 ns $r19 = xxxxxxxx

DM pointer

t= 631.0 ns $r4 = xxxxxxxx || t= 631.0 ns $r20 = xxxxxxxx

t= 631.0 ns $r5 = xxxxxxxx || t= 631.0 ns $r21 = xxxxxxxx

t= 631.0 ns $r6 = xxxxxxxx || t= 631.0 ns $r22 = xxxxxxxx

t= 631.0 ns $r7 = xxxxxxxx || t= 631.0 ns $r23 = xxxxxxxx

t= 631.0 ns $r8 = xxxxxxxx || t= 631.0 ns $r24 = xxxxxxxx

Pass flags for BEQ and BNE

t= 631.0 ns $r9 = xxxxxxxx || t= 631.0 ns $r25 = xxxxxxxx

t= 631.0 ns $r10 = xxxxxxxx || t= 631.0 ns $r26 = xxxxxxxx

t= 631.0 ns $r11 = xxxxxxxx || t= 631.0 ns $r27 = xxxxxxxx

t= 631.0 ns $r12 = xxxxxxxx || t= 631.0 ns $r28 = xxxxxxxx

t= 631.0 ns $r13 = 100100c0 || t= 631.0 ns $r29 = 000003fc

t= 631.0 ns $r14 = 00000000 || t= 631.0 ns $r30 = xxxxxxxx

Stored by R3

t= 631.0 ns $r15 = 00000000 || t= 631.0 ns $r31 = xxxxxxxx

time= 631.0 ns M[3F0]=xxxxxxxx

\_\_\_\_\_\_\_Data Memory(DM)\_\_\_\_\_\_\_\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_IO Memory(IOM)\_\_\_\_\_

t= 631.0 ns DM[000000c0] = 12345678 || t= 631.0 ns IOM[000000c0] = xxxxxxxx

t= 631.0 ns DM[000000c4] = xxxxxxxx || t= 631.0 ns IOM[000000c4] = xxxxxxxx

t= 631.0 ns DM[000000c8] = xxxxxxxx || t= 631.0 ns IOM[000000c8] = xxxxxxxx

t= 631.0 ns DM[000000cc] = xxxxxxxx || t= 631.0 ns IOM[000000cc] = xxxxxxxx

t= 631.0 ns DM[000000d0] = xxxxxxxx || t= 631.0 ns IOM[000000d0] = xxxxxxxx

t= 631.0 ns DM[000000d4] = xxxxxxxx || t= 631.0 ns IOM[000000d4] = xxxxxxxx

t= 631.0 ns DM[000000d8] = xxxxxxxx || t= 631.0 ns IOM[000000d8] = xxxxxxxx

t= 631.0 ns DM[000000dc] = xxxxxxxx || t= 631.0 ns IOM[000000dc] = xxxxxxxx

t= 631.0 ns DM[000000e0] = xxxxxxxx || t= 631.0 ns IOM[000000e0] = xxxxxxxx

t= 631.0 ns DM[000000e4] = xxxxxxxx || t= 631.0 ns IOM[000000e4] = xxxxxxxx

t= 631.0 ns DM[000000e8] = xxxxxxxx || t= 631.0 ns IOM[000000e8] = xxxxxxxx

t= 631.0 ns DM[000000ec] = xxxxxxxx || t= 631.0 ns IOM[000000ec] = xxxxxxxx

t= 631.0 ns DM[000000f0] = xxxxxxxx || t= 631.0 ns IOM[000000f0] = xxxxxxxx

t= 631.0 ns DM[000000f4] = xxxxxxxx || t= 631.0 ns IOM[000000f4] = xxxxxxxx

t= 631.0 ns DM[000000f8] = xxxxxxxx || t= 631.0 ns IOM[000000f8] = xxxxxxxx

t= 631.0 ns DM[000000fc] = xxxxxxxx || t= 631.0 ns IOM[000000fc] = xxxxxxxx

Stopped at time : 631 ns

/\* Instruction Memory Module 2 \*/

R1 <- 0xFFFFFFFF  
R2 <- 0x10  
R15 <- 100100C0

@0

3c 01 ff ff

34 21 ff ff

20 02 00 10

3c 0f 10 01

35 ef 00 c0

1. Logical Shift Right R1 1 bit.
2. Store in DM[R15]
3. Increment R15 by 4
4. Decrement loop counter R2
5. If R2 is zero, jump to exit. Else, branch to top
6. Break.

Top:

00 01 08 42

ad e1 00 00

21 ef 00 04

20 42 ff ff

14 40 ff fb

08 10 00 0c

00 00 00 0d

Exit:

3c 0e 5a 5a

Set pass flag. R14 <- 5A5A3C3C

Then break.

35 ce 3c 3c

00 00 00 0d

/\* Verification 2 \*/

BREAK INSTRUCTION FETCHED 3575.0 ns

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

Result after shifting by one 16 times

t=3581.0 ns $r0 = 00000000 || t=3581.0 ns $r16 = xxxxxxxx

t=3581.0 ns $r1 = 0000ffff || t=3581.0 ns $r17 = xxxxxxxx

t=3581.0 ns $r2 = 00000000 || t=3581.0 ns $r18 = xxxxxxxx

Loop counter ended

t=3581.0 ns $r3 = xxxxxxxx || t=3581.0 ns $r19 = xxxxxxxx

t=3581.0 ns $r4 = xxxxxxxx || t=3581.0 ns $r20 = xxxxxxxx

t=3581.0 ns $r5 = xxxxxxxx || t=3581.0 ns $r21 = xxxxxxxx

t=3581.0 ns $r6 = xxxxxxxx || t=3581.0 ns $r22 = xxxxxxxx

t=3581.0 ns $r7 = xxxxxxxx || t=3581.0 ns $r23 = xxxxxxxx

t=3581.0 ns $r8 = xxxxxxxx || t=3581.0 ns $r24 = xxxxxxxx

t=3581.0 ns $r9 = xxxxxxxx || t=3581.0 ns $r25 = xxxxxxxx

t=3581.0 ns $r10 = xxxxxxxx || t=3581.0 ns $r26 = xxxxxxxx

t=3581.0 ns $r11 = xxxxxxxx || t=3581.0 ns $r27 = xxxxxxxx

Passed flag.

t=3581.0 ns $r12 = xxxxxxxx || t=3581.0 ns $r28 = xxxxxxxx

t=3581.0 ns $r13 = xxxxxxxx || t=3581.0 ns $r29 = 000003fc

DM pointer after looping

t=3581.0 ns $r14 = 5a5a3c3c || t=3581.0 ns $r30 = xxxxxxxx

t=3581.0 ns $r15 = 10010100 || t=3581.0 ns $r31 = xxxxxxxx

time=3581.0 ns M[3F0]=xxxxxxxx

\_\_\_\_\_\_\_Data Memory(DM)\_\_\_\_\_\_\_\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_IO Memory(IOM)\_\_\_\_\_

Result from logical shifting 0xFFFFFFFF

t=3581.0 ns DM[000000c0] = 7fffffff || t=3581.0 ns IOM[000000c0] = xxxxxxxx

t=3581.0 ns DM[000000c4] = 3fffffff || t=3581.0 ns IOM[000000c4] = xxxxxxxx

t=3581.0 ns DM[000000c8] = 1fffffff || t=3581.0 ns IOM[000000c8] = xxxxxxxx

t=3581.0 ns DM[000000cc] = 0fffffff || t=3581.0 ns IOM[000000cc] = xxxxxxxx

t=3581.0 ns DM[000000d0] = 07ffffff || t=3581.0 ns IOM[000000d0] = xxxxxxxx

t=3581.0 ns DM[000000d4] = 03ffffff || t=3581.0 ns IOM[000000d4] = xxxxxxxx

t=3581.0 ns DM[000000d8] = 01ffffff || t=3581.0 ns IOM[000000d8] = xxxxxxxx

t=3581.0 ns DM[000000dc] = 00ffffff || t=3581.0 ns IOM[000000dc] = xxxxxxxx

t=3581.0 ns DM[000000e0] = 007fffff || t=3581.0 ns IOM[000000e0] = xxxxxxxx

t=3581.0 ns DM[000000e4] = 003fffff || t=3581.0 ns IOM[000000e4] = xxxxxxxx

t=3581.0 ns DM[000000e8] = 001fffff || t=3581.0 ns IOM[000000e8] = xxxxxxxx

t=3581.0 ns DM[000000ec] = 000fffff || t=3581.0 ns IOM[000000ec] = xxxxxxxx

t=3581.0 ns DM[000000f0] = 0007ffff || t=3581.0 ns IOM[000000f0] = xxxxxxxx

t=3581.0 ns DM[000000f4] = 0003ffff || t=3581.0 ns IOM[000000f4] = xxxxxxxx

t=3581.0 ns DM[000000f8] = 0001ffff || t=3581.0 ns IOM[000000f8] = xxxxxxxx

t=3581.0 ns DM[000000fc] = 0000ffff || t=3581.0 ns IOM[000000fc] = xxxxxxxx

Stopped at time : 3581 ns

/\* Instruction Memory Module 3 \*/

R1 <- 0x8000FFFF  
R2 <- 0x10  
R15 <- 100100C0

@0

3c 01 80 00

34 21 ff ff

20 02 00 10

3c 0f 10 01

1. Arithmetic Shift Right R1 1 bit.
2. Store in DM[R15]
3. Increment R15 by 4
4. Decrement loop counter R2
5. If R2 is zero, jump to exit. Else, branch to top
6. Break.

Top:

35 ef 00 c0

00 01 08 43

ad e1 00 00

21 ef 00 04

20 42 ff ff

14 40 ff fb

08 10 00 0c

00 00 00 0d

Exit:

3c 0e 5a 5a

Set pass flag. R14 <- 5A5A3C3C

Then break.

35 ce 3c 3c

00 00 00 0d

/\* Verification 3 \*/

BREAK INSTRUCTION FETCHED 3575.0 ns

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

Result after shifting by one 16 times

t=3581.0 ns $r0 = 00000000 || t=3581.0 ns $r16 = xxxxxxxx

t=3581.0 ns $r1 = ffff8000 || t=3581.0 ns $r17 = xxxxxxxx

t=3581.0 ns $r2 = 00000000 || t=3581.0 ns $r18 = xxxxxxxx

t=3581.0 ns $r3 = xxxxxxxx || t=3581.0 ns $r19 = xxxxxxxx

Loop counter ended

t=3581.0 ns $r4 = xxxxxxxx || t=3581.0 ns $r20 = xxxxxxxx

t=3581.0 ns $r5 = xxxxxxxx || t=3581.0 ns $r21 = xxxxxxxx

t=3581.0 ns $r6 = xxxxxxxx || t=3581.0 ns $r22 = xxxxxxxx

t=3581.0 ns $r7 = xxxxxxxx || t=3581.0 ns $r23 = xxxxxxxx

t=3581.0 ns $r8 = xxxxxxxx || t=3581.0 ns $r24 = xxxxxxxx

t=3581.0 ns $r9 = xxxxxxxx || t=3581.0 ns $r25 = xxxxxxxx

t=3581.0 ns $r10 = xxxxxxxx || t=3581.0 ns $r26 = xxxxxxxx

t=3581.0 ns $r11 = xxxxxxxx || t=3581.0 ns $r27 = xxxxxxxx

Passed flag.

t=3581.0 ns $r12 = xxxxxxxx || t=3581.0 ns $r28 = xxxxxxxx

t=3581.0 ns $r13 = xxxxxxxx || t=3581.0 ns $r29 = 000003fc

DM pointer after looping

t=3581.0 ns $r14 = 5a5a3c3c || t=3581.0 ns $r30 = xxxxxxxx

t=3581.0 ns $r15 = 10010100 || t=3581.0 ns $r31 = xxxxxxxx

time=3581.0 ns M[3F0]=xxxxxxxx

\_\_\_\_\_\_\_Data Memory(DM)\_\_\_\_\_\_\_\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_IO Memory(IOM)\_\_\_\_\_

