CECS 440 Senior Design Review

By Julie C Kim

Agenda:

- Introduction
- Baseline MIPS Architecture Brief Overview
- MIPS Hardware Implementation
- Enhancement Instructions and verification
- Future Enhancement Hardware Architecture and Current Result

Regular Processor

MIPS 32 bit Compatible Processor

History of Processor Name:

The Four Design Principle:

- Simplicity Favor **Regularity**
- Smaller is Faster
- Make Common Case Fast
- Good Design Demand Good Compromises

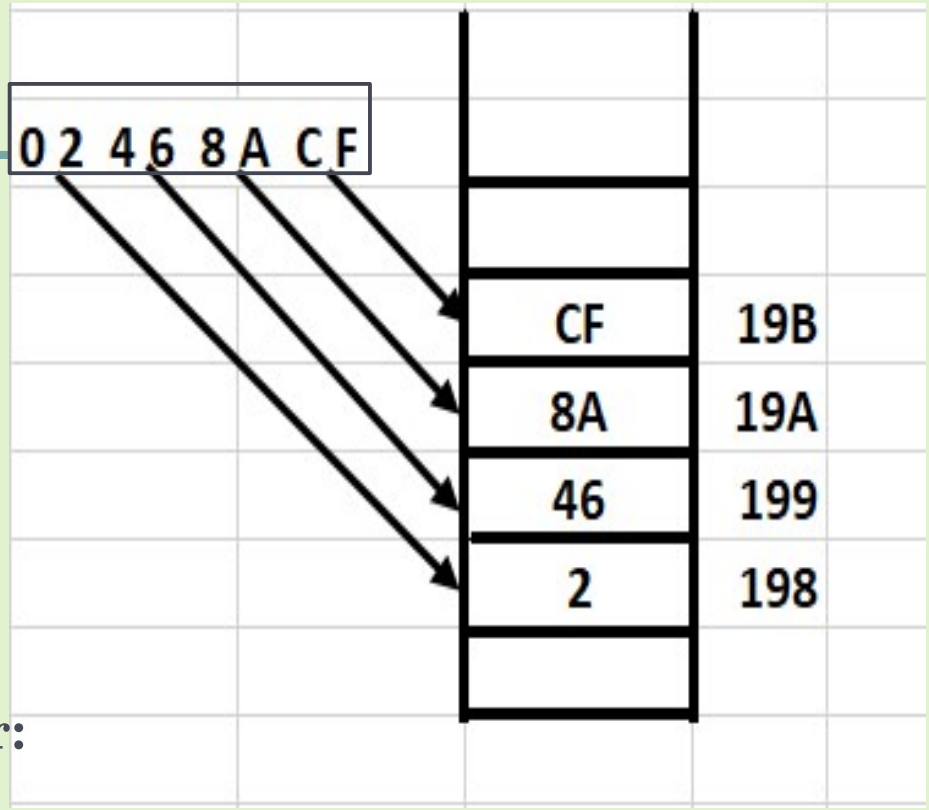
MIPS Processor Architecture Overview

- **Memory Structure:**
  - Harvard Memory Architecture
  - Byte Addressable
- **Data Type:**
  - Unsigned :

  0 to 4,294,967,295
  - Signed :

  -2,147,483,647, to 2,147,483,647
- **Processor Register Set:**
  - Register File: 32 Registers
  - One Program Counter Register: 32 bit wide
  - One Flags Register: 5 bit

Big Endian Format



Flags Register	IE	C	V	N	Z

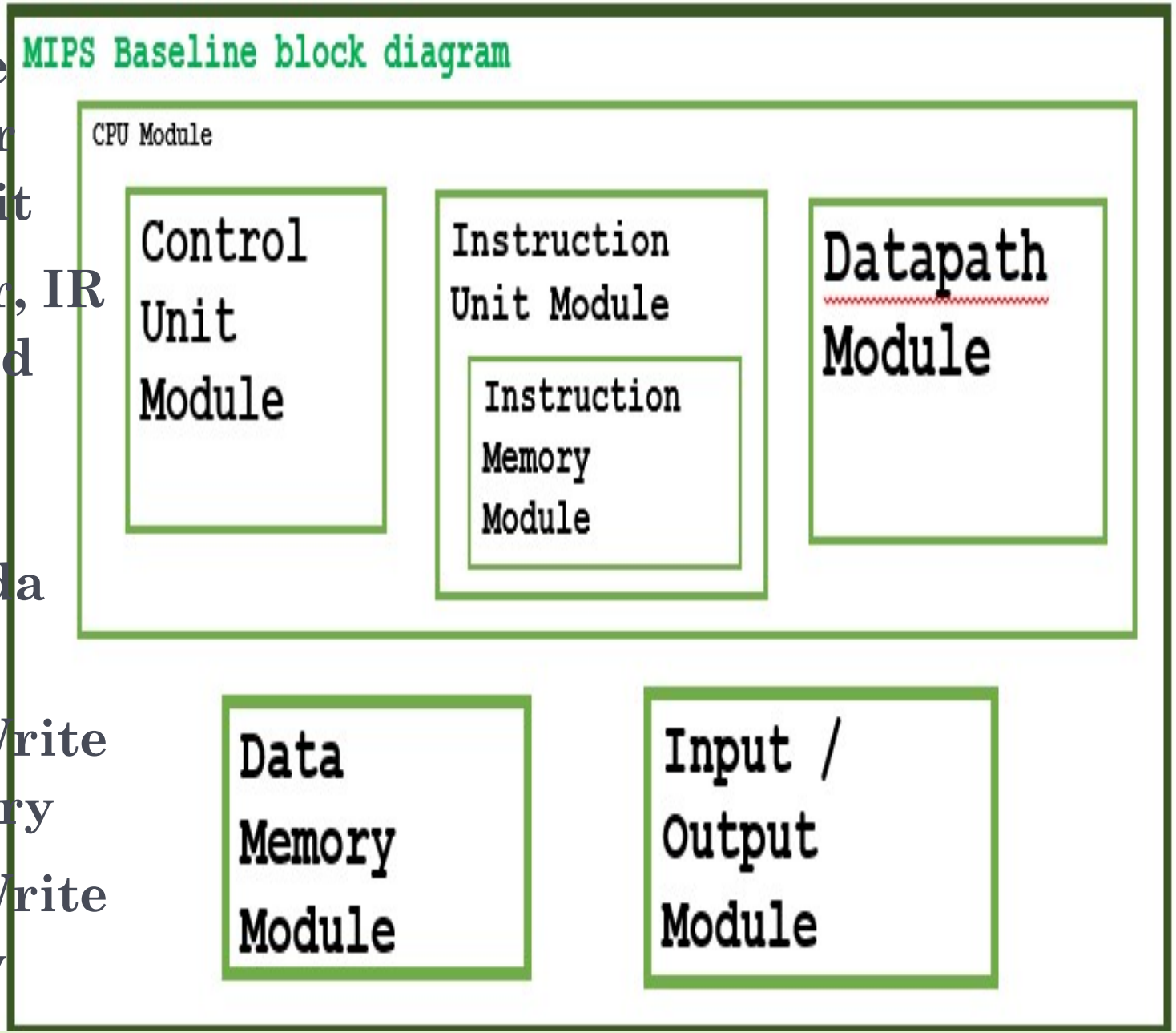
MIPS Processor and Architecture Overview continue

- Addressing Mode:
  - Immediate:
  - Register Direct
  - Register Indirect
- Instruction Set:
  - R-Type: Arithmetic Operation
  - I-Type : Load, Store
  - J-Type: Jump, Jal

	31 26	25 21	20 16	15 11	10 6	5 0
R_type:	6-bit Opcode	5-bit sonce addr	5-bit source addr	5-bit dst addr	shtamnt	5-bit fnt code
I-type:	6-bit Opcode	5-bit base addr	5-bit code	16-bit immediate		
J-type:	6-bit Opcode	26-bit Jump Address				

General Diagram of Hardware Architecture:

- Finite State Machine for Control Unit
- PC Register, IR Register and Instruction Memory
- Integer Dada Flow
- Read and Write Data Memory
- Read and Write I/O Memory



Over View of How Data Flow:

FETCH

↓

DECODE

↓

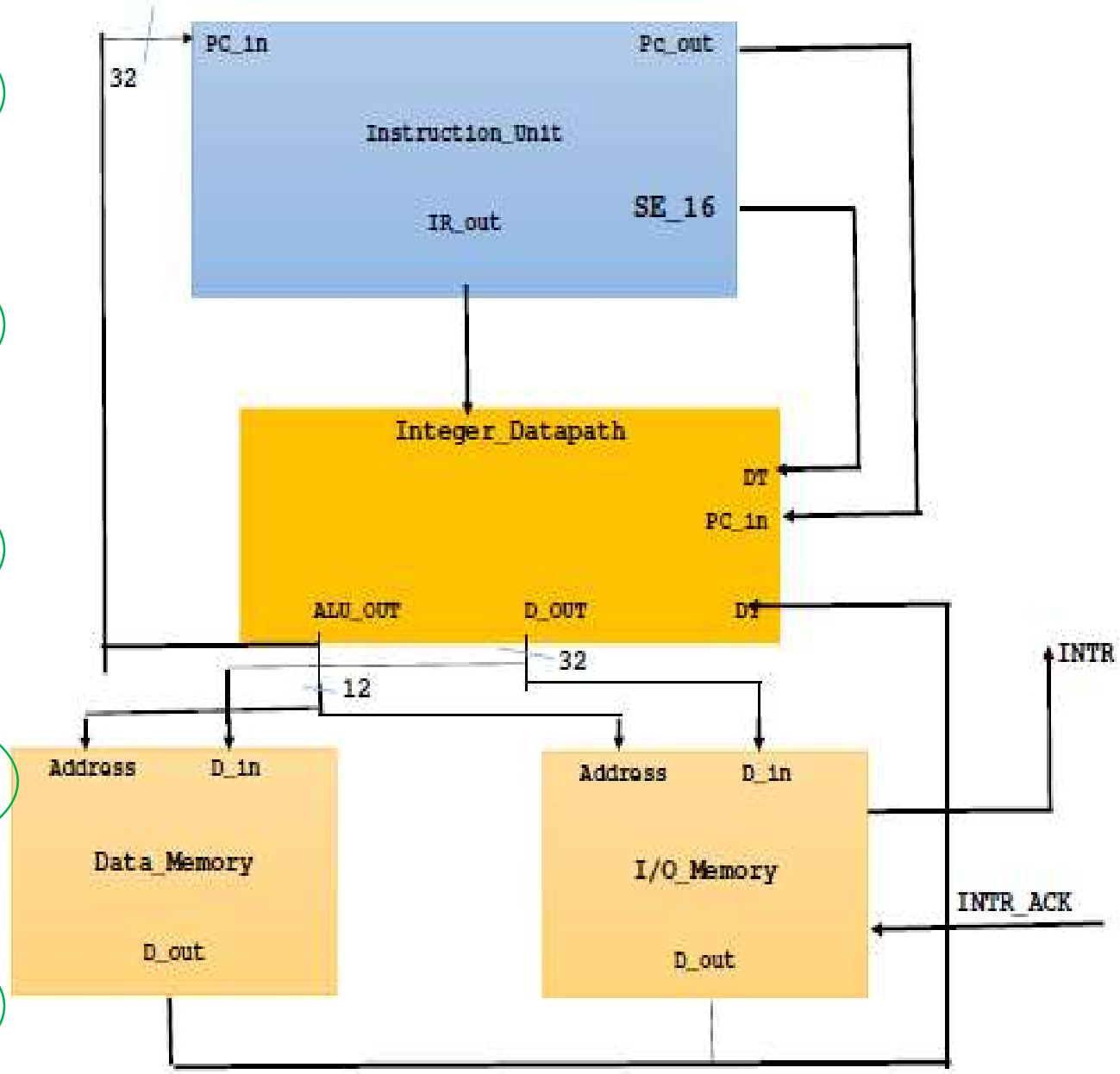
EXECUTION

↓

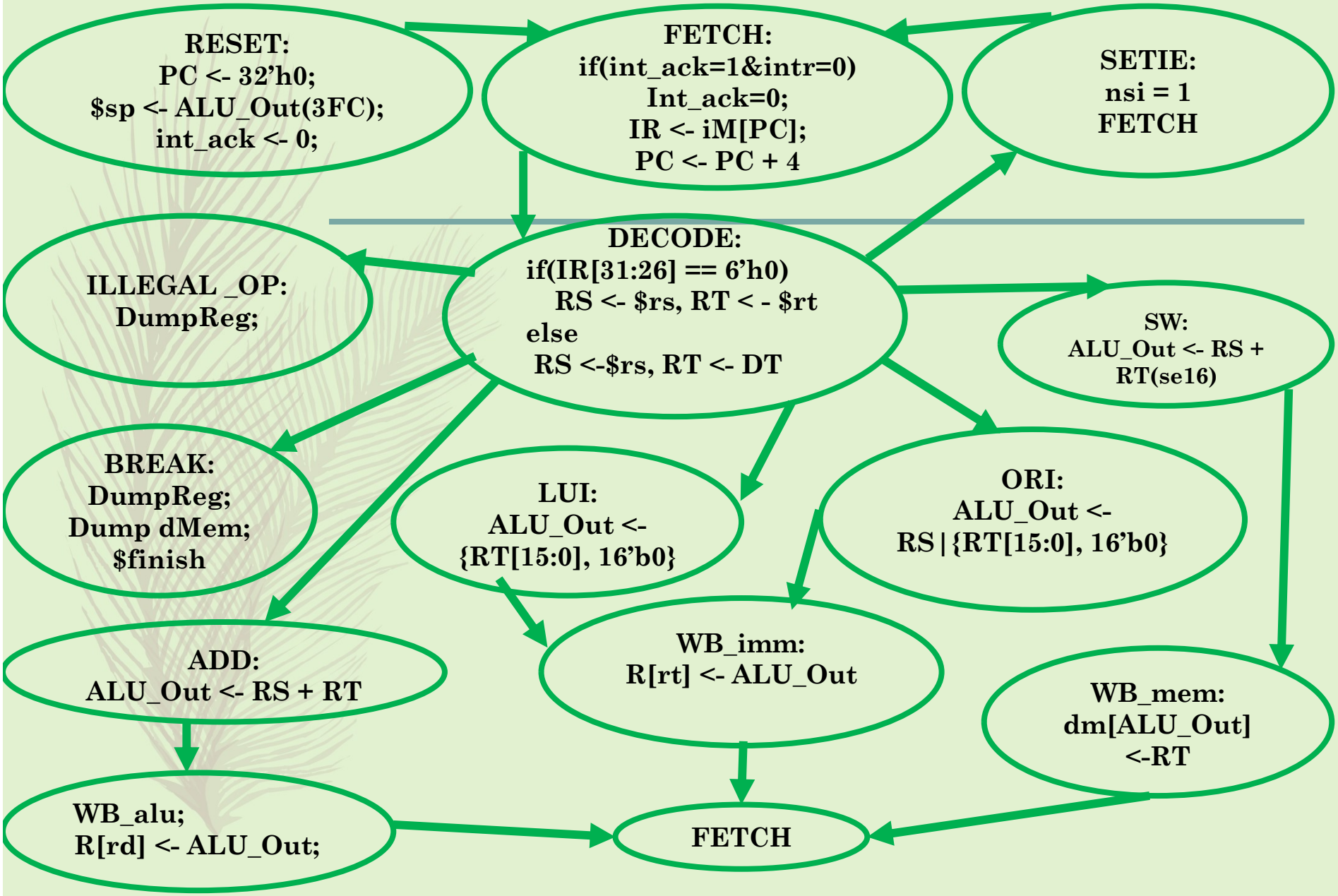
MEMORY READ

↓

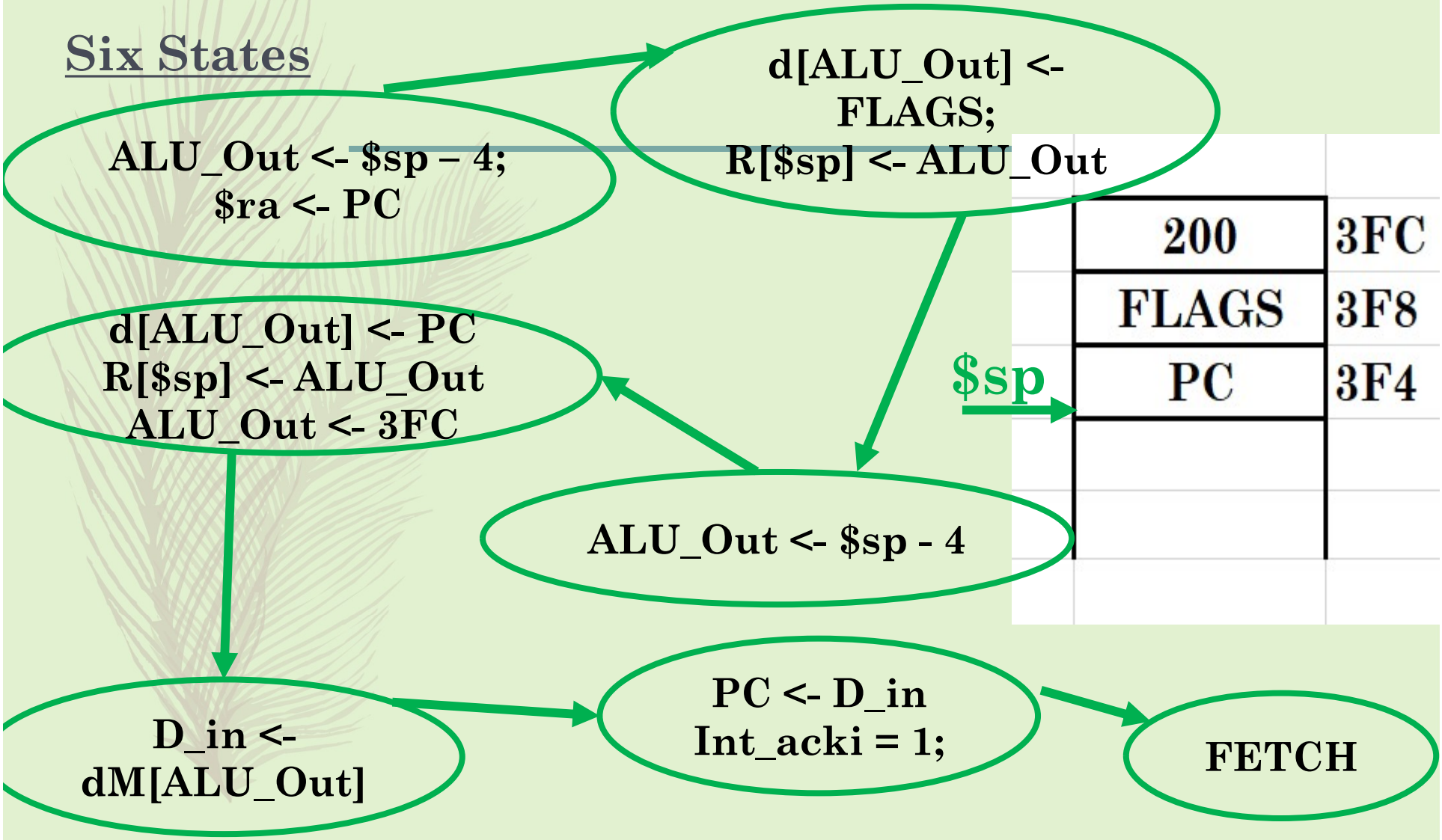
WRITE BACK



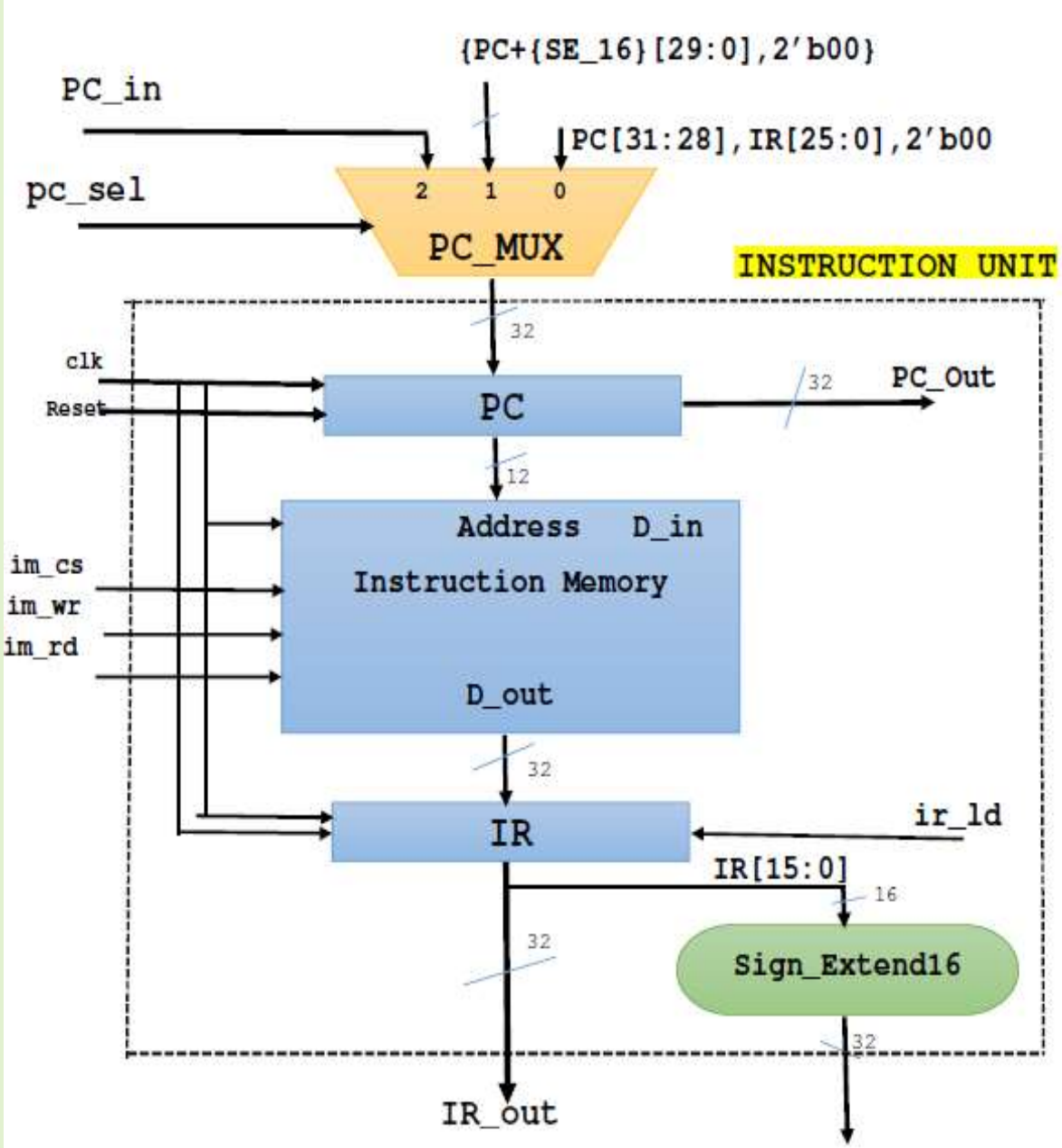
Control Unit Finite State Machine

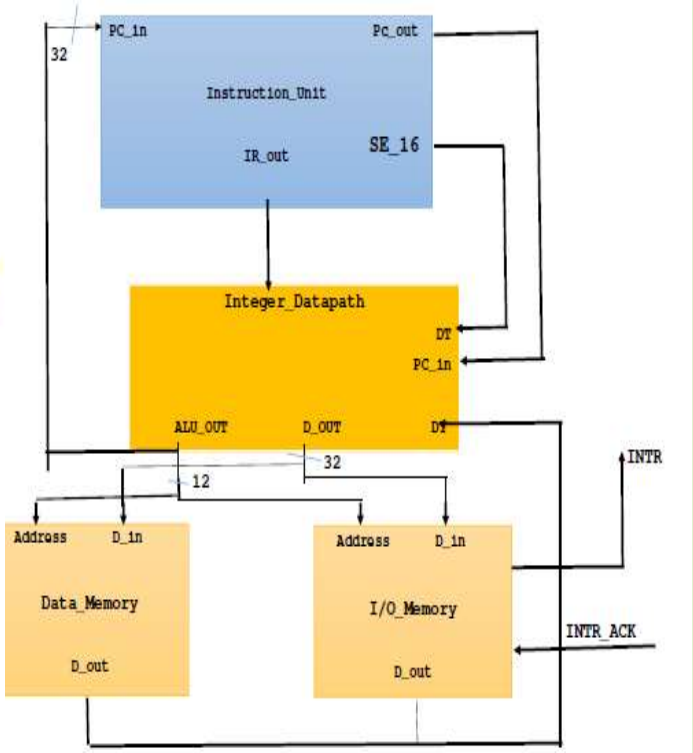


The Flow of Interrupt Service Routine



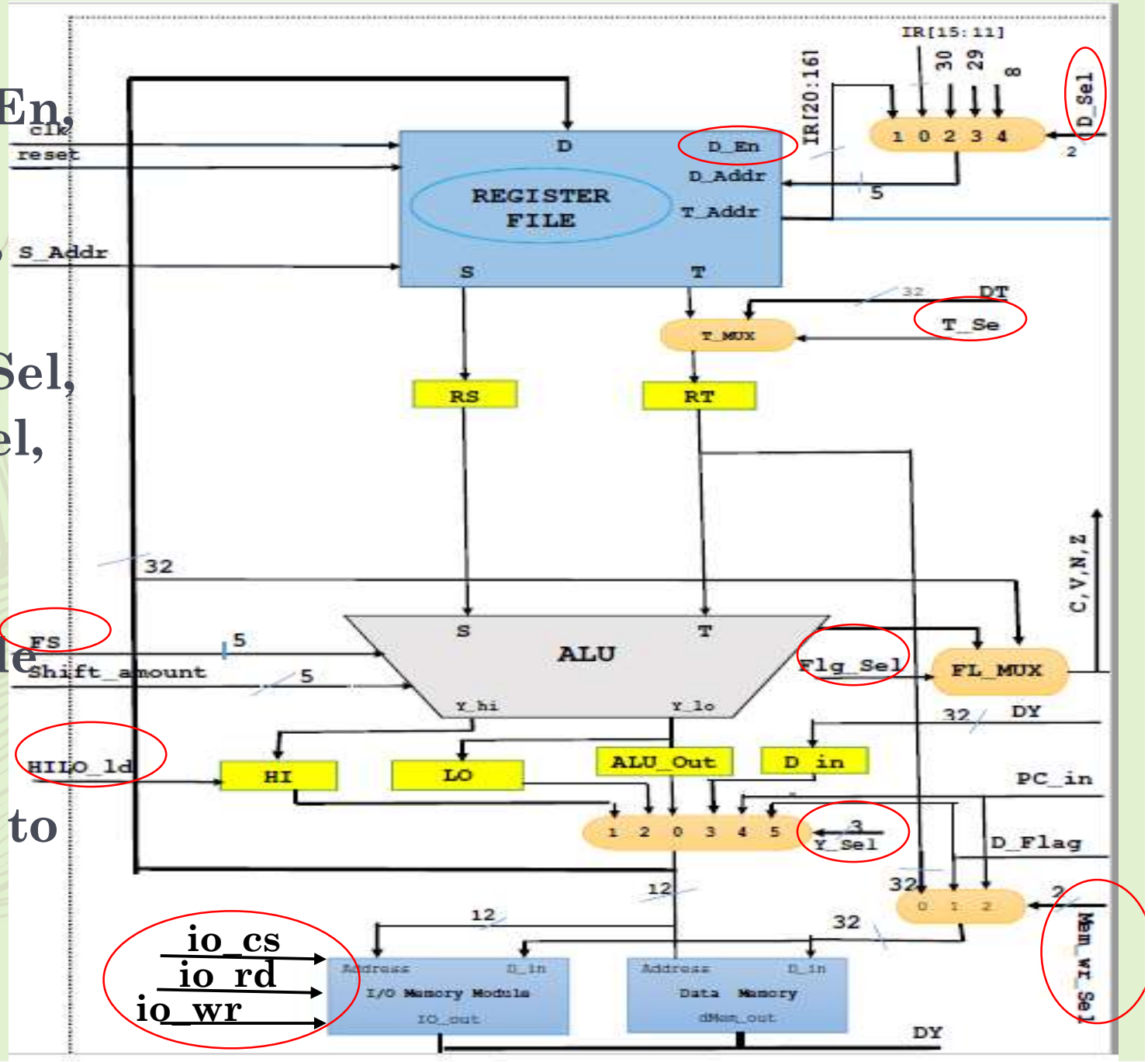
Instruction Unit:





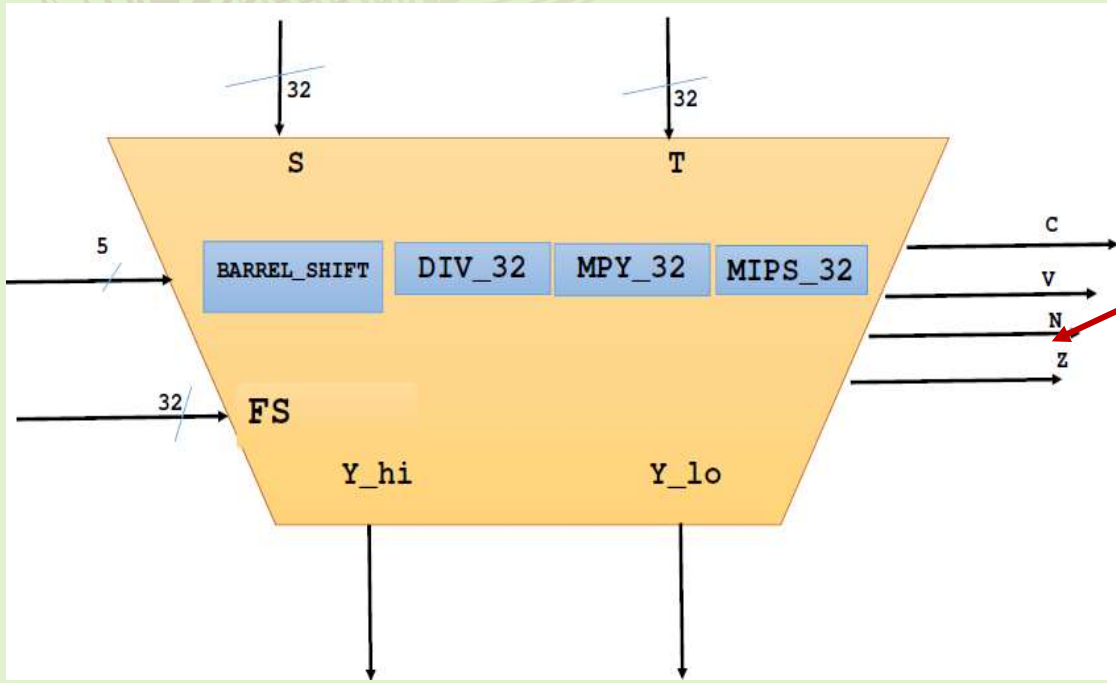
Integer Datapath:

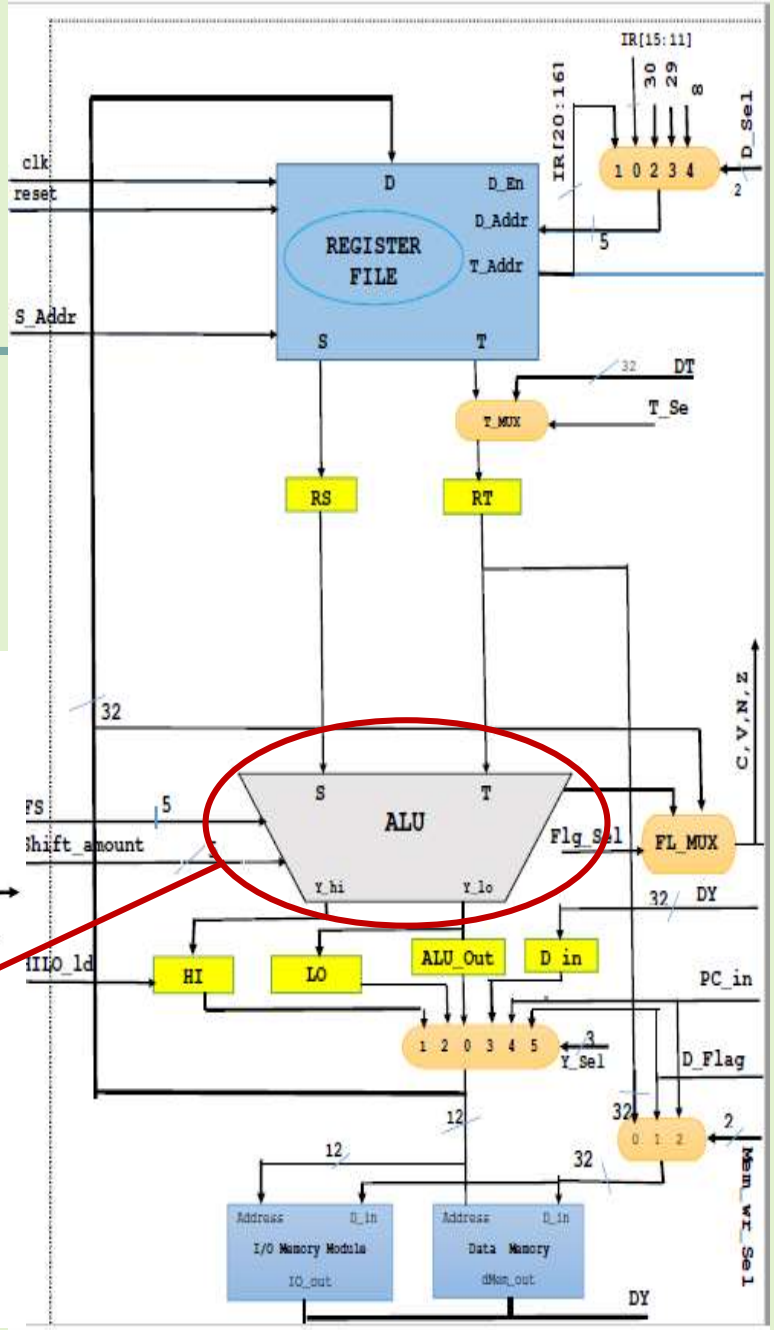
- Control Signals: D_En, D_Sel, Tassels, FS, HILO_ld, Flg_Sel, Y_Sel, Mem_wr_Sel, Memory Selection
- Register File
- RS, RT
- Write Back to Register
- ALU



- Multiply and Divide: HILO_ld
- Mips_32: All other operation
- Barrel Shift: Left, Right, Arithmetic Shift
- Shift Amount: for 1 to 31
- Output from ALU by Y_Sel

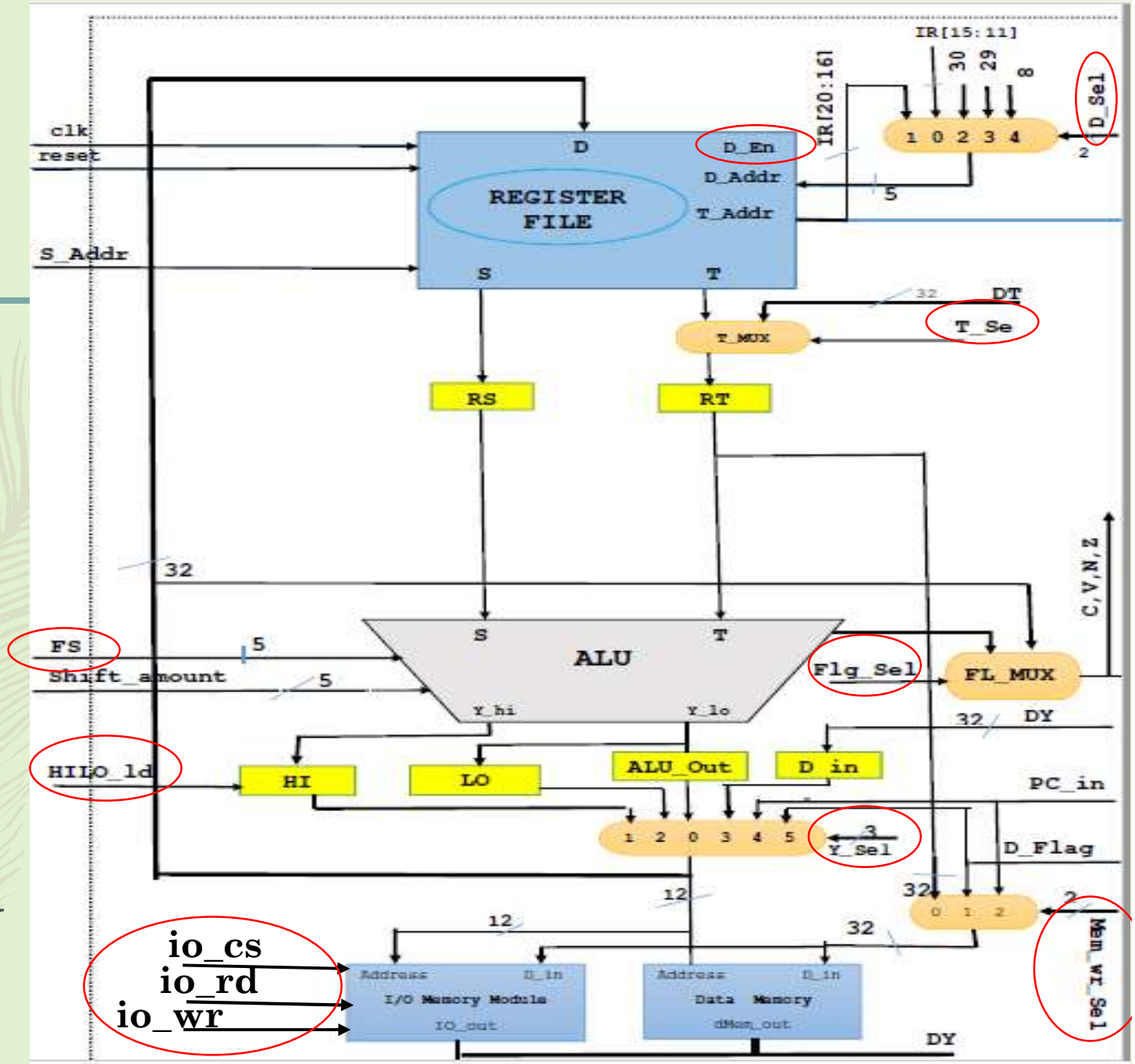
Integer Datapath:





Integer Datapath:

- Memory Address, 12bit
- Memory Data in, 32bit
- D_in: IDP, DY, PC, D_Flag
- Memory Output: DY

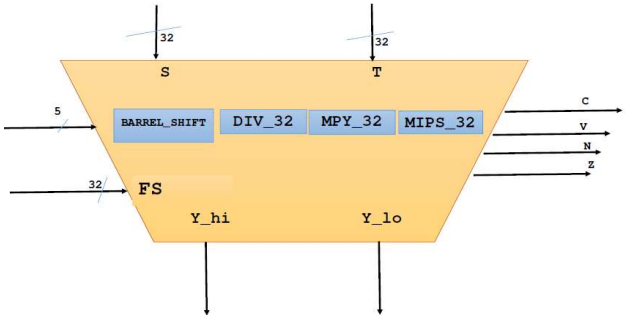


Barrel Shift Left Result Verification:

```
/*****************************************
*BARREL SHIFT LEFT LOGICAL 32
*****************************************/
      {5'h0C, 5'd1}:   {c, y_lo} = {TData[31], {TData[30:0], 1'b0}};        //SLL_1
      {5'h0C, 5'd2}:   {c, y_lo} = {TData[30], {TData[29:0], 2'b0}};        //SLL_2
      {5'h0C, 5'd3}:   {c, y_lo} = {TData[29], {TData[28:0], 3'b0}};        //SLL_3
      {5'h0C, 5'd4}:   {c, y_lo} = {TData[28], {TData[27:0], 4'b0}};        //SLL_4
      {5'h0C, 5'd5}:   {c, y_lo} = {TData[27], {TData[26:0], 5'b0}};        //SLL_5
      {5'h0C, 5'd6}:   {c, y_lo} = {TData[26], {TData[25:0], 6'b0}};        //SLL_6
      {5'h0C, 5'd7}:   {c, y_lo} = {TData[25], {TData[24:0], 7'b0}};        //SLL_7
      {5'h0C, 5'd8}:   {c, y_lo} = {TData[24], {TData[23:0], 8'b0}};        //SLL_8
      {5'h0C, 5'd9}:   {c, y_lo} = {TData[23], {TData[22:0], 9'b0}};        //SLL_9
```