Result from arithemetic shifting 0xFFFFFFFF

t=3581.0 ns DM[000000c0] = c0007fff || t=3581.0 ns IOM[000000c0] = xxxxxxxx

t=3581.0 ns DM[000000c4] = e0003fff || t=3581.0 ns IOM[000000c4] = xxxxxxxx

t=3581.0 ns DM[000000c8] = f0001fff || t=3581.0 ns IOM[000000c8] = xxxxxxxx

t=3581.0 ns DM[000000cc] = f8000fff || t=3581.0 ns IOM[000000cc] = xxxxxxxx

t=3581.0 ns DM[000000d0] = fc0007ff || t=3581.0 ns IOM[000000d0] = xxxxxxxx

t=3581.0 ns DM[000000d4] = fe0003ff || t=3581.0 ns IOM[000000d4] = xxxxxxxx

t=3581.0 ns DM[000000d8] = ff0001ff || t=3581.0 ns IOM[000000d8] = xxxxxxxx

t=3581.0 ns DM[000000dc] = ff8000ff || t=3581.0 ns IOM[000000dc] = xxxxxxxx

t=3581.0 ns DM[000000e0] = ffc0007f || t=3581.0 ns IOM[000000e0] = xxxxxxxx

t=3581.0 ns DM[000000e4] = ffe0003f || t=3581.0 ns IOM[000000e4] = xxxxxxxx

t=3581.0 ns DM[000000e8] = fff0001f || t=3581.0 ns IOM[000000e8] = xxxxxxxx

t=3581.0 ns DM[000000ec] = fff8000f || t=3581.0 ns IOM[000000ec] = xxxxxxxx

t=3581.0 ns DM[000000f0] = fffc0007 || t=3581.0 ns IOM[000000f0] = xxxxxxxx

t=3581.0 ns DM[000000f4] = fffe0003 || t=3581.0 ns IOM[000000f4] = xxxxxxxx

t=3581.0 ns DM[000000f8] = ffff0001 || t=3581.0 ns IOM[000000f8] = xxxxxxxx

t=3581.0 ns DM[000000fc] = ffff8000 || t=3581.0 ns IOM[000000fc] = xxxxxxxx

Stopped at time : 3581 ns :

/\* Instruction Memory Module 4 \*/

@0

3c 01 ff ff

R1 <- 0xFFFFFFFF  
R2 <- 0x10  
R15 <- 100100C0

34 21 ff ff

20 02 00 10

3c 0f 10 01

35 ef 00 c0

* Logical Shift Left R1 1 bit.
* Store in DM[R15]
* Increment R15 by 4
* Decrement loop counter R2
* Set R3 = 1 if R0 < R2 (R3=0 when R2<R0)
* Branch to Top if R3 = 1
* Break.

Top:

00 01 08 40

ad e1 00 00

21 ef 00 04

20 42 ff ff

00 02 18 2a

14 60 ff fa

08 10 00 0d

00 00 00 0d

Exit:

Jump to Exit

3c 0e 5a 5a

35 ce 3c 3c

00 00 00 0d

Break. Should not be fetched.

Set pass flag. R14 <- 5A5A3C3C

Then break.

/\* Verification 4 \*/

Result after shifting by one 16 times

BREAK INSTRUCTION FETCHED 4215.0 ns

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

t=4221.0 ns $r0 = 00000000 || t=4221.0 ns $r16 = xxxxxxxx

Loop counter ended

t=4221.0 ns $r1 = ffff0000 || t=4221.0 ns $r17 = xxxxxxxx

t=4221.0 ns $r2 = 00000000 || t=4221.0 ns $r18 = xxxxxxxx

R3 = 0 when R2 < R0

t=4221.0 ns $r3 = 00000000 || t=4221.0 ns $r19 = xxxxxxxx

t=4221.0 ns $r4 = xxxxxxxx || t=4221.0 ns $r20 = xxxxxxxx

t=4221.0 ns $r5 = xxxxxxxx || t=4221.0 ns $r21 = xxxxxxxx

t=4221.0 ns $r6 = xxxxxxxx || t=4221.0 ns $r22 = xxxxxxxx

t=4221.0 ns $r7 = xxxxxxxx || t=4221.0 ns $r23 = xxxxxxxx

t=4221.0 ns $r8 = xxxxxxxx || t=4221.0 ns $r24 = xxxxxxxx

t=4221.0 ns $r9 = xxxxxxxx || t=4221.0 ns $r25 = xxxxxxxx

t=4221.0 ns $r10 = xxxxxxxx || t=4221.0 ns $r26 = xxxxxxxx

t=4221.0 ns $r11 = xxxxxxxx || t=4221.0 ns $r27 = xxxxxxxx

Passed flag.

t=4221.0 ns $r12 = xxxxxxxx || t=4221.0 ns $r28 = xxxxxxxx

t=4221.0 ns $r13 = xxxxxxxx || t=4221.0 ns $r29 = 000003fc

DM pointer after looping

t=4221.0 ns $r14 = 5a5a3c3c || t=4221.0 ns $r30 = xxxxxxxx

t=4221.0 ns $r15 = 10010100 || t=4221.0 ns $r31 = xxxxxxxx

time=4221.0 ns M[3F0]=xxxxxxxx

\_\_\_\_\_\_\_Data Memory(DM)\_\_\_\_\_\_\_\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_IO Memory(IOM)\_\_\_\_\_

Result from logical shifting 0xFFFFFFFF

t=4221.0 ns DM[000000c0] = fffffffe || t=4221.0 ns IOM[000000c0] = xxxxxxxx

t=4221.0 ns DM[000000c4] = fffffffc || t=4221.0 ns IOM[000000c4] = xxxxxxxx

t=4221.0 ns DM[000000c8] = fffffff8 || t=4221.0 ns IOM[000000c8] = xxxxxxxx

t=4221.0 ns DM[000000cc] = fffffff0 || t=4221.0 ns IOM[000000cc] = xxxxxxxx

t=4221.0 ns DM[000000d0] = ffffffe0 || t=4221.0 ns IOM[000000d0] = xxxxxxxx

t=4221.0 ns DM[000000d4] = ffffffc0 || t=4221.0 ns IOM[000000d4] = xxxxxxxx

t=4221.0 ns DM[000000d8] = ffffff80 || t=4221.0 ns IOM[000000d8] = xxxxxxxx

t=4221.0 ns DM[000000dc] = ffffff00 || t=4221.0 ns IOM[000000dc] = xxxxxxxx

t=4221.0 ns DM[000000e0] = fffffe00 || t=4221.0 ns IOM[000000e0] = xxxxxxxx

t=4221.0 ns DM[000000e4] = fffffc00 || t=4221.0 ns IOM[000000e4] = xxxxxxxx

t=4221.0 ns DM[000000e8] = fffff800 || t=4221.0 ns IOM[000000e8] = xxxxxxxx

t=4221.0 ns DM[000000ec] = fffff000 || t=4221.0 ns IOM[000000ec] = xxxxxxxx

t=4221.0 ns DM[000000f0] = ffffe000 || t=4221.0 ns IOM[000000f0] = xxxxxxxx

t=4221.0 ns DM[000000f4] = ffffc000 || t=4221.0 ns IOM[000000f4] = xxxxxxxx

t=4221.0 ns DM[000000f8] = ffff8000 || t=4221.0 ns IOM[000000f8] = xxxxxxxx

t=4221.0 ns DM[000000fc] = ffff0000 || t=4221.0 ns IOM[000000fc] = xxxxxxxx

Stopped at time : 4221 ns :

/\* Instruction Memory Module 5 \*/

@0

R1 <- 0xFFFFFFFF

R2 <- -16

R15 <- 100100C0

3c 01 ff ff

34 21 ff ff

20 02 ff f0

3c 0f 10 01

35 ef 00 c0

1. Logical Shift Left R1 1 bit.
2. Store in DM[R15]
3. Increment R15 by 4
4. Increment loop counter R2
5. Set R3 = 1 if R2 < zero (R3=0 when R2 > zero)
6. Branch to Top if R3 = 1
7. Break.

Top:

00 01 08 40

ad e1 00 00

21 ef 00 04

20 42 00 01

28 43 00 00

14 60 ff fa

08 10 00 0d

00 00 00 0d

Exit:

3c 0e 5a 5a

35 ce 3c 3c

Jump to Exit

00 00 00 0d

Set pass flag. R14 <- 5A5A3C3C

Then break.

Break. Should not be fetched.

/\* Verification 5 \*/

BREAK INSTRUCTION FETCHED 4215.0 ns

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

Result after shifting by one 16 times

t=4221.0 ns $r0 = 00000000 || t=4221.0 ns $r16 = xxxxxxxx

t=4221.0 ns $r1 = ffff0000 || t=4221.0 ns $r17 = xxxxxxxx

Loop counter ended

t=4221.0 ns $r2 = 00000000 || t=4221.0 ns $r18 = xxxxxxxx

t=4221.0 ns $r3 = 00000000 || t=4221.0 ns $r19 = xxxxxxxx

t=4221.0 ns $r4 = xxxxxxxx || t=4221.0 ns $r20 = xxxxxxxx

R3 = 0 when R2 > zero

t=4221.0 ns $r5 = xxxxxxxx || t=4221.0 ns $r21 = xxxxxxxx

Result from logical shifting 0xFFFFFFFF

t=4221.0 ns $r6 = xxxxxxxx || t=4221.0 ns $r22 = xxxxxxxx

t=4221.0 ns $r7 = xxxxxxxx || t=4221.0 ns $r23 = xxxxxxxx

t=4221.0 ns $r8 = xxxxxxxx || t=4221.0 ns $r24 = xxxxxxxx

t=4221.0 ns $r9 = xxxxxxxx || t=4221.0 ns $r25 = xxxxxxxx

t=4221.0 ns $r10 = xxxxxxxx || t=4221.0 ns $r26 = xxxxxxxx

Passed flag.

t=4221.0 ns $r11 = xxxxxxxx || t=4221.0 ns $r27 = xxxxxxxx

t=4221.0 ns $r12 = xxxxxxxx || t=4221.0 ns $r28 = xxxxxxxx

t=4221.0 ns $r13 = xxxxxxxx || t=4221.0 ns $r29 = 000003fc

t=4221.0 ns $r14 = 5a5a3c3c || t=4221.0 ns $r30 = xxxxxxxx

t=4221.0 ns $r15 = 10010100 || t=4221.0 ns $r31 = xxxxxxxx

DM pointer after looping

time=4221.0 ns M[3F0]=xxxxxxxx

\_\_\_\_\_\_\_Data Memory(DM)\_\_\_\_\_\_\_\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_IO Memory(IOM)\_\_\_\_\_

t=4221.0 ns DM[000000c0] = fffffffe || t=4221.0 ns IOM[000000c0] = xxxxxxxx

t=4221.0 ns DM[000000c4] = fffffffc || t=4221.0 ns IOM[000000c4] = xxxxxxxx

t=4221.0 ns DM[000000c8] = fffffff8 || t=4221.0 ns IOM[000000c8] = xxxxxxxx

t=4221.0 ns DM[000000cc] = fffffff0 || t=4221.0 ns IOM[000000cc] = xxxxxxxx

t=4221.0 ns DM[000000d0] = ffffffe0 || t=4221.0 ns IOM[000000d0] = xxxxxxxx

t=4221.0 ns DM[000000d4] = ffffffc0 || t=4221.0 ns IOM[000000d4] = xxxxxxxx

t=4221.0 ns DM[000000d8] = ffffff80 || t=4221.0 ns IOM[000000d8] = xxxxxxxx

t=4221.0 ns DM[000000dc] = ffffff00 || t=4221.0 ns IOM[000000dc] = xxxxxxxx

t=4221.0 ns DM[000000e0] = fffffe00 || t=4221.0 ns IOM[000000e0] = xxxxxxxx

t=4221.0 ns DM[000000e4] = fffffc00 || t=4221.0 ns IOM[000000e4] = xxxxxxxx

t=4221.0 ns DM[000000e8] = fffff800 || t=4221.0 ns IOM[000000e8] = xxxxxxxx

t=4221.0 ns DM[000000ec] = fffff000 || t=4221.0 ns IOM[000000ec] = xxxxxxxx

t=4221.0 ns DM[000000f0] = ffffe000 || t=4221.0 ns IOM[000000f0] = xxxxxxxx

t=4221.0 ns DM[000000f4] = ffffc000 || t=4221.0 ns IOM[000000f4] = xxxxxxxx

t=4221.0 ns DM[000000f8] = ffff8000 || t=4221.0 ns IOM[000000f8] = xxxxxxxx

t=4221.0 ns DM[000000fc] = ffff0000 || t=4221.0 ns IOM[000000fc] = xxxxxxxx

Stopped at time : 4221 ns

/\* Instruction Memory Module 6 \*/

@0

R15 <- 0x10010000

R14 <- 0x100100C0

R13 <- 16

3c 0f 10 01

35 ef 00 00

3c 0e 10 01

35 ce 00 c0

20 0d 00 10

Load R01 to R12 from Data Memory based on where R15 points at and increment by 4 on each Load instruction

8d e1 00 04

8d e2 00 08

8d e3 00 0c

8d e4 00 10

8d e5 00 14

8d e6 00 18

8d e7 00 1c

8d e8 00 20

1. Use R17 as a temporary register loading data from memory to memory; R17 <- DM[R15]
2. Store DM[R14] <- R17
3. Increment both pointer R14 and R15 by 4
4. Decrement loop counter R13 by 1
5. Branch to mem2mem until R13=0
6. Break

8d e9 00 24

8d ea 00 28

8d eb 00 2c

8d ec 00 30

mem2mem:

8d f1 00 00

ad d1 00 00

21 ef 00 04

21 ce 00 04

**Data memory**

@00            // Big Endian Format

C3 C3 C3 C3    // 0x00:03

12 34 56 78    // 0x04:07

89 AB CD EF    // 0x08:0B

A5 A5 A5 A5    // 0x0C:0F

5A 5A 5A 5A    // 0x10:13  //word 4

24 68 AC E0    // 0x14:17

13 57 9B DF    // 0x18:1B

0F 0F 0F 0F    // 0x1C:1F

F0 F0 F0 F0    // 0x20:23  //word 8

00 00 00 09    // 0x24:27

00 00 00 0A    // 0x28:2B

00 00 00 0B    // 0x2C:2F

00 00 00 0C    // 0x30:33  //word 12

00 00 00 0D    // 0x34:37

FF FF FF F8    // 0x38:3B

00 00 75 CC    // 0x3C:3F

@1CC

AB CD EF 01    // 0x1CC:1CF

@3F8

00 00 00 00    // 0x3F8:3FB

21 ad ff ff

15 a0 ff fa

00 00 00 0d

/\* Verification 6 \*/

BREAK INSTRUCTION FETCHED 4865.0 ns

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

Temporary register

t=4871.0 ns $r0 = 00000000 || t=4871.0 ns $r16 = xxxxxxxx

t=4871.0 ns $r1 = 12345678 || t=4871.0 ns $r17 = 000075cc

t=4871.0 ns $r2 = 89abcdef || t=4871.0 ns $r18 = xxxxxxxx

t=4871.0 ns $r3 = a5a5a5a5 || t=4871.0 ns $r19 = xxxxxxxx

t=4871.0 ns $r4 = 5a5a5a5a || t=4871.0 ns $r20 = xxxxxxxx

t=4871.0 ns $r5 = 2468ace0 || t=4871.0 ns $r21 = xxxxxxxx

Data memory loaded to R1 through R12

t=4871.0 ns $r6 = 13579bdf || t=4871.0 ns $r22 = xxxxxxxx

t=4871.0 ns $r7 = 0f0f0f0f || t=4871.0 ns $r23 = xxxxxxxx

t=4871.0 ns $r8 = f0f0f0f0 || t=4871.0 ns $r24 = xxxxxxxx

Loop counter

t=4871.0 ns $r9 = 00000009 || t=4871.0 ns $r25 = xxxxxxxx

t=4871.0 ns $r10 = 0000000a || t=4871.0 ns $r26 = xxxxxxxx

t=4871.0 ns $r11 = 0000000b || t=4871.0 ns $r27 = xxxxxxxx

t=4871.0 ns $r12 = 0000000c || t=4871.0 ns $r28 = xxxxxxxx

Data memory pointers

t=4871.0 ns $r13 = 00000000 || t=4871.0 ns $r29 = 000003fc

t=4871.0 ns $r14 = 10010100 || t=4871.0 ns $r30 = xxxxxxxx

t=4871.0 ns $r15 = 10010040 || t=4871.0 ns $r31 = xxxxxxxx

time=4871.0 ns M[3F0]=xxxxxxxx

\_\_\_\_\_\_\_Data Memory(DM)\_\_\_\_\_\_\_\_ \_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_IO Memory(IOM)\_\_\_\_\_

t=4871.0 ns DM[000000c0] = c3c3c3c3 || t=4871.0 ns IOM[000000c0] = xxxxxxxx

t=4871.0 ns DM[000000c4] = 12345678 || t=4871. ns IOM[000000c4] = xxxxxxxx

Data memory from [000] stored to [0C0] through mem2mem loop.

t=4871.0 ns DM[000000c8] = 89abcdef || t=4871.0 ns IOM[000000c8] = xxxxxxxx

t=4871.0 ns DM[000000cc] = a5a5a5a5 || t=4871.0 ns IOM[000000cc] = xxxxxxxx

t=4871.0 ns DM[000000d0] = 5a5a5a5a || t=4871.0 ns IOM[000000d0] = xxxxxxxx

t=4871.0 ns DM[000000d4] = 2468ace0 || t=4871.0 ns IOM[000000d4] = xxxxxxxx

t=4871.0 ns DM[000000d8] = 13579bdf || t=4871.0 ns IOM[000000d8] = xxxxxxxx

t=4871.0 ns DM[000000dc] = 0f0f0f0f || t=4871.0 ns IOM[000000dc] = xxxxxxxx

t=4871.0 ns DM[000000e0] = f0f0f0f0 || t=4871.0 ns IOM[000000e0] = xxxxxxxx

t=4871.0 ns DM[000000e4] = 00000009 || t=4871.0 ns IOM[000000e4] = xxxxxxxx

t=4871.0 ns DM[000000e8] = 0000000a || t=4871.0 ns IOM[000000e8] = xxxxxxxx

t=4871.0 ns DM[000000ec] = 0000000b || t=4871.0 ns IOM[000000ec] = xxxxxxxx

t=4871.0 ns DM[000000f0] = 0000000c || t=4871.0 ns IOM[000000f0] = xxxxxxxx

t=4871.0 ns DM[000000f4] = 0000000d || t=4871.0 ns IOM[000000f4] = xxxxxxxx

t=4871.0 ns DM[000000f8] = fffffff8 || t=4871.0 ns IOM[000000f8] = xxxxxxxx

t=4871.0 ns DM[000000fc] = 000075cc || t=4871.0 ns IOM[000000fc] = xxxxxxxx

Stopped at time : 4871 ns

/\* Instruction Memory Module 7 \*/

**Data memory**

@00            // Big Endian Format

C3 C3 C3 C3    // 0x00:03

12 34 56 78    // 0x04:07

89 AB CD EF    // 0x08:0B

A5 A5 A5 A5    // 0x0C:0F

5A 5A 5A 5A    // 0x10:13  //word 4

24 68 AC E0    // 0x14:17

13 57 9B DF    // 0x18:1B

0F 0F 0F 0F    // 0x1C:1F

F0 F0 F0 F0    // 0x20:23  //word 8

00 00 00 09    // 0x24:27

00 00 00 0A    // 0x28:2B

00 00 00 0B    // 0x2C:2F

00 00 00 0C    // 0x30:33  //word 12

00 00 00 0D    // 0x34:37

FF FF FF F8    // 0x38:3B

00 00 75 CC    // 0x3C:3F

@1CC

AB CD EF 01    // 0x1CC:1CF

@3F8

00 00 00 00    // 0x3F8:3FB

@0

R15 <- 0x10010000

R14 <- 0x100100C0

R13 <- 16

3c 0f 10 01

35 ef 00 00

3c 0e 10 01

35 ce 00 c0

20 0d 00 10

8d e1 00 04

Load R01 to R12 from Data Memory based on where R15 points at and increment by 4 on each Load instruction

8d e2 00 08

8d e3 00 0c

8d e4 00 10

8d e5 00 14

8d e6 00 18

8d e7 00 1c

8d e8 00 20

Jump And Link (JAL) to mem2mem

8d e9 00 24

8d ea 00 28

8d eb 00 2c

Set passed flag at R15 <- 0xFFFFFFFF

8d ec 00 30

0c 10 00 15

Break

3c 0f ff ff

35 ef ff ff

1. Use R17 as a temporary register loading data from memory to memory; R17 <- DM[R15]
2. Store DM[R14] <- R17
3. Increment both pointer R14 and R15 by 4
4. Decrement loop counter R13 by 1
5. Branch to mem2mem until R13=0
6. Break

00 00 00 0d

mem2mem:

8d f1 00 00

ad d1 00 00

21 ef 00 04

21 ce 00 04

21 ad ff ff

Jump Register (JR) at R31

15 a0 ff fa

03 e0 00 08

Break

00 00 00 0d

/\* Verification 7 \*/

BREAK INSTRUCTION FETCHED 5015.0 ns

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

Temporary register

t=5021.0 ns $r0 = 00000000 || t=5021.0 ns $r16 = xxxxxxxx

t=5021.0 ns $r1 = 12345678 || t=5021.0 ns $r17 = 000075cc

t=5021.0 ns $r2 = 89abcdef || t=5021.0 ns $r18 = xxxxxxxx

t=5021.0 ns $r3 = a5a5a5a5 || t=5021.0 ns $r19 = xxxxxxxx

t=5021.0 ns $r4 = 5a5a5a5a || t=5021.0 ns $r20 = xxxxxxxx

t=5021.0 ns $r5 = 2468ace0 || t=5021.0 ns $r21 = xxxxxxxx

Data memory loaded to R1 through R12

t=5021.0 ns $r6 = 13579bdf || t=5021.0 ns $r22 = xxxxxxxx

t=5021.0 ns $r7 = 0f0f0f0f || t=5021.0 ns $r23 = xxxxxxxx

t=5021.0 ns $r8 = f0f0f0f0 || t=5021.0 ns $r24 = xxxxxxxx

t=5021.0 ns $r9 = 00000009 || t=5021.0 ns $r25 = xxxxxxxx

t=5021.0 ns $r10 = 0000000a || t=5021.0 ns $r26 = xxxxxxxx

Loop counter

t=5021.0 ns $r11 = 0000000b || t=5021.0 ns $r27 = xxxxxxxx

t=5021.0 ns $r12 = 0000000c || t=5021. 0 ns $r28 = xxxxxxxx

Data memory pointers

t=5021.0 ns $r13 = 00000000 || t=5021.0 ns $r29 = 000003fc

Data memory from [000] stored to [0C0] through mem2mem loop.