Opcode					Function
0x00	$rs	$rt	$rd	shmt	0x00

ALU Internal Design




```
*****************************************************
C E C B 4 4 0 A_L_U T e s t b e n c h R e s u l t s
-----------------------------------------------------
SHIFT LEFT LOGICAL:
Finished circuit initialization process.
SLL 04: T=800ffff0 Ylo=00ffff00 c=0 v=0 n=0 z
SLL 08: T=800ffff0 Ylo=0ffff000 c=0 v=0 n=0 z
SLL 16: T=800ffff0 Ylo=fff00000 c=1 v=0 n=1 z
SLL 20: T=800ffff0 Ylo=ff000000 c=1 v=0 n=1 z
```

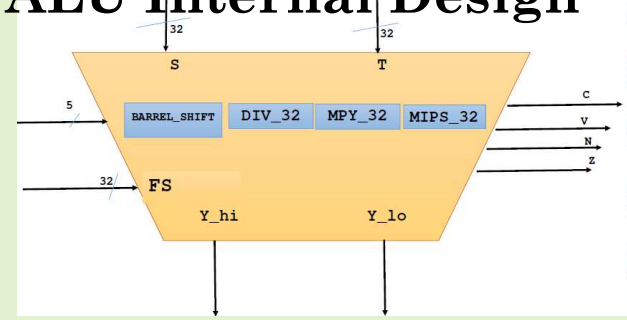
Barrel Shift Right Result Verification:

```
/*****************************************
*BARREL SHIFT RIGHT LOGICAL 32 bit
*****************************************/
    {5'h0D, 5'd1}:   {c, y_lo} = {TData[0],  {1'b0,  TData[31:1]}};      //SRL_1
    {5'h0D, 5'd2}:   {c, y_lo} = {TData[1],  {2'b0,  TData[31:2]}};      //SRL_2
    {5'h0D, 5'd3}:   {c, y_lo} = {TData[2],  {3'b0,  TData[31:3]}};      //SRL_3
    {5'h0D, 5'd4}:   {c, y_lo} = {TData[3],  {4'b0,  TData[31:4]}};      //SRL_4
    {5'h0D, 5'd5}:   {c, y_lo} = {TData[4],  {5'b0,  TData[31:5]}};      //SRL_5
    {5'h0D, 5'd6}:   {c, y_lo} = {TData[5],  {6'b0,  TData[31:6]}};      //SRL_6
    {5'h0D, 5'd7}:   {c, y_lo} = {TData[6],  {7'b0,  TData[31:7]}};      //SRL_7
    {5'h0D, 5'd8}:   {c, y_lo} = {TData[7],  {8'b0,  TData[31:8]}};      //SRL_8
    {5'h0D, 5'd9}:   {c, y_lo} = {TData[8],  {9'b0,  TData[31:9]}};      //SRL_9
    {5'h0D, 5'd10}:  {c, y_lo} = {TData[9],  {10'b0, TData[31:10]}};     //SRL_10
```

Opcode Function

0x00 $rs $rt $rd shmt 0x02

ALU Internal Design



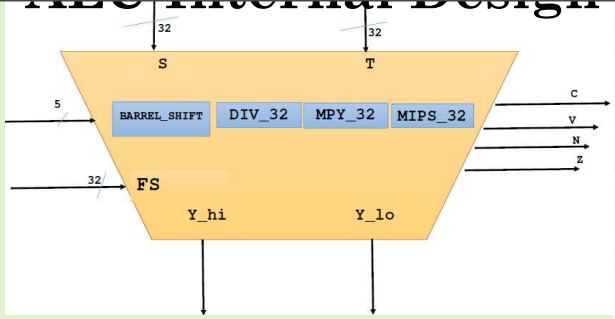

```
SHIFT RIGHT LOGICAL:
SRL 04: T=800ffff0 Ylo=0800ffff c=0 v=0 n=0
SRL 08: T=800ffff0 Ylo=00800fff c=1 v=0 n=0
SRL 16: T=800ffff0 Ylo=0000800f c=1 v=0 n=0
SRL 20: T=800ffff0 Ylo=00000800 c=1 v=0 n=0
```

Barrel Arithmetic Shift Result:

```
/*****************************************
*BARREL SHIFT RIGHT ARITHMATIC 32
*****************************************/
    {5'h0E, 5'd1}:  {c, y_lo} = {TData[0],  { TData[31],     TData[31:1]}};     //SRA_1
    {5'h0E, 5'd2}:  {c, y_lo} = {TData[1],  {{2{TData[31]}}, TData[31:2]}};     //SRA_2
    {5'h0E, 5'd3}:  {c, y_lo} = {TData[2],  {{3{TData[31]}}, TData[31:3]}};     //SRA_3
    {5'h0E, 5'd4}:  {c, y_lo} = {TData[3],  {{4{TData[31]}}, TData[31:4]}};     //SRA_4
    {5'h0E, 5'd5}:  {c, y_lo} = {TData[4],  {{5{TData[31]}}, TData[31:5]}};     //SRA_5
    {5'h0E, 5'd6}:  {c, y_lo} = {TData[5],  {{6{TData[31]}}, TData[31:6]}};     //SRA_6
    {5'h0E, 5'd7}:  {c, y_lo} = {TData[6],  {{7{TData[31]}}, TData[31:7]}};     //SRA_7
    {5'h0E, 5'd8}:  {c, y_lo} = {TData[7],  {{8{TData[31]}}, TData[31:8]}};     //SRA_8
    {5'h0E, 5'd9}:  {c, y_lo} = {TData[8],  {{9{TData[31]}}, TData[31:9]}};     //SRA_9
    {5'h0E, 5'd10}: {c, y_lo} = {TData[9],  {{10{TData[31]}}, TData[31:10]}};   //SRA_10
```

Opcode					Function
0x00	$rs	$rt	$rd	shmt	0x03

ALU Internal Design




```
SHIFT RIGHT ARITHMATIC:
SRA 04: T=800ffff0 Ylo=f800ffff c=0 v=0 n=1
SRA 08: T=800ffff0 Ylo=ff800fff c=1 v=0 n=1
SRA 16: T=800ffff0 Ylo=ffff800f c=1 v=0 n=1
SRA 20: T=800ffff0 Ylo=fffff800 c=1 v=0 n=1
```

Enhanced Instructions Set

- INPUT
- OUTPUT
- RETI

INPUT instruction

FETCH:
if(int_ack=1&intr=0)
Int_ack=0;
IR <- iM[PC];
PC <- PC + 4

DECODE:
if(IR[31:26] == 6'h0)
RS <- $rs, RT < - $rt
else
RS <-$rs, RT <- DT

SW:
ALU_Out <- RS + RT(se16)

I/O_rd:
D_in <- M[ALU_Out]

WB_imm:
R[rt] <- ALU_Out

E-Type		Opcode	$rs	$rt	se_16
	INPUT	0x1C	5-bit	5-bit	16-bit immediat

OUTPUT instruction

FETCH:
if(int_ack=1&intr=0)
Int_ack=0;
IR <- iM[PC];
PC <- PC + 4

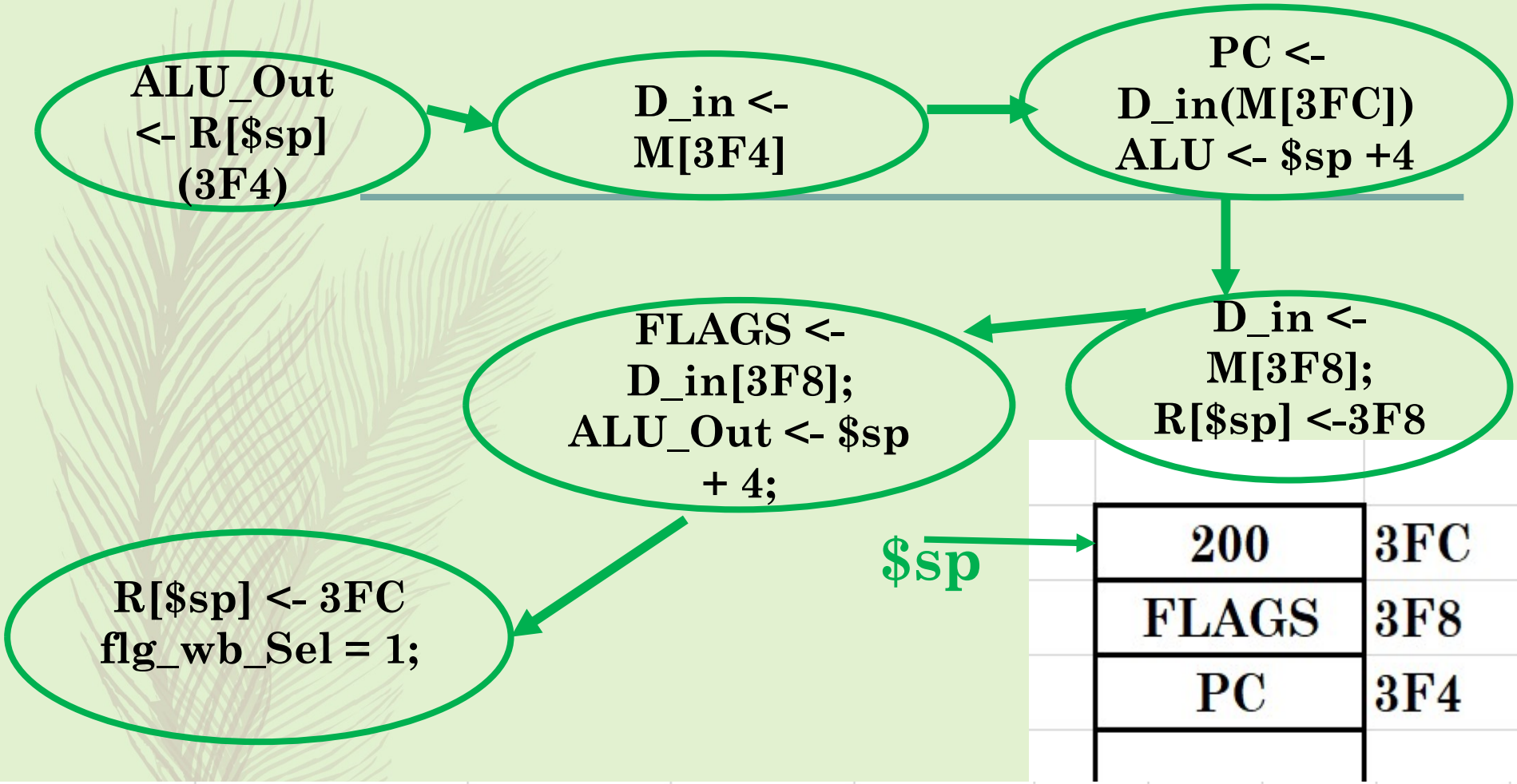
DECODE:
if(IR[31:26] == 6'h0)
RS <- $rs, RT < - $rt
else
RS <-$rs, RT <- DT

SW:
ALU_Out <- RS + RT(se16)

WB_I/O:
ioM[ALU_Out] <- RT

E-Type		Opcode	$rs	$rt	se_16
	OUTPUT	0x1D	5-bit	5-bit	16-bit immediat

RETI instruction



E-Type		Opcode	$rs	$rt	se_16
	RETI	0x1E	$sp	$r0	16-bit immediat

Verifications Result:

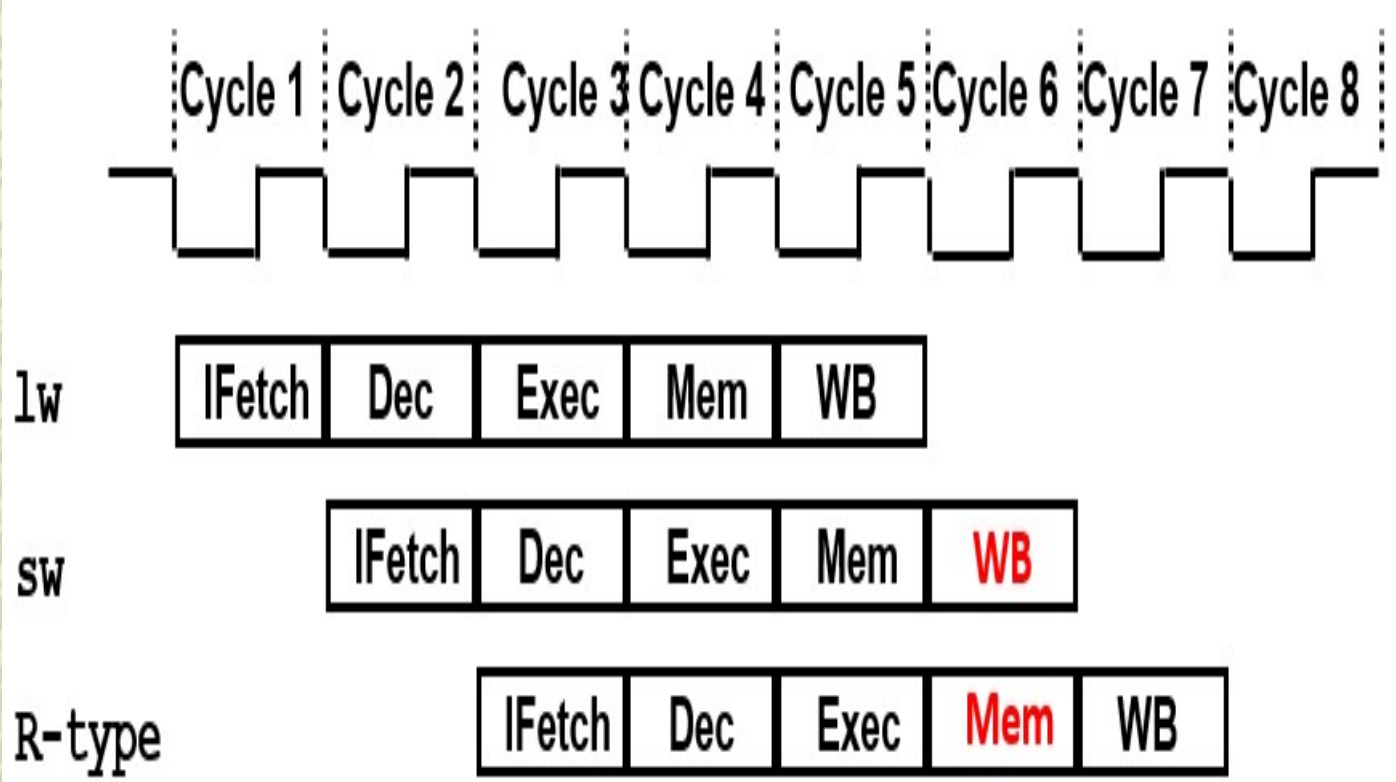
```
3c 10 10 01   // isr:        lui     $16, 0x1001
36 10 00 c0   //             ori     $16, 0x00C0
3c 11 80 00   //             lui     $17, 0x8000
36 31 ff ff   //             ori     $17, 0xFFFF
20 12 00 10   //             addi    $18, $0, 0x10

76 11 00 00   // out_IO:     output  $17, 0($16)
00 11 88 83   //             sra     $17, $17, 2
22 10 00 04   //             addi    $16, $16, 4
22 52 ff ff   //             addi    $18, $18, -1
16 40 ff fb   //             bne     $18, $00, out_IO

3c 10 10 01   //             lui     $16, 0x1001
36 10 00 c0   //             ori     $16, 0x00C0
72 13 00 00   //             input   $19,  0($16)
72 14 00 04   //             input   $20,  4($16)
72 15 00 08   //             input   $21,  8($16)
72 16 00 0c   //             input   $22, 12($16)
72 17 00 10   //             input   $23, 16($16)
72 18 00 14   //             input   $24, 20($16)
7B A0 00 00   //             reti
[sp] as saved PC)
```

```
S_addr[0c]=fffffefb || T_addr[1c]=xxxxxxxx
S_addr[0d]=fffffefa || T_addr[1d]=000003fc
S_addr[0e]=fffffef9 || T_addr[1e]=xxxxxxxx
S_addr[0f]=fffffef8 || T_addr[1f]=00000044
FINISH READING REGISTER FILE AFTER BREAK!!!
M[3F4:3F7]=00000044
DISPLAY I/O module from 0C0
io_addr[000000b0]=xxxxxxxx
io_addr[000000b4]=xxxxxxxx
io_addr[000000b8]=xxxxxxxx
io_addr[000000bc]=xxxxxxxx
io_addr[000000c0]=8000ffff
io_addr[000000c4]=e0003fff
io_addr[000000c8]=f8000fff
io_addr[000000cc]=fe0003ff
io_addr[000000d0]=ff8000ff
io_addr[000000d4]=ffe0003f
io_addr[000000d8]=fff8000f
io_addr[000000dc]=fffe0003
io_addr[000000e0]=ffff8000
io_addr[000000e4]=ffffe000
io_addr[000000e8]=fffff800
io_addr[000000ec]=ffffffe00
io_addr[000000f0]=fffffff80
io_addr[000000f4]=ffffffffe0
io_addr[000000f8]=ffffffff8
io_addr[000000fc]=fffffffffe
io_addr[00000100]=xxxxxxxx
```

```
T_addr[10]=100100c0
T_addr[11]=ffffffff
T_addr[12]=00000000
T_addr[13]=8000ffff
T_addr[14]=e0003fff
T_addr[15]=f8000fff
T_addr[16]=fe0003ff
T_addr[17]=ff8000ff
T_addr[18]=ffe0003f
```

Output of I/O Memory

Current Pipeline Enhancement

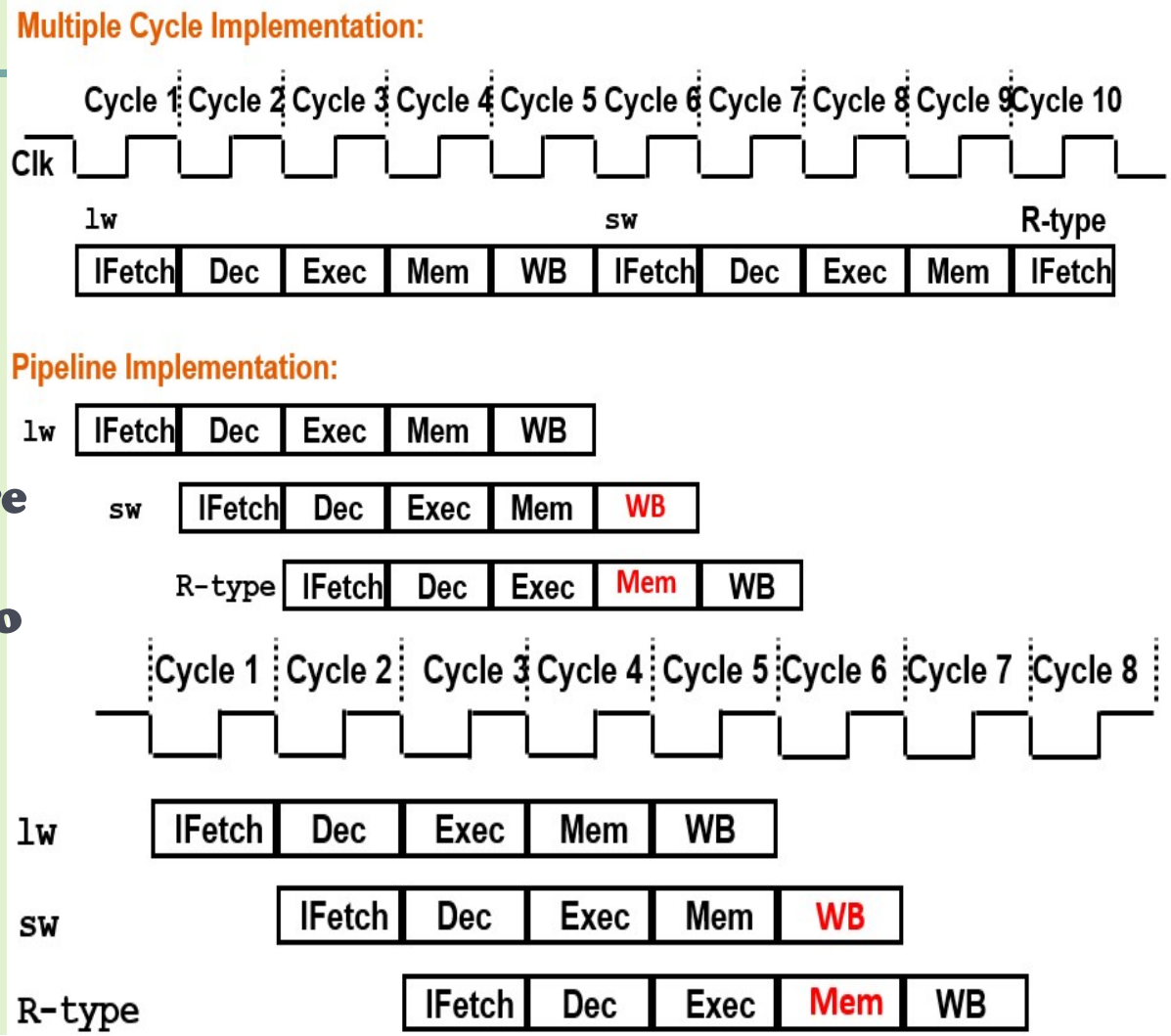


Pipeline Concept and Issues:

- Divide one instruction to execute in many stages depend on each architecture. Each stage is one clock cycle, and the instruction is fetched every clock cycle from instruction unit.
- MIPS is a 5 stages pipeline with two separate memories.

Hazards of Pipeline:

1. **Structural Hazard:** two instruction access memory at same time.
2. **Control Hazard:** Program Counter (PC)is generate before knowing the next instruction address to be fetched.
3. **Data Hazard:** Data dependency to the register result of the previous instruction.

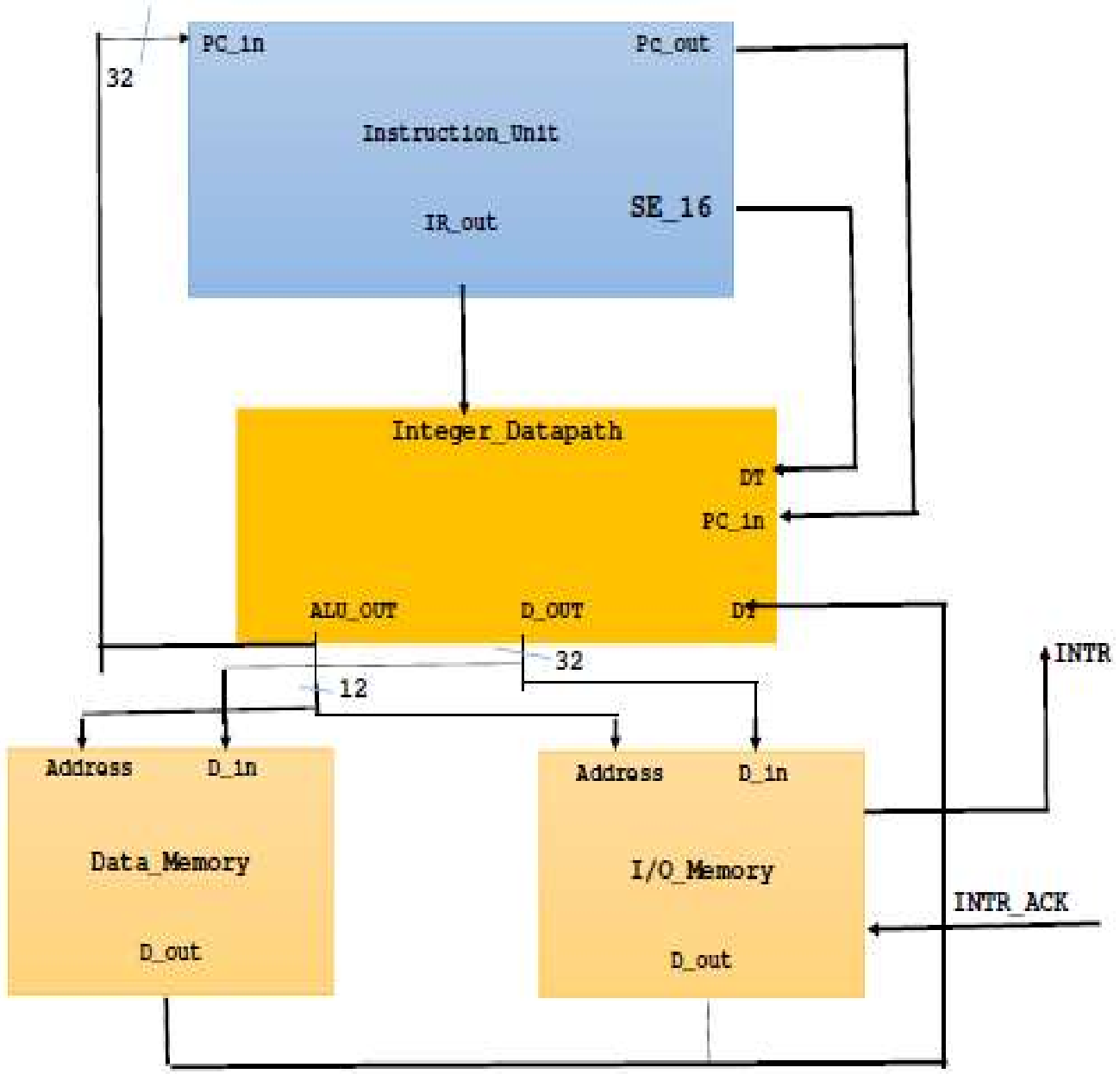


How to Fix Structural Hazard:

TWO

MEMORY

MODULES



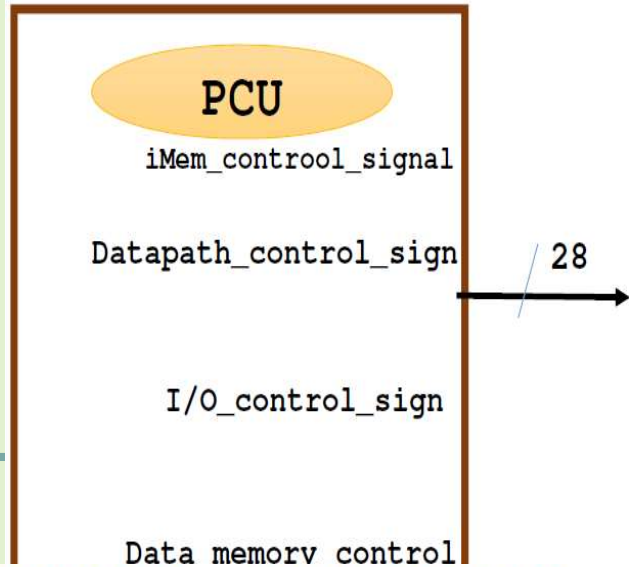
Modifications

Control Unit

ID_EX_reg (28)

EX_MEM_reg (12)

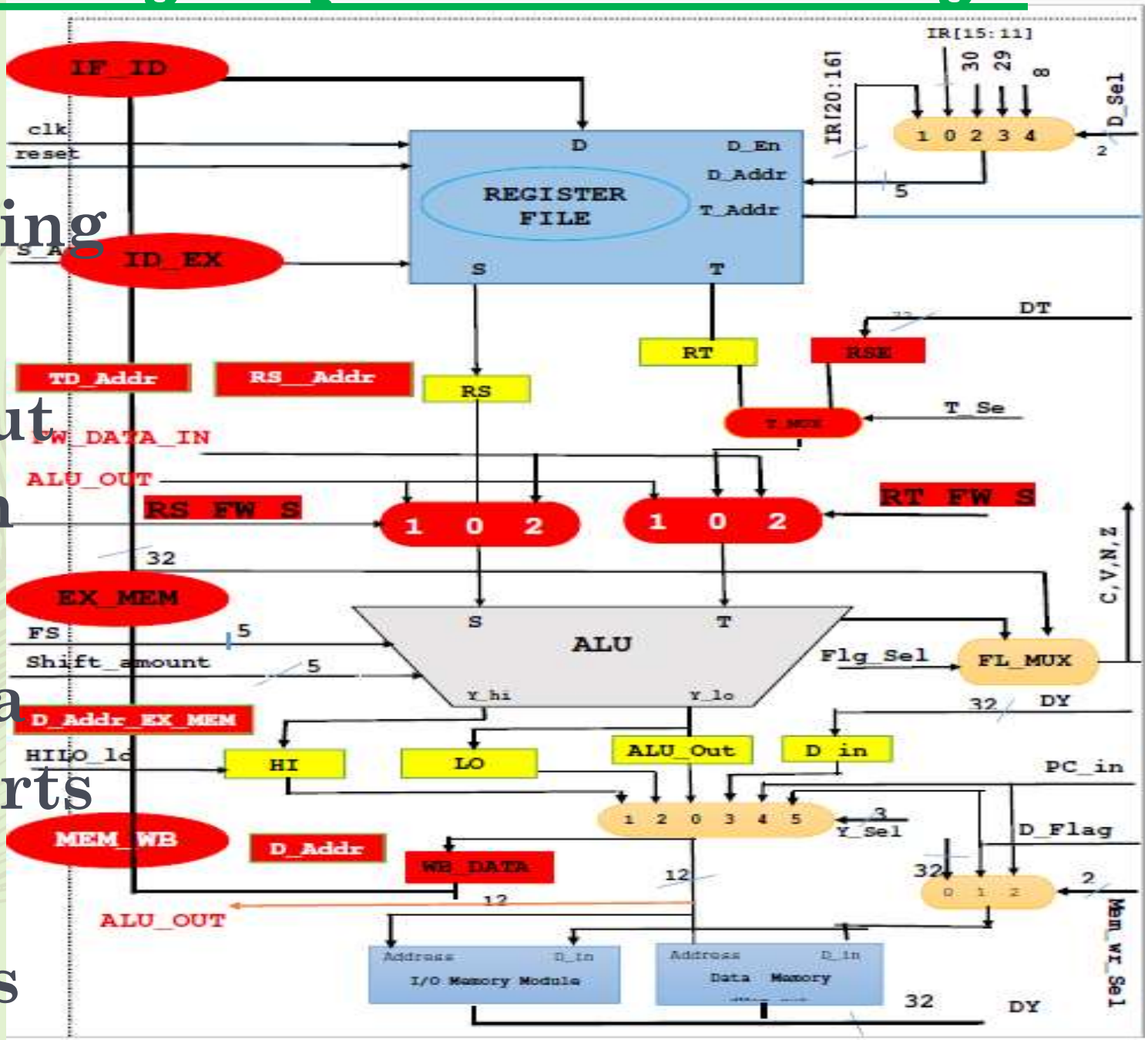
MEM_WB_reg (6)




```
{S_Sel, T_Sel, D_Sel, HILO_ld} = 5'b0_0_00_0;  //CNT[27:23], ID_EX_control
FS                             = 5'h0C;        //CNT[22:18], ID_EX_control
Y_Sel                          = 3'b000;       //CNT[17:15], EX_MEM_control
{Mem_wr_Sel, flg_wb_Sel}       = 3'b00_0;      //CNT[14:12], EX_MEM_control
{dm_cs, dm_rd, dm_wr}          = 3'b0_0_0;     //CNT[11:9],  EX_MEM_control
{io_cs, io_rd, io_wr}          = 3'b0_0_0;     //CNT[8:6],   EX_MEM_control
{D_En, D_Sel, Y_Sel}           = 6'b1_00_000;  //CNT[5:0],   MEM_WB_control

assign  CNT = {{S_Sel, T_Sel, D_Sel, HILO_ld}, FS,
               Y_Sel, {Mem_wr_Sel, flg_wb_Sel},
               {dm_cs, dm_rd, dm_wr}, {io_cs, io_rd, io_wr},
               {D_En, D_Sel, Y_Sel}};
```

FIVE Stages Pipeline Hardware Design:

- Forwarding Unit
- Two Input Selection ports
- Two Data Input Ports
- Six more Registers



Pipeline Verifications Result:

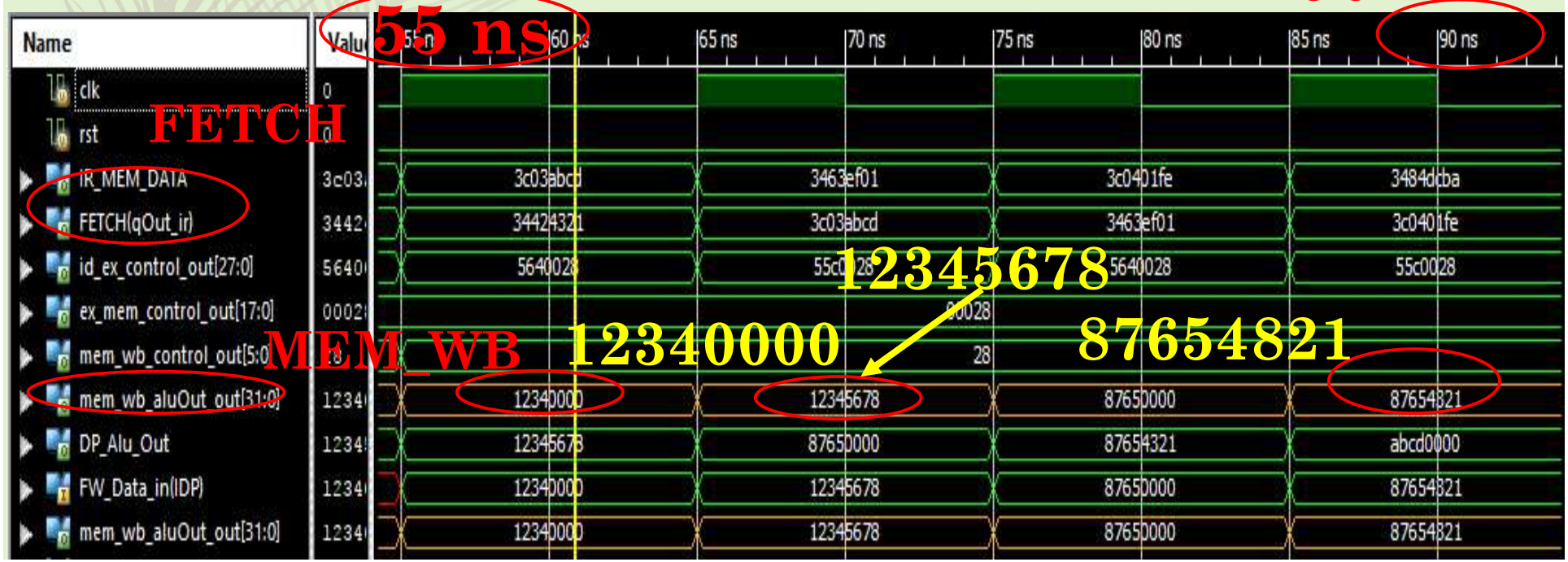
File Edit Format View Help

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

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

3c 07 10 01   // lui  $07, 0x1001
34 e7 03 F0   // ori  $07, 0x03F0        R07   <-- 0x100103F
ac ef 00 00   // sw   $15, 0($07)       M[R07] <-- R15
00 00 00 0d   // break
```

FFFFFF00

100103F0

20 ns
55 ns
3C011234
3C03ABCD
FETCH
MEM_WB
12340000
Name
clk
rst
IR_MEM_DATA
FETCH(qOut_ir)
id_ex_control_out[27:0]
ex_mem_control_out[17:0]
mem_wb_control_out[5:0]
mem_wb_aluOut_out[31:0]
DP_Alu_Out
FW_Data_in(IDP)
mem_wb_aluOut_out[31:0]
3c011234
34215678
3c028765
34424321
3c03abcd
00000000
3c011234
34215678
3c028765
34424321
0000000
0300020
5640028
55c0028
5640028
00000
00020
00028
00
28
00000000
XXXXXXXX
12340000
00000000
XXXXXXXX
12340000
12345678
00000000
XXXXXXXX
12340000
00000000
XXXXXXXX
12340000


55 ns
90 ns
FETCH
MEM_WB
12345678
12340000
87654821
Name
clk
rst
IR_MEM_DATA
FETCH(qOut_ir)
id_ex_control_out[27:0]
ex_mem_control_out[17:0]
mem_wb_control_out[5:0]
mem_wb_aluOut_out[31:0]
DP_Alu_Out
FW_Data_in(IDP)
mem_wb_aluOut_out[31:0]
3c03abcd
3463ef01
3c0401fe
3484dcba
34424321
3c03abcd
3463ef01
3c0401fe
5640028
5640028
55c0028
00028
28
12340000
12345678
87650000
87654821
12345678
87650000
87654321
abcd0000
12340000
12345678
87650000
87654821
12340000
12345678
87650000
87654821

ISim (P.20131013) - [Pipeline_2_Control_Unit.v]

File Edit View Simulation Window Layout Help

Instances and Processes

Instance and Process Name

- Pepeline_Toplevel_Testbe
  - uut
    - PCU_2
    - instrct_unit
    - int_dp
      - reg_file
      - rSE_rg
      - rS_rg
      - rT_rg
      - ALU_32bit
      - alu_out_rg
      - d_in_rg
      - HI_ld_reg
      - LO_ld_reg
      - Cont_44_0
      - Cont_45_1
      - Cont_49_2
      - Cont_53_3
      - Cont_68_4
      - Cont_78_5
      - Cont_96_6
      - Cont_100_7
      - Cont_104_8
      - Cont_155_9
      - Cont_161_0
    - data_mem
    - io_mem
    - rs_rt_addr
    - rt_rd_addr
    - id_ex_control

Objects

Simulation Objects for reg_file

Object Name	Value
S_Addr[4:0]	xxxxx
T_Addr[4:0]	xxxxx
D_Addr[4:0]	xxxxx
S[31:0]	xxxxxxxxxxxx
T[31:0]	xxxxxxxxxxxx
reg_mem[0:31...	00000000 123
[0,31:0]	00000000
[1,31:0]	12345678
[2,31:0]	87654321
[3,31:0]	abcdef01
[4,31:0]	01fedcba
[5,31:0]	5a5a5a5a
[6,31:0]	ffffffff
[7,31:0]	100103f0
[8,31:0]	xxxxxxxx
[9,31:0]	xxxxxxxx
[10,31:0]	xxxxxxxx
[11,31:0]	xxxxxxxx
[12,31:0]	xxxxxxxx
[13,31:0]	xxxxxxxx
[14,31:0]	xxxxxxxx
[15,31:0]	xxxxxxxx
[31:0]	xxxxxxxx
[31:0]	xxxxxxxx
[31:0]	xxxxxxxx
[31:0]	xxxxxxxx
[20,31:0]	xxxxxxxx
[21,31:0]	xxxxxxxx
[22,31:0]	xxxxxxxx

```
361        #1 {nsi, nsc, nsv, nsn, nsz}    = 5'b0;
362        $display("IR[31:0]=%h ", IR[31:0]);
363        $display("SLTU");
364     end  //end SLTU
365
366     BREAK:
367        begin
368        $display(" ");
369        {S_Sel, T_Sel, D_Sel, HILO_ld} = 5'b0_0_00_0; //CN
370        FS                             = 5'h00;       //CN
371        Y_Sel                          = 3'b000;      //CN
372        {Mem_wr_Sel, flg_wb_Sel}       = 3'b00_0;     //CN
373        {dm_cs, dm_rd, dm_wr}          = 3'b0_0_0;    //CN
374        {io_cs, io_rd, io_wr}          = 3'b0_0_0;    //CN
375        {D_En, D_Sel, Y_Sel}           = 6'b0_00_000; //CN
376        int_ack = 1'b0;
377        #1 {nsi, nsc, nsv, nsn, nsz}    = 5'b0;
378        $display("IR[31:0]=%h ", IR[31:0]);
379        $display("BREAK");
380        $display ("REGISTER' AFTER BREAK");
381
382        Dump_Registers; //task to output MIPS RegFile
383        $finish;
384        end  //end BREAK
385     SETIE:
386        begin
387           {S_Sel, T_Sel, D_Sel, HILO_ld} = 5'b0_0_00_0; /
388           FS                             = 5'h00;       /
389           Y_Sel                          = 3'b000;      /
390           {Mem_wr_Sel, flg_wb_Sel}       = 3'b00_0;     /
391           {dm_cs, dm_rd, dm_wr}          = 3'b0_0_0;    /
392           {io_cs, io_rd, io_wr}          = 3'b0_0_0;    /
393           {D_En, D_Sel, Y_Sel}           = 6'b0_00_000; /
```

Default.wcfg | Pipeline_2_Control_Unit.v

Console