t=5021.0 ns $r14 = 10010100 || t=5021.0 ns $r30 = xxxxxxxx

t=5021.0 ns $r15 = ffffffff || t=5021.0 ns $r31 = 00000048

time=5021.0 ns M[3F0]=xxxxxxxx

Passed flag R15

\_\_\_\_\_\_\_Data Memory(DM)\_\_\_\_\_\_\_\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_IO Memory(IOM)\_\_\_\_\_

t=5021.0 ns DM[000000c0] = c3c3c3c3 || t=5021.0 ns IOM[000000c0] = xxxxxxxx

t=5021.0 ns DM[000000c4] = 12345678 || t=5021.0 ns IOM[000000c4] = xxxxxxxx

t=5021.0 ns DM[000000c8] = 89abcdef || t=5021.0 ns IOM[000000c8] = xxxxxxxx

t=5021.0 ns DM[000000cc] = a5a5a5a5 || t=5021.0 ns IOM[000000cc] = xxxxxxxx

t=5021.0 ns DM[000000d0] = 5a5a5a5a || t=5021.0 ns IOM[000000d0] = xxxxxxxx

t=5021.0 ns DM[000000d4] = 2468ace0 || t=5021.0 ns IOM[000000d4] = xxxxxxxx

t=5021.0 ns DM[000000d8] = 13579bdf || t=5021.0 ns IOM[000000d8] = xxxxxxxx

t=5021.0 ns DM[000000dc] = 0f0f0f0f || t=5021.0 ns IOM[000000dc] = xxxxxxxx

t=5021.0 ns DM[000000e0] = f0f0f0f0 || t=5021.0 ns IOM[000000e0] = xxxxxxxx

t=5021.0 ns DM[000000e4] = 00000009 || t=5021.0 ns IOM[000000e4] = xxxxxxxx

t=5021.0 ns DM[000000e8] = 0000000a || t=5021.0 ns IOM[000000e8] = xxxxxxxx

t=5021.0 ns DM[000000ec] = 0000000b || t=5021.0 ns IOM[000000ec] = xxxxxxxx

t=5021.0 ns DM[000000f0] = 0000000c || t=5021.0 ns IOM[000000f0] = xxxxxxxx

t=5021.0 ns DM[000000f4] = 0000000d || t=5021.0 ns IOM[000000f4] = xxxxxxxx

t=5021.0 ns DM[000000f8] = fffffff8 || t=5021.0 ns IOM[000000f8] = xxxxxxxx

t=5021.0 ns DM[000000fc] = 000075cc || t=5021.0 ns IOM[000000fc] = xxxxxxxx

Stopped at time : 5021 ns :

/\* Instruction Memory Module 8 \*/

Set Memory Pointer R15 <- 0x10010000

@0

3c 0f 10 01

35 ef 00 00

Load R1 through R7 from Data Memory

8d e1 00 00

8d e2 00 04

8d e3 00 08

1. Multiply R01, R02
2. Put [31:0] result into R08
3. Branch to Fail1 if R05 not equal to R08. Should not branch.

8d e4 00 0c

8d e5 00 10

8d e6 00 14

8d e7 00 18

00 22 00 18

00 00 40 12

14 a8 00 10

1. Multiply R03, R02
2. Put [31:0] result into R09
3. Put [63:32] result into R10
4. Branch to Fail2L if R06 not equal to R09. Should not branch.
5. Branch to Fail2H if R07 not equal to R10. Should not branch.

00 62 00 18

00 00 48 12

00 00 50 10

14 c9 00 0f

14 ea 00 11

00 24 00 18

00 00 58 12

1. Multiply R01, R04
2. Put [31:0] result into R11
3. Put [63:32] result into R12
4. Branch to Fail3L if R06 not equal to R11. Should not branch.
5. Branch to Fail3H if R07 not equal to R12. Should not branch.

00 00 60 10

14 cb 00 10

14 ec 00 12

00 64 00 18

00 00 68 12

14 ad 00 12

pass:

3c 0e 00 00

1. Multiply R03, R04
2. Put [31:0] result into R13
3. Branch to Fail4 if R05 not equal to R13. Should not branch.

35 ce 00 00

00 00 00 0d

fail1:

3c 0e ff ff

35 ce ff ff

00 00 00 0d

fail2L:

Set passed flag to R14 <- 0x00000000

Then break.

3c 0e ff ff

35 ce ff fe

00 00 00 0d

fail2H:

3c 0e ff ff

35 ce ff fd

Set failed flag to R14 <- 0xFFFFFFFF

Then break.

00 00 00 0d

fail3L:

3c 0e ff ff

35 ce ff fc

00 00 00 0d

fail3H:

3c 0e ff ff

35 ce ff fb

00 00 00 0d

fail4:

3c 0e ff ff

35 ce ff fa

00 00 00 0d

/\* Verification 8 \*/

BREAK INSTRUCTION FETCHED 1115.0 ns

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

t=1121.0 ns $r0 = 00000000 || t=1121.0 ns $r16 = xxxxxxxx

Load R1 through R7 from Data Memory

t=1121.0 ns $r1 = 00000019 || t=1121.0 ns $r17 = xxxxxxxx

t=1121.0 ns $r2 = 000003e8 || t=1121.0 ns $r18 = xxxxxxxx

t=1121.0 ns $r3 = ffffffe7 || t=1121.0 ns $r19 = xxxxxxxx

t=1121.0 ns $r4 = fffffc18 || t=1121.0 ns $r20 = xxxxxxxx

t=1121.0 ns $r5 = 000061a8 || t=1121.0 ns $r21 = xxxxxxxx

t=1121.0 ns $r6 = ffff9e58 || t=1121.0 ns $r22 = xxxxxxxx

R8 to R13, results from multiplications

t=1121.0 ns $r7 = ffffffff || t=1121.0 ns $r23 = xxxxxxxx

t=1121.0 ns $r8 = 000061a8 || t=1121.0 ns $r24 = xxxxxxxx

t=1121.0 ns $r9 = ffff9e58 || t=1121.0 ns $r25 = xxxxxxxx

t=1121.0 ns $r10 = ffffffff || t=1121.0 ns $r26 = xxxxxxxx

t=1121.0 ns $r11 = ffff9e58 || t=1121.0 ns $r27 = xxxxxxxx

Passed flag

t=1121.0 ns $r12 = ffffffff || t=1121.0 ns $r28 = xxxxxxxx

t=1121.0 ns $r13 = 000061a8 || t=1121.0 ns $r29 = 000003fc

t=1121.0 ns $r14 = 00000000 || t=1121.0 ns $r30 = xxxxxxxx

Data Memory pointer

t=1121.0 ns $r15 = 10010000 || t=1121.0 ns $r31 = xxxxxxxx

time=1121.0 ns M[3F0]=xxxxxxxx

\_\_\_\_\_\_\_Data Memory(DM)\_\_\_\_\_\_\_\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_IO Memory(IOM)\_\_\_\_\_

t=1121.0 ns DM[000000c0] = xxxxxxxx || t=1121.0 ns IOM[000000c0] = xxxxxxxx

t=1121.0 ns DM[000000c4] = xxxxxxxx || t=1121.0 ns IOM[000000c4] = xxxxxxxx

t=1121.0 ns DM[000000c8] = xxxxxxxx || t=1121.0 ns IOM[000000c8] = xxxxxxxx

t=1121.0 ns DM[000000cc] = xxxxxxxx || t=1121.0 ns IOM[000000cc] = xxxxxxxx

t=1121.0 ns DM[000000d0] = xxxxxxxx || t=1121.0 ns IOM[000000d0] = xxxxxxxx

t=1121.0 ns DM[000000d4] = xxxxxxxx || t=1121.0 ns IOM[000000d4] = xxxxxxxx

t=1121.0 ns DM[000000d8] = xxxxxxxx || t=1121.0 ns IOM[000000d8] = xxxxxxxx

t=1121.0 ns DM[000000dc] = xxxxxxxx || t=1121.0 ns IOM[000000dc] = xxxxxxxx

t=1121.0 ns DM[000000e0] = xxxxxxxx || t=1121.0 ns IOM[000000e0] = xxxxxxxx

t=1121.0 ns DM[000000e4] = xxxxxxxx || t=1121.0 ns IOM[000000e4] = xxxxxxxx

t=1121.0 ns DM[000000e8] = xxxxxxxx || t=1121.0 ns IOM[000000e8] = xxxxxxxx

t=1121.0 ns DM[000000ec] = xxxxxxxx || t=1121.0 ns IOM[000000ec] = xxxxxxxx

t=1121.0 ns DM[000000f0] = xxxxxxxx || t=1121.0 ns IOM[000000f0] = xxxxxxxx

t=1121.0 ns DM[000000f4] = xxxxxxxx || t=1121.0 ns IOM[000000f4] = xxxxxxxx

t=1121.0 ns DM[000000f8] = xxxxxxxx || t=1121.0 ns IOM[000000f8] = xxxxxxxx

t=1121.0 ns DM[000000fc] = xxxxxxxx || t=1121.0 ns IOM[000000fc] = xxxxxxxx

Stopped at time : 1121 ns

/\* Instruction Memory Module 9 \*/

Set Memory Pointer R15 <- 0x10010000

Set failed flag to R14 <- 0xFFFFFFFF

Then break.

Set passed flag to R14 <- 0x00000000

Then break.

@0

3c 0f 10 01

35 ef 00 c0

R1 <- 0xFFFFFF8A; R2 <- 0x0000008A

20 01 ff 8a

20 02 00 8a

0c 10 00 22

Jump and Link to slt\_tests

3c 0d 77 88

35 ad 77 88

R13 <- 0x77887788;   
R12 <- 0x88778877;  
R11 <- 0xFFFFFFFF;

3c 0c 88 77

35 8c 88 77

3c 0b ff ff

35 6b ff ff

01 ac 50 26

1. XOR R10, R13, R12; R10 <- FFFFFFFF
2. Branch to xor\_pass if R10 = R11
3. If not branch, R14 <- 0xFFFFFFFB and break.

11 4b 00 02

20 0e ff fb

00 00 00 0d

xor\_pass:

1. R9 <- 0x00000000
2. Branch to and\_pass if R9 = zero
3. If not branch, R14 <- 0xFFFFFFFA and break.

01 ac 48 24

11 20 00 02

20 0e ff fa

00 00 00 0d

and\_pass:

1. R9 <- 0x100100CA; R8 <- 0x100100CA;
2. Branch to or\_pass if R9 = R8
3. If not branch, R14 <- 0xFFFFFFF9 and break.

01 e2 48 25

3c 08 10 01

35 08 00 ca

11 09 00 02

20 0e ff f9

00 00 00 0d

1. R9 <- 0xEFFEFF35; R8 <- 0xEFFEFF35;
2. Branch to nor\_pass if R9 = R8
3. If not branch, R14 <- 0xFFFFFFF8 and break

or\_pass:

01 e2 48 27

3c 08 ef fe

35 08 ff 35

11 09 00 02

20 0e ff f8

1. DM[D0] <- R8(0xEFFEFF35)
2. Clear R14, and break. PASS ALL !!

00 00 00 0d

nor\_pass:

ad e8 00 10

00 00 70 20

00 00 00 0d

If R1 < R2, branch to slt1.

If not, set fail flag. R14 <- 0xFFFFFFFF.

Set pass flag. DM[C0] <- 0xC0

subroutine:

00 22 18 2a

14 60 00 02

20 0e ff ff

00 00 00 0d slt1:

20 04 00 c0

ad e4 00 00

00 41 18 2b

If R2 < R1, branch to slt2.

If not, set fail flag. R14 <- 0xFFFFFFFE.

Set pass flag. DM[C4] <- 0xC4

14 60 00 02

20 0e ff fe

00 00 00 0d slt2:

20 05 00 c4

If R2 < R1, branch to slt2.

If not, set fail flag. R14 <- 0xFFFFFFFE.

Set pass flag. DM[C4] <- 0xC4

ad e5 00 04

00 41 18 2a

10 60 00 02

20 0e ff fd

00 00 00 0d slt3:

If R2 !< R1, branch to slt3.

If not, set fail flag. R14 <- 0xFFFFFFFD.

Set pass flag. DM[C8] <- 0xC8

20 06 00 c8

ad e6 00 08

00 22 18 2b

10 60 00 02

20 0e ff fc

00 00 00 0d slt4:

20 07 00 cc

ad e7 00 0c

If R1 !< R2, branch to slt4.

If not, set fail flag. R14 <- 0xFFFFFFFC.

Set pass flag. DM[CC] <- 0xCC

03 e0 00 08

Jump Register back to where Jump and Link was.

/\* Verification 9 \*/

BREAK INSTRUCTION FETCHED 1735.0 ns

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

t=1741.0 ns $r0 = 00000000 || t=1741.0 ns $r16 = xxxxxxxx

t=1741.0 ns $r1 = ffffff8a || t=1741.0 ns $r17 = xxxxxxxx

t=1741.0 ns $r2 = 0000008a || t=1741.0 ns $r18 = xxxxxxxx

t=1741.0 ns $r3 = 00000000 || t=1741.0 ns $r19 = xxxxxxxx

t=1741.0 ns $r4 = 000000c0 || t=1741.0 ns $r20 = xxxxxxxx

Scratch registers R1 – R3 and R8 – R13

Pass flags R4 – R7

t=1741.0 ns $r5 = 000000c4 || t=1741.0 ns $r21 = xxxxxxxx

t=1741.0 ns $r6 = 000000c8 || t=1741.0 ns $r22 = xxxxxxxx

t=1741.0 ns $r7 = 000000cc || t=1741.0 ns $r23 = xxxxxxxx

t=1741.0 ns $r8 = effeff35 || t=1741.0 ns $r24 = xxxxxxxx

t=1741.0 ns $r9 = effeff35 || t=1741.0 ns $r25 = xxxxxxxx

t=1741.0 ns $r10 = ffffffff || t=1741.0 ns $r26 = xxxxxxxx

t=1741.0 ns $r11 = ffffffff || t=1741.0 ns $r27 = xxxxxxxx

t=1741.0 ns $r12 = 88778877 || t=1741.0 ns $r28 = xxxxxxxx

If pass all tests, this should be 0x00

t=1741.0 ns $r13 = 77887788 || t=1741.0 ns $r29 = 000003fc

t=1741.0 ns $r14 = 00000000 || t=1741.0 ns $r30 = xxxxxxxx

t=1741.0 ns $r15 = 100100c0 || t=1741.0 ns $r31 = 00000014

time=1741.0 ns M[3F0]=xxxxxxxx

Data memory pointer.

\_\_\_\_\_\_\_Data Memory(DM)\_\_\_\_\_\_\_\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_IO Memory(IOM)\_\_\_\_\_

t=1741.0 ns DM[000000c0] = 000000c0 || t=1741.0 ns IOM[000000c0] = xxxxxxxx

t=1741.0 ns DM[000000c4] = 000000c4 || t=1741.0 ns IOM[000000c4] = xxxxxxxx

Pass flags from slt\_tests

t=1741.0 ns DM[000000c8] = 000000c8 || t=1741.0 ns IOM[000000c8] = xxxxxxxx

t=1741.0 ns DM[000000cc] = 000000cc || t=1741.0 ns IOM[000000cc] = xxxxxxxx

t=1741.0 ns DM[000000d0] = effeff35 || t=1741.0 ns IOM[000000d0] = xxxxxxxx

t=1741.0 ns DM[000000d4] = xxxxxxxx || t=1741.0 ns IOM[000000d4] = xxxxxxxx

t=1741.0 ns DM[000000d8] = xxxxxxxx || t=1741.0 ns IOM[000000d8] = xxxxxxxx

t=1741.0 ns DM[000000dc] = xxxxxxxx || t=1741.0 ns IOM[000000dc] = xxxxxxxx

t=1741.0 ns DM[000000e0] = xxxxxxxx || t=1741.0 ns IOM[000000e0] = xxxxxxxx

t=1741.0 ns DM[000000e4] = xxxxxxxx || t=1741.0 ns IOM[000000e4] = xxxxxxxx

t=1741.0 ns DM[000000e8] = xxxxxxxx || t=1741.0 ns IOM[000000e8] = xxxxxxxx

t=1741.0 ns DM[000000ec] = xxxxxxxx || t=1741.0 ns IOM[000000ec] = xxxxxxxx

t=1741.0 ns DM[000000f0] = xxxxxxxx || t=1741.0 ns IOM[000000f0] = xxxxxxxx

t=1741.0 ns DM[000000f4] = xxxxxxxx || t=1741.0 ns IOM[000000f4] = xxxxxxxx

t=1741.0 ns DM[000000f8] = xxxxxxxx || t=1741.0 ns IOM[000000f8] = xxxxxxxx

t=1741.0 ns DM[000000fc] = xxxxxxxx || t=1741.0 ns IOM[000000fc] = xxxxxxxx

Stopped at time : 1741 ns :

/\* Instruction Memory Module 10 \*/

Set Memory Pointer R15 <- 0x10010000

Set failed flag to R14 <- 0xFFFFFFFF

Then break.

Set passed flag to R14 <- 0x00000000

Then break.

@0

3c 0f 10 01

35 ef 00 00

8d e1 00 00

Load R1 through R8 from Data Memory

8d e2 00 04

8d e3 00 08

8d e4 00 0c

8d e5 00 10

1. Divide R1 by R2
2. Put[31:0] result into R09
3. Put[63:32] result into R10
4. Branch to fail1Q if R9 not equal to R5. Should not branch.
5. Branch to fail1R if R10 not equal to R6. Should not branch.

8d e6 00 14

8d e7 00 18

8d e8 00 1c

00 22 00 1a

00 00 48 12

00 00 50 10

15 25 00 16

1. Divide R2 by R3
2. Put[31:0] result into R09
3. Put[63:32] result into R10
4. Branch to fail2Q if R9 not equal to R7. Should not branch.
5. Branch to fail2R if R10 not equal to R8. Should not branch.

15 46 00 18

00 62 00 1a

00 00 48 12

00 00 50 10

15 27 00 17

15 48 00 19

1. Divide R4 by R1
2. Put[31:0] result into R09
3. Put[63:32] result into R10
4. Branch to fail3Q if R9 not equal to R7. Should not branch.
5. Branch to fail3R if R10 not equal to R6. Should not branch.

00 24 00 1a

00 00 48 12

00 00 50 10

15 27 00 18

15 46 00 1a

00 64 00 1a

00 00 48 12

00 00 50 10

1. Divide R4 by R3
2. Put[31:0] result into R09
3. Put[63:32] result into R10
4. Branch to fail4Q if R9 not equal to R4. Should not branch.
5. Branch to fail4R if R10 not equal to R8. Should not branch.

15 25 00 19

15 48 00 1b

3c 0b 00 00

35 6b 00 00

00 0b 60 20

00 0b 68 20

00 0b 70 20

Clear R11-R14 <- 0x00000000; Then break.

00 00 00 0d

3c 0e ff ff

Fail1Q: R14 <- 0xFFFFFFFF then break.

35 ce ff ff

00 00 00 0d

Fail1R: R14 <- 0xFFFFFFFE then break.

3c 0e ff ff

35 ce ff fe

00 00 00 0d

Fail2Q: R14 <- 0xFFFFFFFD then break.

3c 0e ff ff

35 ce ff fd

Fail2R: R14 <- 0xFFFFFFFC then break.

00 00 00 0d

3c 0e ff ff

35 ce ff fc

00 00 00 0d

3c 0e ff ff

35 ce ff fb

Fail3Q: R14 <- 0xFFFFFFFB then break.

00 00 00 0d

Fail3R: R14 <- 0xFFFFFFFA then break.

3c 0e ff ff

35 ce ff fa

Fail4Q: R14 <- 0xFFFFFFF9 then break.

00 00 00 0d

3c 0e ff ff

35 ce ff f9

00 00 00 0d

Fail4R: R14 <- 0xFFFFFFF8 then break.

3c 0e ff ff

35 ce ff f8

00 00 00 0d

/\* Verification 10 \*/

BREAK INSTRUCTION FETCHED 1465.0 ns

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

t=1471.0 ns $r0 = 00000000 || t=1471.0 ns $r16 = xxxxxxxx

Load R1 through R8 from Data Memory

t=1471.0 ns $r1 = 00040911 || t=1471.0 ns $r17 = xxxxxxxx

t=1471.0 ns $r2 = 000003e8 || t=1471.0 ns $r18 = xxxxxxxx

t=1471.0 ns $r3 = fffbf6ef || t=1471.0 ns $r19 = xxxxxxxx

t=1471.0 ns $r4 = fffffc18 || t=1471.0 ns $r20 = xxxxxxxx

t=1471.0 ns $r5 = 00000108 || t=1471.0 ns $r21 = xxxxxxxx

t=1471.0 ns $r6 = 000001d1 || t=1471.0 ns $r22 = xxxxxxxx

Last result from R3 and R4 division

t=1471.0 ns $r7 = fffffef8 || t=1471.0 ns $r23 = xxxxxxxx

t=1471.0 ns $r8 = fffffe2f || t=1471.0 ns $r24 = xxxxxxxx

t=1471.0 ns $r9 = 00000108 || t=1471.0 ns $r25 = xxxxxxxx

t=1471.0 ns $r10 = fffffe2f || t=1471.0 ns $r26 = xxxxxxxx

Passed flags

t=1471.0 ns $r11 = 00000000 || t=1471.0 ns $r27 = xxxxxxxx

t=1471.0 ns $r12 = 00000000 || t=1471.0 ns $r28 = xxxxxxxx

t=1471.0 ns $r13 = 00000000 || t=1471.0 ns $r29 = 000003fc

Data Memory pointer

t=1471.0 ns $r14 = 00000000 || t=1471.0 ns $r30 = xxxxxxxx

t=1471.0 ns $r15 = 10010000 || t=1471.0 ns $r31 = xxxxxxxx

time=1471.0 ns M[3F0]=xxxxxxxx

\_\_\_\_\_\_\_Data Memory(DM)\_\_\_\_\_\_\_\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_IO Memory(IOM)\_\_\_\_\_

t=1471.0 ns DM[000000c0] = xxxxxxxx || t=1471.0 ns IOM[000000c0] = xxxxxxxx

t=1471.0 ns DM[000000c4] = xxxxxxxx || t=1471.0 ns IOM[000000c4] = xxxxxxxx

t=1471.0 ns DM[000000c8] = xxxxxxxx || t=1471.0 ns IOM[000000c8] = xxxxxxxx

t=1471.0 ns DM[000000cc] = xxxxxxxx || t=1471.0 ns IOM[000000cc] = xxxxxxxx

t=1471.0 ns DM[000000d0] = xxxxxxxx || t=1471.0 ns IOM[000000d0] = xxxxxxxx

t=1471.0 ns DM[000000d4] = xxxxxxxx || t=1471.0 ns IOM[000000d4] = xxxxxxxx

t=1471.0 ns DM[000000d8] = xxxxxxxx || t=1471.0 ns IOM[000000d8] = xxxxxxxx

t=1471.0 ns DM[000000dc] = xxxxxxxx || t=1471.0 ns IOM[000000dc] = xxxxxxxx

t=1471.0 ns DM[000000e0] = xxxxxxxx || t=1471.0 ns IOM[000000e0] = xxxxxxxx

t=1471.0 ns DM[000000e4] = xxxxxxxx || t=1471.0 ns IOM[000000e4] = xxxxxxxx

t=1471.0 ns DM[000000e8] = xxxxxxxx || t=1471.0 ns IOM[000000e8] = xxxxxxxx

t=1471.0 ns DM[000000ec] = xxxxxxxx || t=1471.0 ns IOM[000000ec] = xxxxxxxx

t=1471.0 ns DM[000000f0] = xxxxxxxx || t=1471.0 ns IOM[000000f0] = xxxxxxxx

t=1471.0 ns DM[000000f4] = xxxxxxxx || t=1471.0 ns IOM[000000f4] = xxxxxxxx

t=1471.0 ns DM[000000f8] = xxxxxxxx || t=1471.0 ns IOM[000000f8] = xxxxxxxx

t=1471.0 ns DM[000000fc] = xxxxxxxx || t=1471.0 ns IOM[000000fc] = xxxxxxxx

Stopped at time : 1471 ns :

/\* Instruction Memory Module 11 \*/

Set Memory Pointer R15 <- 0x10010000

If R2 < R1, branch to slt2.

If not, set fail flag. R14 <- 0xFFFFFFFE.

Set pass flag. DM[C4] <- 0xC4

If R1 < R2, branch to slt1.

If not, set fail flag. R14 <- 0xFFFFFFFF.

Set pass flag. DM[C0] <- 0xC0

1. DM[D0] <- R8(0xEFFEFF35)
2. Clear R14, and break. PASS ALL !!

@0

3c 0f 10 01

35 ef 00 c0

R1 <- 0xFFFFFF8A; R2 <- 0x0000008A

20 01 ff 8a

20 02 00 8a

0c 10 00 1a

Jump and Link to slt\_tests

3c 0d ff ff

35 ad 55 55

3c 0c ff ff

R13 <- 0xFFFF5555;   
R12 <- 0xFFFFFAF5;  
R11 <- 0xFFFFFFFF;  
R10 <- 0x0000F0F0;

35 8c fa f5

3c 0b ff ff

35 6b ff ff

3c 0a 00 00

35 4a f0 f0

39 a9 aa aa

01 2b 40 22

1. R9 <- 0xFFFFFFFF; R8 <- 0x0
2. Branch to xor\_p1,
3. else R14 <- 0xFFFFFFF9, and break. (fail)

11 00 00 02

20 0e ff f9

00 00 00 0d xor\_p1:

31 87 f5 fa

00 ea 40 22

1. R7 <- 0x0000F0F0;
2. Branch to xor\_p2,
3. else R14 <- 0xFFFFFFF8 and break. (fail)

11 00 00 02

20 0e ff f8

00 00 00 0d xor\_p2:

ad e1 00 18

00 00 00 0d

00 00 00 0d

DM[D8] <- 0xFFFFFF8A and break; (pass)

subroutine

2c 23 ff 8b

14 60 00 02

1. Branch to slt\_p1 if R1 < 0xFF8B
2. And DM[C0] < 0xC0
3. Else R14 <- 0xFFFFFFFF (fail)

20 0e ff ff

00 00 00 0d slt\_p1:

20 04 00 c0

ad e4 00 00

2c 23 ff 89

1. Branch to slt\_p2 if R1 !< 0xFF89
2. And DM[C4] < 0xC4
3. Else R14 <- 0xFFFFFFFE (fail)

10 60 00 02

20 0e ff fe

00 00 00 0d slt\_p2:

20 05 00 c4

ad e5 00 04

2c 23 ff 8a

1. Branch to slt\_3 if R1 !< 0xFF8A
2. And DM[C8] < 0xC8
3. Else R14 <- 0xFFFFFFFD (fail)

10 60 00 02

20 0e ff fd

00 00 00 0d slt\_p3:

20 06 00 c8

ad e6 00 08

2c 43 00 8b

1. Branch to slt\_p4 if R2 < 0x008B
2. And DM[CC] < 0xCC
3. Else R14 <- 0xFFFFFFFC (fail)

14 60 00 02

20 0e ff fc

00 00 00 0d slt\_p4:

20 07 00 cc

ad e7 00 0c

2c 43 00 89

10 60 00 02

20 0e ff fb

1. Branch to slt\_p5 if R2 !< 0x0089
2. And DM[D0] < 0xD0
3. Else R14 <- 0xFFFFFFFB (fail)

00 00 00 0d slt\_p5:

20 08 00 d0

ad e8 00 10

1. Branch to slt\_p6 if R2 !< 0x008A
2. And DM[D4] < 0xD4
3. Else R14 <- 0xFFFFFFFA (fail)

2c 43 00 8a

10 60 00 02

20 0e ff fa

00 00 00 0d slt\_p6:

20 06 00 d4

ad e6 00 14

Jump register to where jump and link was

20 0e 00 00

03 e0 00 08

/\* Verification 11 \*/

BREAK INSTRUCTION FETCHED 1895.0 ns

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

Comparison values

t=1901.0 ns $r0 = 00000000 || t=1901.0 ns $r16 = xxxxxxxx

t=1901.0 ns $r1 = ffffff8a || t=1901.0 ns $r17 = xxxxxxxx

t=1901.0 ns $r2 = 0000008a || t=1901.0 ns $r18 = xxxxxxxx

t=1901.0 ns $r3 = 00000000 || t=1901.0 ns $r19 = xxxxxxxx

t=1901.0 ns $r4 = 000000c0 || t=1901.0 ns $r20 = xxxxxxxx

Scratch registers

t=1901.0 ns $r5 = 000000c4 || t=1901.0 ns $r21 = xxxxxxxx

t=1901.0 ns $r6 = 000000d4 || t=1901.0 ns $r22 = xxxxxxxx

t=1901.0 ns $r7 = 0000f0f0 || t=1901.0 ns $r23 = xxxxxxxx

t=1901.0 ns $r8 = 00000000 || t=1901.0 ns $r24 = xxxxxxxx

t=1901.0 ns $r9 = ffffffff || t=1901.0 ns $r25 = xxxxxxxx

t=1901.0 ns $r10 = 0000f0f0 || t=1901.0 ns $r26 = xxxxxxxx

Pattern starters

t=1901.0 ns $r11 = ffffffff || t=1901.0 ns $r27 = xxxxxxxx

t=1901.0 ns $r12 = fffffaf5 || t=1901.0 ns $r28 = xxxxxxxx

t=1901.0 ns $r13 = ffff5555 || t=1901.0 ns $r29 = 000003fc

t=1901.0 ns $r14 = 00000000 || t=1901.0 ns $r30 = xxxxxxxx

t=1901.0 ns $r15 = 100100c0 || t=1901.0 ns $r31 = 00000014

Memory pointer

time=1901.0 ns M[3F0]=xxxxxxxx

\_\_\_\_\_\_\_Data Memory(DM)\_\_\_\_\_\_\_\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_IO Memory(IOM)\_\_\_\_\_

t=1901.0 ns DM[000000c0] = 000000c0 || t=1901.0 ns IOM[000000c0] = xxxxxxxx

t=1901.0 ns DM[000000c4] = 000000c4 || t=1901.0 ns IOM[000000c4] = xxxxxxxx

Pass flags

t=1901.0 ns DM[000000c8] = 000000c8 || t=1901.0 ns IOM[000000c8] = xxxxxxxx

t=1901.0 ns DM[000000cc] = 000000cc || t=1901.0 ns IOM[000000cc] = xxxxxxxx

t=1901.0 ns DM[000000d0] = 000000d0 || t=1901.0 ns IOM[000000d0] = xxxxxxxx

t=1901.0 ns DM[000000d4] = 000000d4 || t=1901.0 ns IOM[000000d4] = xxxxxxxx

t=1901.0 ns DM[000000d8] = ffffff8a || t=1901.0 ns IOM[000000d8] = xxxxxxxx

Stored R1 proving pasted all tests.

t=1901.0 ns DM[000000dc] = xxxxxxxx || t=1901.0 ns IOM[000000dc] = xxxxxxxx

t=1901.0 ns DM[000000e0] = xxxxxxxx || t=1901.0 ns IOM[000000e0] = xxxxxxxx

t=1901.0 ns DM[000000e4] = xxxxxxxx || t=1901.0 ns IOM[000000e4] = xxxxxxxx

t=1901.0 ns DM[000000e8] = xxxxxxxx || t=1901.0 ns IOM[000000e8] = xxxxxxxx

t=1901.0 ns DM[000000ec] = xxxxxxxx || t=1901.0 ns IOM[000000ec] = xxxxxxxx

t=1901.0 ns DM[000000f0] = xxxxxxxx || t=1901.0 ns IOM[000000f0] = xxxxxxxx

t=1901.0 ns DM[000000f4] = xxxxxxxx || t=1901.0 ns IOM[000000f4] = xxxxxxxx

t=1901.0 ns DM[000000f8] = xxxxxxxx || t=1901.0 ns IOM[000000f8] = xxxxxxxx

t=1901.0 ns DM[000000fc] = xxxxxxxx || t=1901.0 ns IOM[000000fc] = xxxxxxxx

Stopped at time : 1901 ns

/\* Instruction Memory Module 12 \*/

@0

Set memory pointer R15 <- 0x100100C0  
R1 <- 0xFFFFFF8A  
R2 <- 0x0000008A  
Jump and Link to blt\_tests  
At the return, DM[D8] <- R1, DM[DC] <- R2, then break.

3c 0f 10 01

35 ef 00 c0

20 01 ff 8a

20 02 00 8a

0c 10 00 08

ad e1 00 18

ad e2 00 1c

00 00 00 0d

Branch to blez\_p1 if R1 <= 0 (pass)

18 20 00 02 blt\_tests

20 0e ff ff

Branch to blez\_f2 if R2 <= 0  
DM[C0] <- 0xC0, DM[C4] <- 0xC4  
R14 <- 0XFFFFFFFE (fail)

00 00 00 0d

20 03 00 c0 blez\_p1:

ad e3 00 00

18 40 00 03

20 04 00 c4

ad e4 00 04

Branch to blez\_p3 if R0 <= 0 (pass)  
DM[C8] <- 0xC8  
R14 <- 0XFFFFFFFD (fail)

08 10 00 13

20 0e ff fe blez\_f2:

00 00 00 0d

18 00 00 02 blez\_p2:

20 0e ff fd

00 00 00 0d

20 05 00 c8 blez\_p3:

Branch to bgtz\_p1 if (R2>0) (pass)

ad e5 00 08

1c 40 00 02

20 0e ff fc

Branch to bgtz\_f2 if (R1>0 ) (fail)  
DM[CC] <- 0xCC, DM[D0] <- D0  
R14 <- 0xFFFFFFFB (fail)

00 00 00 0d

20 06 00 cc bgtz\_p1:

ad e6 00 0c

1c 20 00 03

20 07 00 d0

ad e7 00 10

Branch to bgtz\_f2 if (R1 >0) (fail)  
DM[D4 <- 0xD4  
R14 <- 0xFFFFFFFA (fail)

08 10 00 23

20 0e ff fb bgtz\_f2:

00 00 00 0d

1c 20 00 03 bgtz\_p2:

20 08 00 d4

ad e8 00 14

08 10 00 29

Return to where jump and link was and clear R14 <- 0x00000000

20 0e ff fa bgtz\_f3:

00 00 00 0d

20 0e 00 00 bgtz\_p3:

03 e0 00 08

/\* Verification 12 \*/

BREAK INSTRUCTION FETCHED 1255.0 ns

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

t=1261.0 ns $r0 = 00000000 || t=1261.0 ns $r16 = xxxxxxxx

Data for comparisions

t=1261.0 ns $r1 = ffffff8a || t=1261.0 ns $r17 = xxxxxxxx

t=1261.0 ns $r2 = 0000008a || t=1261.0 ns $r18 = xxxxxxxx

t=1261.0 ns $r3 = 000000c0 || t=1261.0 ns $r19 = xxxxxxxx

t=1261.0 ns $r4 = 000000c4 || t=1261.0 ns $r20 = xxxxxxxx

Pass flags

t=1261.0 ns $r5 = 000000c8 || t=1261.0 ns $r21 = xxxxxxxx

t=1261.0 ns $r6 = 000000cc || t=1261.0 ns $r22 = xxxxxxxx

t=1261.0 ns $r7 = 000000d0 || t=1261.0 ns $r23 = xxxxxxxx

Return address

t=1261.0 ns $r8 = 000000d4 || t=1261.0 ns $r24 = xxxxxxxx

t=1261.0 ns $r9 = xxxxxxxx || t=1261.0 ns $r25 = xxxxxxxx

t=1261.0 ns $r10 = xxxxxxxx || t=1261.0 ns $r26 = xxxxxxxx

Memory pointer

t=1261.0 ns $r11 = xxxxxxxx || t=1261.0 ns $r27 = xxxxxxxx

t=1261.0 ns $r12 = xxxxxxxx || t=1261.0 ns $r28 = xxxxxxxx

t=1261.0 ns $r13 = xxxxxxxx || t=1261.0 ns $r29 = 000003fc

t=1261.0 ns $r14 = 00000000 || t=1261.0 ns $r30 = xxxxxxxx

t=1261.0 ns $r15 = 100100c0 || t=1261.0 ns $r31 = 00000014

time=1261.0 ns M[3F0]=xxxxxxxx

\_\_\_\_\_\_\_Data Memory(DM)\_\_\_\_\_\_\_\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_IO Memory(IOM)\_\_\_\_\_

R1, R2 as pass flags after passed blt\_test

t=1261.0 ns DM[000000c0] = 000000c0 || t=1261.0 ns IOM[000000c0] = xxxxxxxx

t=1261.0 ns DM[000000c4] = 000000c4 || t=1261.0 ns IOM[000000c4] = xxxxxxxx

t=1261.0 ns DM[000000c8] = 000000c8 || t=1261.0 ns IOM[000000c8] = xxxxxxxx

t=1261.0 ns DM[000000cc] = 000000cc || t=1261.0 ns IOM[000000cc] = xxxxxxxx

Pass flags

t=1261.0 ns DM[000000d0] = 000000d0 || t=1261.0 ns IOM[000000d0] = xxxxxxxx

t=1261.0 ns DM[000000d4] = 000000d4 || t=1261.0 ns IOM[000000d4] = xxxxxxxx

t=1261.0 ns DM[000000d8] = ffffff8a || t=1261.0 ns IOM[000000d8] = xxxxxxxx

t=1261.0 ns DM[000000dc] = 0000008a || t=1261.0 ns IOM[000000dc] = xxxxxxxx

R1, R2 as pass flags after passed all instructions

t=1261.0 ns DM[000000e0] = xxxxxxxx || t=1261.0 ns IOM[000000e0] = xxxxxxxx

t=1261.0 ns DM[000000e4] = xxxxxxxx || t=1261.0 ns IOM[000000e4] = xxxxxxxx

t=1261.0 ns DM[000000e8] = xxxxxxxx || t=1261.0 ns IOM[000000e8] = xxxxxxxx

t=1261.0 ns DM[000000ec] = xxxxxxxx || t=1261.0 ns IOM[000000ec] = xxxxxxxx

t=1261.0 ns DM[000000f0] = xxxxxxxx || t=1261.0 ns IOM[000000f0] = xxxxxxxx

t=1261.0 ns DM[000000f4] = xxxxxxxx || t=1261.0 ns IOM[000000f4] = xxxxxxxx

t=1261.0 ns DM[000000f8] = xxxxxxxx || t=1261.0 ns IOM[000000f8] = xxxxxxxx

t=1261.0 ns DM[000000fc] = xxxxxxxx || t=1261.0 ns IOM[000000fc] = xxxxxxxx

Stopped at time : 1261 ns

/\* Instruction Memory Module 13 \*/

@0

Set interrupt flag to 1

00 00 00 1f

3c 01 12 34

34 21 56 78

3c 02 87 65

R1 <- 0x12345678  
R2 <- 0x87654321  
R3 <- 0xABCDEF01  
R4 <- 0x0FEDCBA  
R5 <- 0x5A5A5A5A  
R6 <- 0xFFFFFF00

34 42 43 21

3c 03 ab cd

34 63 ef 01

3c 04 01 fe

34 84 dc ba

3c 05 5a 5a

34 a5 5a 5a

3c 06 ff ff

34 c6 ff ff

3c 07 ff ff

Add R6 with R7 through R14 to R8 through R15.

34 e7 ff 00

00 c7 40 20

00 c8 48 20

00 c9 50 20

00 ca 58 20

00 cb 60 20

R7 <- 100103F0  
DM[3F0] <- R15

00 cc 68 20

00 cd 70 20

00 ce 78 20

3c 07 10 01

Interrupt Service Routine

34 e7 03 f0

ac ef 00 00

00 00 00 0d

R16 <- 0x100100C0 // IOmem address pointer  
R17 <- 0x8000FFFF // pattern  
R18 <- 0x0010 // loop counter

@200

3c 10 10 01

36 10 00 c0

3c 11 80 00

36 31 ff ff

20 12 00 10

IOM[R16] <- R17  
shift right arithmetic by 2 bits  
Increment memory pointer  
Decrement loop counter  
Branch to out\_IO if R18 != 0

76 11 00 00 out\_IO:

00 11 88 83

22 10 00 04

22 52 ff ff

16 40 ff fb

3c 10 10 01

36 10 00 c0

R16 <- 0x100100C0 // Set memory pointer

72 13 00 00

72 14 00 04

72 15 00 08

72 16 00 0c

72 17 00 10

Input R19 to R24 from IOM[R16]   
Increment R16 by 4

72 18 00 14

03 e0 00 08

Jump register to return address (R31)

/\* Verification 13 \*/

Memory pointer

Loaded R1-R6

BREAK INSTRUCTION FETCHED 4955.0 ns

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

Result from shifting

t=4961.0 ns $r0 = 00000000 || t=4961.0 ns $r16 = 100100c0

t=4961.0 ns $r1 = 12345678 || t=4961.0 ns $r17 = ffffffff

t=4961.0 ns $r2 = 87654321 || t=4961.0 ns $r18 = 00000000

Loop counter

t=4961.0 ns $r3 = abcdef01 || t=4961.0 ns $r19 = 8000ffff

Results from Add instructions

t=4961.0 ns $r4 = 01fedcba || t=4961.0 ns $r20 = e0003fff

t=4961.0 ns $r5 = 5a5a5a5a || t=4961.0 ns $r21 = f8000fff

Input from IOM

t=4961.0 ns $r6 = ffffffff || t=4961.0 ns $r22 = fe0003ff

t=4961.0 ns $r7 = 100103f0 || t=4961. 0 ns $r23 = ff8000ff

t=4961.0 ns $r8 = fffffeff || t=4961.0 ns $r24 = ffe0003f

t=4961.0 ns $r9 = fffffefe || t=4961.0 ns $r25 = xxxxxxxx

t=4961.0 ns $r10 = fffffefd || t=4961.0 ns $r26 = xxxxxxxx

t=4961.0 ns $r11 = fffffefc || t=4961.0 ns $r27 = xxxxxxxx

t=4961.0 ns $r12 = fffffefb || t=4961.0 ns $r28 = xxxxxxxx

t=4961.0 ns $r13 = fffffefa || t=4961.0 ns $r29 = 000003fc

t=4961.0 ns $r14 = fffffef9 || t=4961.0 ns $r30 = xxxxxxxx

t=4961.0 ns $r15 = fffffef8 || t=4961.0 ns $r31 = 00000064

Results from shifting

time=4961.0 ns M[3F0]=fffffef8

\_\_\_\_\_\_\_Data Memory(DM)\_\_\_\_\_\_\_\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_IO Memory(IOM)\_\_\_\_\_

t=4961.0 ns DM[000000c0] = xxxxxxxx || t=4961.0 ns IOM[000000c0] = 8000ffff

t=4961.0 ns DM[000000c4] = xxxxxxxx || t=4961.0 ns IOM[000000c4] = e0003fff

t=4961.0 ns DM[000000c8] = xxxxxxxx || t=4961.0 ns IOM[000000c8] = f8000fff

t=4961.0 ns DM[000000cc] = xxxxxxxx || t=4961.0 ns IOM[000000cc] = fe0003ff

t=4961.0 ns DM[000000d0] = xxxxxxxx || t=4961.0 ns IOM[000000d0] = ff8000ff

t=4961.0 ns DM[000000d4] = xxxxxxxx || t=4961.0 ns IOM[000000d4] = ffe0003f

t=4961.0 ns DM[000000d8] = xxxxxxxx || t=4961.0 ns IOM[000000d8] = fff8000f

t=4961.0 ns DM[000000dc] = xxxxxxxx || t=4961.0 ns IOM[000000dc] = fffe0003

t=4961.0 ns DM[000000e0] = xxxxxxxx || t=4961.0 ns IOM[000000e0] = ffff8000

t=4961.0 ns DM[000000e4] = xxxxxxxx || t=4961.0 ns IOM[000000e4] = ffffe000

t=4961.0 ns DM[000000e8] = xxxxxxxx || t=4961.0 ns IOM[000000e8] = fffff800

t=4961.0 ns DM[000000ec] = xxxxxxxx || t=4961.0 ns IOM[000000ec] = fffffe00

t=4961.0 ns DM[000000f0] = xxxxxxxx || t=4961.0 ns IOM[000000f0] = ffffff80

t=4961.0 ns DM[000000f4] = xxxxxxxx || t=4961.0 ns IOM[000000f4] = ffffffe0

t=4961.0 ns DM[000000f8] = xxxxxxxx || t=4961.0 ns IOM[000000f8] = fffffff8

t=4961.0 ns DM[000000fc] = xxxxxxxx || t=4961.0 ns IOM[000000fc] = fffffffe

Stopped at time : 4961 ns

/\* Instruction Memory Module 14 \*/

@0

R1 <- 0x12345678  
R2 <- 0x87654321  
R3 <- 0xABCDEF01  
R4 <- 0x0FEDCBA  
R5 <- 0x5A5A5A5A  
R6 <- 0xFFFFFF00

Set interrupt flag to 1

Add R6 with R7 through R14 to R8 through R15.

R7 <- 100103F0  
DM[3F0] <- R15

00 00 00 1f

3c 01 12 34

34 21 56 78

3c 02 87 65

34 42 43 21

3c 03 ab cd

34 63 ef 01

3c 04 01 fe

34 84 dc ba

3c 05 5a 5a

34 a5 5a 5a

3c 06 ff ff

34 c6 ff ff

3c 07 ff ff

34 e7 ff 00

00 c7 40 20

00 c8 48 20

00 c9 50 20

00 ca 58 20

Interrupt Service Routine

Push current PC and flags on the stack

00 cb 60 20

00 cc 68 20

00 cd 70 20

00 ce 78 20

3c 07 10 01

34 e7 03 f0

ac ef 00 00

00 00 00 0d

R16 <- 0x100100C0 // IOmem address pointer  
R17 <- 0x8000FFFF // pattern  
R18 <- 0x0010 // loop counter

@200

3c 10 10 01

36 10 00 c0

3c 11 80 00

36 31 ff ff

IOM[R16] <- R17  
shift right arithmetic by 2 bits  
Increment memory pointer  
Decrement loop counter  
Branch to out\_IO if R18 != 0

20 12 00 10

76 11 00 00

00 11 88 83

22 10 00 04

22 52 ff ff

16 40 ff fb

3c 10 10 01

36 10 00 c0

72 13 00 00

R16 <- 0x100100C0 // Set memory pointer

72 14 00 04

72 15 00 08

72 16 00 0c

72 17 00 10

Input R19 to R24 from IOM[R16]   
Increment R16 by 4

72 18 00 14

7B A0 00 00

Return Interrupt Routine  
Pop the flags and PC back from stack

/\* Verification 14 \*/

Results from shifting

Memory pointer

Loaded R1-R6

BREAK INSTRUCTION FETCHED 5025.0 ns

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

Result from shifting

t=5031.0 ns $r0 = 00000000 || t=5031.0 ns $r16 = 100100c0

t=5031.0 ns $r1 = 12345678 || t=5031.0 ns $r17 = ffffffff

Loop counter

t=5031.0 ns $r2 = 87654321 || t=5031.0 ns $r18 = 00000000

Results from Add instructions

t=5031.0 ns $r3 = abcdef01 || t=5031.0 ns $r19 = 8000ffff

t=5031.0 ns $r4 = 01fedcba || t=5031.0 ns $r20 = e0003fff

t=5031.0 ns $r5 = 5a5a5a5a || t=5031.0 ns $r21 = f8000fff

Input from IOM

t=5031.0 ns $r6 = ffffffff || t=5031.0 ns $r22 = fe0003ff

t=5031.0 ns $r7 = 100103f0 || t=5031.0 ns $r23 = ff8000ff

t=5031.0 ns $r8 = fffffeff || t=5031.0 ns $r24 = ffe0003f

t=5031.0 ns $r9 = fffffefe || t=5031.0 ns $r25 = xxxxxxxx

t=5031.0 ns $r10 = fffffefd || t=5031.0 ns $r26 = xxxxxxxx

t=5031.0 ns $r11 = fffffefc || t=5031.0 ns $r27 = xxxxxxxx

t=5031.0 ns $r12 = fffffefb || t=5031.0 ns $r28 = xxxxxxxx

t=5031.0 ns $r13 = fffffefa || t=5031.0 ns $r29 = 100103f0

t=5031.0 ns $r14 = fffffef9 || t=5031.0 ns $r30 = xxxxxxxx

t=5031.0 ns $r15 = fffffef8 || t=5031.0 ns $r31 = 00000064

time=5031.0 ns M[3F0]=fffffef8

\_\_\_\_\_\_\_Data Memory(DM)\_\_\_\_\_\_\_\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_IO Memory(IOM)\_\_\_\_\_

t=5031.0 ns DM[000000c0] = xxxxxxxx || t=5031.0 ns IOM[000000c0] = 8000ffff

t=5031.0 ns DM[000000c4] = xxxxxxxx || t=5031.0 ns IOM[000000c4] = e0003fff

t=5031.0 ns DM[000000c8] = xxxxxxxx || t=5031.0 ns IOM[000000c8] = f8000fff

t=5031.0 ns DM[000000cc] = xxxxxxxx || t=5031.0 ns IOM[000000cc] = fe0003ff

t=5031.0 ns DM[000000d0] = xxxxxxxx || t=5031.0 ns IOM[000000d0] = ff8000ff

t=5031.0 ns DM[000000d4] = xxxxxxxx || t=5031.0 ns IOM[000000d4] = ffe0003f

t=5031.0 ns DM[000000d8] = xxxxxxxx || t=5031.0 ns IOM[000000d8] = fff8000f

t=5031.0 ns DM[000000dc] = xxxxxxxx || t=5031.0 ns IOM[000000dc] = fffe0003

t=5031.0 ns DM[000000e0] = xxxxxxxx || t=5031.0 ns IOM[000000e0] = ffff8000

t=5031.0 ns DM[000000e4] = xxxxxxxx || t=5031.0 ns IOM[000000e4] = ffffe000

t=5031.0 ns DM[000000e8] = xxxxxxxx || t=5031.0 ns IOM[000000e8] = fffff800

t=5031.0 ns DM[000000ec] = xxxxxxxx || t=5031.0 ns IOM[000000ec] = fffffe00

t=5031.0 ns DM[000000f0] = xxxxxxxx || t=5031.0 ns IOM[000000f0] = ffffff80

t=5031.0 ns DM[000000f4] = xxxxxxxx || t=5031.0 ns IOM[000000f4] = ffffffe0

t=5031.0 ns DM[000000f8] = xxxxxxxx || t=5031.0 ns IOM[000000f8] = fffffff8

t=5031.0 ns DM[000000fc] = xxxxxxxx || t=5031.0 ns IOM[000000fc] = fffffffe

Stopped at time : 5031 ns

/\* Instruction Memory Module for Enhancement \*/

@0

3c 01 12 34

R1 <- 0x01234567  
R2 <- 0x87654321  
R3 <- 0xABCDEF01

DM[3F8] <- R1; DM[3F4] <- R2; DM[3F0] <- R3

34 21 56 78

7c 01 00 09

3c 02 87 65

34 42 43 21

7c 02 00 09

3c 03 ab cd

34 63 ef 01

7c 03 00 09

No\_operation

7c 00 00 08

7c 04 00 0a

R4 <- DM[3F0] (0xABCDEF01)  
R5 <- DM[3F4] (0x87654321)  
R6 <- DM[3F8] (0x01234567)  
Branch to blt\_p if R1 < R2  
else R16 <- 0xFFFFFFFF (fail) then break.

7c 05 00 0a

7c 06 00 0a

b0 22 00 03

3c 10 ff ff

36 10 ff ff

00 00 00 0d

3c 10 11 11 blt\_p:

36 10 11 11

R16 <- 0x11111111 (pass)  
Branch to bge\_p if R6 >= zero  
else R17 <- 0xFFFFFFFF (fail) then break.

b4 c0 00 03

3c 11 ff ff

36 31 ff ff

00 00 00 0d

7e 11 00 07 bge\_p:

7c 11 38 81

R17 <- R16(0x11111111) (pass)  
Rotate Left R17 by 2; R7 <- 0x44444444  
Rotate Right R17 by 3; R8 <- 0x22222222

7c 11 40 c2

20 0d 00 07

20 0e 00 07

3c 09 00 00

35 29 00 01

3c 0f 10 01

Initiate loop\_counter1 R13 <- 0x07  
Initiate loop\_counter2 R14 <- 0x07  
R9 <- 0x00000001  
set Memory pointer R15 <- 0x100100C0

35 ef 00 c0

75 e9 00 00 IO:

00 09 49 00

21 ef 00 04

b9 ad ff fc

7d 2a 00 07

Store R9 in IO memory at R15 address  
IOM[R15] <- R9;  
Shift left logic R9 by 4  
Increment pointer R15 <- R15 +4  
Loop until R13 = zero, then R10 <- R9

ad ea 00 00 DM:

00 0a 51 02

21 ef 00 04

b9 ce ff fc

7c 01 00 05

00 00 00 0d

Break.

DM[R15] <- R10;  
Shift right logic R10 by 4  
Increment pointer R15 <- R15 +4  
Loop until R14 = zero, then R1 <- 0x00

/\* Verification Enhancement \*/

Pass all instructions.

BREAK INSTRUCTION FETCHED 3565.0 ns

Pass flags

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

t=3571.0 ns $r0 = 00000000 || t=3571.0 ns $r16 = 11111111

t=3571.0 ns $r1 = 00000000 || t=3571.0 ns $r17 = 11111111

t=3571.0 ns $r2 = 87654321 || t=3571.0 ns $r18 = xxxxxxxx

t=3571.0 ns $r3 = abcdef01 || t=3571.0 ns $r19 = xxxxxxxx

t=3571.0 ns $r4 = abcdef01 || t=3571.0 ns $r20 = xxxxxxxx

Load Immediate to R1-R3 and push them on Data memory stack.

t=3571.0 ns $r5 = 87654321 || t=3571.0 ns $r21 = xxxxxxxx

t=3571.0 ns $r6 = 12345678 || t=3571.0 ns $r22 = xxxxxxxx

t=3571.0 ns $r7 = 44444444 || t=3571.0 ns $r23 = xxxxxxxx

t=3571.0 ns $r8 = 22222222 || t=3571.0 ns $r24 = xxxxxxxx

t=3571.0 ns $r9 = 10000000 || t=3571.0 ns $r25 = xxxxxxxx

Pop data from the stack to R4-R6

t=3571.0 ns $r10 = 00000001 || t=3571.0 ns $r26 = xxxxxxxx

Shows that 0xABCDEF01 was pushed to this address

t=3571.0 ns $r11 = xxxxxxxx || t=3571.0 ns $r27 = xxxxxxxx

t=3571.0 ns $r12 = xxxxxxxx || t=3571.0 ns $r28 = xxxxxxxx

t=3571.0 ns $r13 = 00000000 || t=3571.0 ns $r29 = 000003fc

Loop counters

t=3571.0 ns $r14 = 00000000 || t=3571.0 ns $r30 = xxxxxxxx

t=3571.0 ns $r15 = 100100f8 || t=3571.0 ns $r31 = xxxxxxxx

Memory pointer

Result from shiftings

Results from ROTL and ROTR

time=3571.0 ns M[3F0]=abcdef01

\_\_\_\_\_\_\_Data Memory(DM)\_\_\_\_\_\_\_\_\_\_\_\_\_\_||\_\_\_\_\_\_\_\_\_IO Memory(IOM)\_\_\_\_\_

t=3571.0 ns DM[000000c0] = xxxxxxxx || t=3571.0 ns IOM[000000c0] = 00000001

t=3571.0 ns DM[000000c4] = xxxxxxxx || t=3571.0 ns IOM[000000c4] = 00000010

t=3571.0 ns DM[000000c8] = xxxxxxxx || t=3571.0 ns IOM[000000c8] = 00000100

t=3571.0 ns DM[000000cc] = xxxxxxxx || t=3571.0 ns IOM[000000cc] = 00001000

t=3571.0 ns DM[000000d0] = xxxxxxxx || t=3571.0 ns IOM[000000d0] = 00010000

t=3571.0 ns DM[000000d4] = xxxxxxxx || t=3571.0 ns IOM[000000d4] = 00100000

t=3571.0 ns DM[000000d8] = xxxxxxxx || t=3571.0 ns IO [000000d8] = 01000000

t=3571.0 ns DM[000000dc] = 10000000 || t=3571.0 ns IOM[000000dc] = xxxxxxxx

t=3571.0 ns DM[000000e0] = 01000000 || t=3571.0 ns IOM[000000e0] = xxxxxxxx

t=3571.0 ns DM[000000e4] = 00100000 || t=3571.0 ns IOM[000000e4] = xxxxxxxx

Walking one forward from looping

t=3571.0 ns DM[000000e8] = 00010000 || t=3571.0 ns IOM[000000e8] = xxxxxxxx

t=3571.0 ns DM[000000ec] = 00001000 || t=3571.0 ns IOM[000000ec] = xxxxxxxx

t=3571.0 ns DM[000000f0] = 00000100 || t=3571.0 ns IOM[000000f0] = xxxxxxxx

t=3571.0 ns DM[000000f4] = 00000010 || t=3571.0 ns IOM[000000f4] = xxxxxxxx

t=3571.0 ns DM[000000f8] = xxxxxxxx || t=3571.0 ns IOM[000000f8] = xxxxxxxx

Walking one backward from looping

t=3571.0 ns DM[000000fc] = xxxxxxxx || t=3571.0 ns IOM[000000fc] = xxxxxxxx

Stopped at time : 3571 ns