```
time= 441.0 ps, S_addr[0f]=xxxxxxxx || T_addr[1f]=xxxxxxxx
FINISH READING REGISTER FILE AFTER BREAK!!!.
Stopped at time : 441 ns : File "C:/Users/chealykim/Desktop/Pipeline/Pipeline/Pipeline_2_Control_Unit.v" Line 383
ISim>
```

441 ns

Instance and Process Name

- MIPS_Processor_Topleve
  - uut
    - instrct_unit
    - mcu
    - intgr_p2
      - rs_rg
      - rt_rg
      - alu_out_rg
      - d_in_rg
      - reg_file
      - ALU_32bit
      - HI_ld_reg
      - LO_ld_reg
      - Cont_41_0
      - Cont_42_1
      - Cont_45_2
      - Cont_46_3
      - Cont_50_4
      - Cont_116_5
      - Cont_123_6
    - data_mem
    - io_mem
    - Cont_39_0
    - Cont_73_1
    - Cont_80_2
    - Cont_81_3
    - Cont_84_4
  - Always_30_0
  - Initial_32_
- glb

Object Name	Value
clk	0
rst	0
D_En	0
D[31:0]	00000000
S_Addr[4:0]	00000
T_Addr[4:0]	00000
D_Addr[4:0]	00000
S[31:0]	00000000
T[31:0]	00000000
reg_mem[0:31...	00000000
[0,31:0]	00000000
[1,31:0]	12345678
[2,31:0]	87654321
[3,31:0]	abcdef01
[4,31:0]	01fedcba
[5,31:0]	5a5a5a5a
[6,31:0]	ffffffff
[7,31:0]	100103f0
[8,31:0]	ffffffff
[9,31:0]	ffffffe
[10,31:0]	ffffffd
[11,31:0]	ffffffc
[12,31:0]	ffffffb
[13,31:0]	ffffffa
[14,31:0]	ffffff9
[15,31:0]	ffffff8
[16,31:0]	xxxxxxxx
[17,31:0]	xxxxxxxx
[18,31:0]	xxxxxxxx

```
1355        end //end WB_io
1356
1357     BREAK:
1358        begin
1359           $display ("BREAK INSTRUCTION FETCHED %t", $time);
1360           //control word assignment for  "deasserting" everything
1361           @(negedge clk)
1362           {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b10_0_0_0;
1363           {im_cs, im_rd, im_wr} = 3'b0_0_0;
1364           {D_En, D_Sel, S_Sel, T_Sel, HILO_ld, Y_Sel, Mem_wr_Sel, f
1365           {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
1366           FS = 5'h0;
1367           SA = 5'd0;
1368           int_ack = 1'b0;
1369           $display ("REGISTER' AFTER BREAK");
1370           Dump_Registers; //task to output MIPS RegFile
1371           $display ("FINISH READING REGISTER FILE AFTER BREAK!!!");
1372           $finish;
1373        end //end BREAK
1374
1375     ILLEGAL_OP:
1376        begin
1377           $display("ILLEGAL OPCODE FECTCHED %t", $time);
1378           //control word assignments for "deasserting" everything
1379           @(negedge clk)
1380           {pc_sel, pc_ld, pc_inc, ir_ld} = 5'b10_0_0_0;
1381           {im_cs, im_rd, im_wr} = 3'b0_0_0;
1382           {D_En, D_Sel, S_Sel, T_Sel, HILO_ld, Y_Sel, Mem_wr_Sel, f
1383           {dm_cs, dm_rd, dm_wr} = 3'b0_0_0;
1384           FS = 5'h0;
```

Default.wcfg | MCU.v*

1231.0 ns

Console

```
time=1231.0 ps, S_addr[0f]=ffffffef8  || T_addr[1f]=xxxxxxxx
FINISH READING REGISTER FILE AFTER BREAK!!!
Stopped at time : 1231 ns : File "C:/Users/chealykim/Desktop/440_Final_Project/MIPS_Processor/MIPS_Processor/MCU.v" Line 1391
ISim>
```

Pipeline Verifications Result:

File Edit Format View Help

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

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

3c 07 10 01    // lui   $07, 0x1001
34 e7 03 F0    // ori   $07, 0x03F0        R07   <-- 0x100103F
ac ef 00 00    // sw    $15, 0($07)       M[R07] <-- R15
00 00 00 0d    // break
```

$07

FFFFFF00

$07

100103F0

Pipeline Simulation

isim.log - Notepad

File Edit Format View Help

```
IR[31:0]=0000000d
BREAK
REGISTER' AFTER BREAK
time= 291.0 ps, S_addr[00]=00000000  || T_addr[10]=xxxxxxxx
time= 301.0 ps, S_addr[01]=12345678  || T_addr[11]=xxxxxxxx
time= 311.0 ps, S_addr[02]=87654321  || T_addr[12]=xxxxxxxx
time= 321.0 ps, S_addr[03]=abcdef01  || T_addr[13]=xxxxxxxx
time= 331.0 ps, S_addr[04]=01fedcba  || T_addr[14]=xxxxxxxx
time= 341.0 ps, S_addr[05]=5a5a5a5a  || T_addr[15]=xxxxxxxx
time= 351.0 ps, S_addr[06]=ffffffff  || T_addr[16]=xxxxxxxx
time= 361.0 ps, S_addr[07]=100103f0  || T_addr[17]=xxxxxxxx
time= 371.0 ps, S_addr[08]=xxxxxxxx  || T_addr[18]=xxxxxxxx
time= 381.0 ps, S_addr[09]=xxxxxxxx  || T_addr[19]=xxxxxxxx
time= 391.0 ps, S_addr[0a]=xxxxxxxx  || T_addr[1a]=xxxxxxxx
time= 401.0 ps, S_addr[0b]=xxxxxxxx  || T_addr[1b]=xxxxxxxx
time= 411.0 ps, S_addr[0c]=xxxxxxxx  || T_addr[1c]=xxxxxxxx
time= 421.0 ps, S_addr[0d]=xxxxxxxx  || T_addr[1d]=xxxxxxxx
time= 431.0 ps, S_addr[0e]=xxxxxxxx  || T_addr[1e]=xxxxxxxx
time= 441.0 ps, S_addr[0f]=xxxxxxxx  || T_addr[1f]=xxxxxxxx
FINISH READING REGISTER FILE AFTER BREAK!!!.
Stopped at time : 441 ns : File
```

Multicycle Simulation

isim.log - Notepad

File Edit Format View Help

```
BREAK INSTRUCTION FETCHED 1065.0 ps
REGISTER' AFTER BREAK
time=1081.0 ps, S_addr[00]=00000000  || T_addr[10]=xxxxxxxx
time=1091.0 ps, S_addr[01]=12345678  || T_addr[11]=xxxxxxxx
time=1101.0 ps, S_addr[02]=87654321  || T_addr[12]=xxxxxxxx
time=1111.0 ps, S_addr[03]=abcdef01  || T_addr[13]=xxxxxxxx
time=1121.0 ps, S_addr[04]=01fedcba  || T_addr[14]=xxxxxxxx
time=1131.0 ps, S_addr[05]=5a5a5a5a  || T_addr[15]=xxxxxxxx
time=1141.0 ps, S_addr[06]=ffffffff  || T_addr[16]=xxxxxxxx
time=1151.0 ps, S_addr[07]=100103f0  || T_addr[17]=xxxxxxxx
time=1161.0 ps, S_addr[08]=ffffffff  || T_addr[18]=xxxxxxxx
time=1171.0 ps, S_addr[09]=ffffffef  || T_addr[19]=xxxxxxxx
time=1181.0 ps, S_addr[0a]=ffffffed  || T_addr[1a]=xxxxxxxx
time=1191.0 ps, S_addr[0b]=ffffffec  || T_addr[1b]=xxxxxxxx
time=1201.0 ps, S_addr[0c]=ffffffeb  || T_addr[1c]=xxxxxxxx
time=1211.0 ps, S_addr[0d]=ffffffea  || T_addr[1d]=000003fc
time=1221.0 ps, S_addr[0e]=ffffffe9  || T_addr[1e]=xxxxxxxx
time=1231.0 ps, S_addr[0f]=ffffffe8  || T_addr[1f]=xxxxxxxx
FINISH READING REGISTER FILE AFTER BREAK!!!
Stopped at time : 1231 ns : File
"C:/Users/chealykim/Desktop/MultiCycle/Lab6_v_file/MCU.v" Line 1392
```

Pipeline Verifications Result:

File Edit Format View Help

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

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

3c 07 10 01    // lui   $07, 0x1001
34 e7 03 F0    // ori   $07, 0x03F0        R07   <-- 0x100103F
ac ef 00 00    // sw    $15, 0($07)       M[R07] <-- R15
00 00 00 0d    // break
```

$07

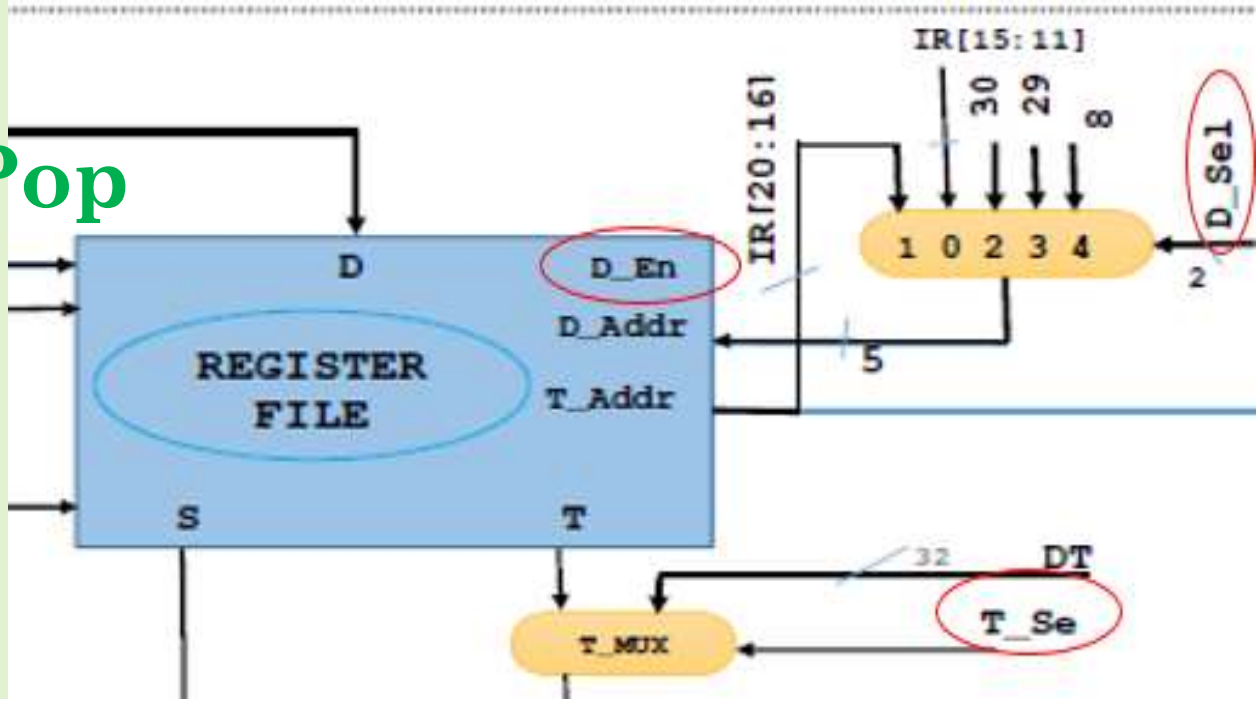
FFFFFF00

$07

100103F0

Further Enhancement

- Improve the Function of FWU
- LW
- Branch
- Jump
- Push and Pop



Thanks for Your
Attentions

Table of Contents

Purpose **2**
Instruction Set Architecture **2**
Harvard Memory Architecture and Organization 2
Machine Register Set 2
Data Type **2**
Addressing Mode 2
Instruction Set 2
R_Type Instructions 3
I-Type Instructions 23
J-Type Instructions 37
Enhanced Instructions 41
Verilog Implementation/Design/Verification 49
Source code of top level 49
Source code of Verilog modules 51
Memory modules 73
Log File Annotation 82
Memory Module_1 82
Memory Module_2 83
Memory Module_3 84
Memory Module_4 85
Memory Module_5 86
Memory Module_6 87
Memory Module_7 88
Memory Module_8 89
Memory Module_9 91
Memory Module_10 93
Memory Module_11 95
Memory Module_12 97
Memory Module_13 99
Memory_Module_14 101
Enhanced Module 103
External isr I/O Module 105
Hardware Implementation **107**
General Block Diagram 107
Detail block diagram 109
Discussion / Comment **110**

I. Purpose:

A full and complete report on customized MIPS processor instruction set(ISA). The report contains the baseline instrunctions set, example of implementation and brief explaination. To further enhance the MIPS baseline processor, some enhance instruction have been added provided with explaination and example of how to implement them.

II. Instruction set architecture:

A.Memory structure:

Baseline MIPS is RICS architecture wich 32 bit memory module and instruction set module are separate. Memory access directly from memory module is restricted. For load and store instruction, CPU registers file is used to direct the datapat. All operands are for CPU only to improve the through put. 32-bit space of address and byte addressable, where each byte is one memory address. The memory is in big endian format, which means most significant bit is at the low addressing mode.

B.Machine Register Set:

The registers file is 32-bit register with 32 registers. Five bit address, two read ports and one write port. Reading from register is asynchronous, and writing to it is synchronous. A 32-bit Program Counter to read the instruction from the instruction memory. It has the function of auto increament after each instruction is loaded. The PC loads in different PC address when the instruction branch or jump are executed. The 5-bit flags register keep the flag generated by the ALU and the interrupt bit. .

C. Data Types:

Baseline MIPS data type is 32-bit signed and unsigned interger.

D. Addressing Modes:

Immediate, Register, Register indirect are the addressing mode used in MIPS architecture. Fore Immediate modes, there are 16-bit and 26-bit. 16-bit used to calculate the PC address for branching instruction execution. 26-bit used when jump instruction is executed to calculate the next address to go to by shifte left twice and concatenate with upper 4-bit of the PC.

E.Instruction Set:

There are three type of instruction sets for MIPS baseline, R-type, I-type and J-type. For customized enhanced MIPS processor, enhanced instruction type is added. Six files for R-Type, 4 files for I-type and 2 files for J-type. The number of opcoded bits are fixed for all three file, 6-bits. The address bit are fixed as 5-bit. The rest are based on the instructions type.

1. R Type Instructions:IR[31:26] == 0

Name	Opcode[5:0]	Name	Opcode[5:0]
sll	0x00	sub	0x22
srl	0x02	subu	0x23
sra	0x03	and	0x24
jr	0x08	or	0x25
mfhi	0x10	xor	0x26
mflo	0x12	nor	0x27
mult	0x18	slt	0x2A
div	0x1A	sltu	0x2B
add	0x20	break	0x0D
addu	0x21	setie	0x1F

R type Instruction:

setie **set maskable interrupt**

31 – 26	25 – 6	5 – 0
Opcode 0x00	unused	Funct 0x1f

31 – 26	25 – 6	5 – 0
Opcode 0x00	unused	Funct 0x1F

Format: retie

Purpose:

To allow the interrupt input to the processor

Description: retie

In implementing the interrupt from external device, maskable interrupt needed to be set to allow the CPU to response to the interrupt. The interrupt is deasserted when interrupt acknowledge is asserted.

Restriction:

No register is needed in this instruction.

Operation:

Setie

Exception:

N/A

Programming Notes:

N/A

Implementation:

N/A

sll **logically shift a 32-bit value to the left**

31 – 26	25 – 21	20 – 16	15 – 11	10 – 6	5 – 0
Opcode 0x00	rs	rt	rd	shamt	Funct 0x00

31 – 26	25 – 21	20 – 16	15 – 11	10 – 6	5 – 0
Opcode 0x00	rs 0x00	rt 0x11	rd 0x9	shamt 4	Funct 0x00

Format: sll rd, rt, 4 ⟹ sll $t1, $s1, 4;

Purpose:

Shift left logical operation is to shift the value in a source register by some numbers of bits and place it in destination register.

Description: reg[rd] ←-- reg[rt] << shamt

Shift left the content in reg[rt] by amount of bits and fill the right empty bits with 0. The result after the shifting is placed in the destination register reg[rd].

1111_1000_1111_0011_1110_0110_0111_1111

<< 4 : ~~1111~~_1000_1111_0011_1110_0110_0111_1111_**0000**

Restriction:

General purpose registers only to be used in the operation are register 8 to 25. As in hex are register 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF, 0x10, 0x11, 0x12, 0X13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19.

Operation:

reg[rd] ←-- reg[rt] << shamt

Exception:

N/A

Programming Notes:

N/A

Implementation:

N/A

srl **logically shift a 32-bit value to the right**

31 – 26	25 – 21	20 – 16	15 – 11	10 – 6	5 – 0
Opcode 0x02	rs	rt	rd	shamt	Funct 0x02

31 – 26	25 – 21	20 – 16	15 – 11	10 – 6	5 – 0
Opcode 0x02	rs 0x00	rt 0x11	rd 0x9	shamt 4	Funct 0x02

Format: slr rd, rt, 4 sll $t1, $s1, 4;

Purpose:

Shift right logically the 32-bit data to the right and place the new value in the destination register.

Description: reg[rd] ←-- reg[rt] << shamt

Shift left the content in reg[rt] by amount of bits and fill the right empty bits with 0. The result after the shifting is placed in the destination register reg[rd].

1111_1000_1111_0011_1110_0110_0111_1111

>> 4: 0000_1111_1000_1111_0011_1110_0110_0111_~~1111~~

Restriction:

General purpose registers only to be used in the operation are register 8 to 25. As in hex are register 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF, 0x10, 0x11, 0x12, 0X13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19.

Operation:

reg[rd] ←-- reg[rt] >> shamt

Exception:

N/A

Programming Notes:

N/A

Implementation:

N/A

sra **shift right arithmetic 32-bit value**

31–26	25–21	20–16	15–11	10–6	5–0
Opcode 0x03	rs	rt	rd	shamt	Funct 0x03

⇩

31–26	25–21	20–16	15–11	10–6	5–0
Opcode 0x03	rs 0x00	rt 0x11	rd 0x9	shamt 4	Funct 0x03

Format: sra rd, rt, 4 sll $t1, $s1, 4;

Purpose:

Shift a 32-bit value to the right and keep the signed bit.

Description: reg[rd] ←-- reg[rt] >>> shamt

Shift the 32-bit value to the rigt bit by bit based on the amount of the shift bit. The signed bit, the most significant bit, is kept and fill in the blank bit with 0. The new value is placed in the destination register reg[rd].

1111_1000_1111_0011_1110_0110_0111_1111

>> 4: 1000_1111_1000_1111_0011_1110_0110_0111_~~1111~~

Restriction:

General purpose registers only to be used in the operation are register 8 to 25. As in hex are register 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF, 0x10, 0x11, 0x12, 0X13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19.

Operation:

temp ←-- reg[rt] >> 4

reg[rd] ←-- temp || 0x8FFFFFFF;

Exception:

N/A

Programming Notes:

N/A

Implementation:

N/A

jr **jump register**

31 – 26	25 – 21	20 – 16	15 – 11	10 – 6	5 – 0
Opcode 0x08	rs	rt	rd	shamt	Funct 0x08

31 – 26	25 – 21	20 – 16	15 – 11	10 – 6	5 – 0
Opcode 0x08	rs 0x11	rt 0x00	rd 0x00	shamt 0	Funct 0x08

Format: jr rs; jr $s1

Purpose:

Jump to a the based address hold in the source register as its content.

Description: address ←-- reg[rs]

Go to the address that the register pointed to after returning from the caller.

Restriction:

General purpose registers only to be used in the operation are register 8 to 25. As in hex are register 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF, 0x10, 0x11, 0x12, 0X13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19. It also include register 0x1F.

Operation:

PC ←-- rs[address];

Exception:

N/A

Programming Notes:

N/A

Implementation:

N/A

mfhi **move from hi 32-bit value**

31 – 26	25 – 21	20 – 16	15 – 11	10 – 6	5 – 0
Opcode 0x10	rs	rt	rd	shamt	Funct 0x10

31 – 26	25 – 21	20 – 16	15 – 11	10 – 6	5 – 0
Opcode 0x10	rs 0x00	rt 0x11	rd 0x9	shamt 0	Funct 0x10

Format: mfhi rd, rt mfhi $t1, $s1;

Purpose:

Move the upper 16 bits value to the destination register.

Description: reg[rd] ←-- reg[rt]

Take only the upper most significant bit from a source register $s1(0x11) and place it in the destination register $t1(x09) and fill the rest msb bits with 0.

Restriction:

General purpose registers only to be used in the operation are register 8 to 25. As in hex are register 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF, 0x10, 0x11, 0x12, 0X13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19.

Operation:

reg[rd] ←-- {16'b0, reg[31:16]};

Exception:

N/A

Programming Notes:

N/A

Implementation:

N/A

mflo **move from low 32-bit value**

31 – 26	25 – 21	20 – 16	15 – 11	10 – 6	5 – 0
Opcode 0x12	rs	rt	rd	shamt	Funct 0x12

31 – 26	25 – 21	20 – 16	15 – 11	10 – 6	5 – 0
Opcode 0x12	rs 0x00	rt 0x11	rd 0x9	shamt 0x00	Funct 0x12

Format: mflo rd, rt mflo $t1, $s1;

Purpose:

Move the lower 16 bits value to the destination register.

Description: reg[rd] ←-- reg[rt]

Take only the lower 16-bit from a source register $s1(0x11) and place it in the destination register $t1(0x9) and fill the rest of the upper bits with 0.

Restriction:

General purpose registers only to be used in the operation are register 8 to 25. As in hex are register 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF, 0x10, 0x11, 0x12, 0X13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19.

Operation:

reg[rd] ←-- {16'b0, reg[15:0]};

Exception:

N/A

Programming Notes:

N/A

Implementation:

N/A

mult **multiply two 32-bit values**

31–26	25–21	20–16	15–11	10–6	5–0
Opcode 0x18	rs	rt	rd	shamt	Funct 0x18

⇩

31–26	25–21	20–16	15–11	10–6	5–0
Opcode 0x18	rs 0x11	rt 0x12	rd 0x00	shamt 0x00	Funct 0x18

Format: mult {hi, lo}, rs * rt mult {$t1, $t2}, $s1 * $s2;

Purpose:

Multiply 32-bit value in the two source register of register “rs” and “rt”.

Description: {hi[63:32], lo[31:0]} ←-- reg[rs] * reg[rt]

Multiplication of one 32-bit value from $s1 and another 32-bit value from $s2. Place the 64-bit result in two 32-bit registers.

Restriction:

General purpose registers only to be used in the operation are register 8 to 25. As in hex are register 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF, 0x10, 0x11, 0x12, 0X13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19.

Operation:

temp ←-- reg[rs] * reg[rt];

reg[hi] ←-- temp[63:32];

reg[lo] ←-- temp[31:0];

Exception:

N/A

Programming Notes:

Using a temporary integer to hold the 64-bit result from multiplication. Partition the upper 32-bit and lower 32-bit to each 32-bit register accordingly.

Implementation:

Use two registers to hold the 64-bit value of the result.

div **divide two 32-bit values**

31 – 26	25 – 21	20 – 16	15 – 11	10 – 6	5 – 0
Opcode 0x1A	rs	rt	rd	shamt	Funct 0x1A

31 – 26	25 – 21	20 – 16	15 – 11	10 – 6	5 – 0
Opcode 0x1A	rs 0x11	rt 0x12	rd 0x00	shamt 0x00	Funct 0x1A

Format: div {hi, lo}, rs / rt div {$t1, $t2}, $s1/$s2;

Purpose:

Divide a 32-bit value with another 32-bit value from two source register of register "rs" and "rt" and place result in two registers

Description: {hi[63:32], lo[31:0]} ←-- reg[rs] / reg[rt]

Divide a 32-bit value from register $s1 with 32-bit value from register $s2. Place the remainder and quotient in two 32-bti registers.

Restriction:

General purpose registers only to be used in the operation are register 8 to 25. As in hex are register 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF, 0x10, 0x11, 0x12, 0X13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19.

Operation:

reg[hi] ←-- reg[rs] % reg[rt];

reg[lo] ←-- reg[rs] / reg[rt];

Exception:

N/A

Programming Notes:

The remainder is place in the 32 upper bit of a 64-bit value. And the quotient is placed in the lower 32-bit of the 64-bit value.

Implementation:

Use two registers to hold each result as specified.

add **adding two 32-bit signed values**

31 – 26	25 – 21	20 – 16	15 – 11	10 – 6	5 – 0
Opcode 0x20	rs	rt	rd	shamt	Funct 0x20

⇩

31 – 26	25 – 21	20 – 16	15 – 11	10 – 6	5 – 0
Opcode 0x20	rs 0x11	rt 0x12	rd 0x9	shamt 0x00	Funct 0x20

Format: add rd, rs, rt add $t1, $s1, $s2;

Purpose:

Addition of a 32-bit value with another 32-bit value from two source register of register "rs" and "rt" and place result in the destination register "rd".

Description: reg[rd]←-- reg[rs] + reg[rt]

Adding two 32-bit value from two source registers and place the result in the 32-bit destination register. The value are signed. The carry and overflow flag is sent implicitly to the 5-bit flags register.

Restriction:

General purpose registers only to be used in the operation are register 8 to 25. As in hex are register 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF, 0x10, 0x11, 0x12, 0X13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19.

Operation:

temp[32:0] ←-- reg[rs] + reg[rt];

reg[rd] ←-- temp[31:0];

reg[flags] ←-- temp[32];

Exception:

Carry and overflow flag occur. The flag is implicitly save in 5-bit flag register.

Programming Notes:

Use a temporary variable of 33-bit to hold the carry flag if there is a carry flag. The overflow flag will be implicitly sent to the 5-bit flags register.

Implementation:

The lower 31 bit from the temporary variable is saved to the destination register "rd" as the result of the operation.

addu **adding two 32-bit unsigned values**

31 – 26	25 – 21	20 – 16	15 – 11	10 – 6	5 – 0
Opcode 0x21	rs	rt	rd	shamt	Funct 0x21

31 – 26	25 – 21	20 – 16	15 – 11	10 – 6	5 – 0
Opcode 0x21	rs 0x11	rt 0x12	rd 0x9	shamt 0x00	Funct 0x21

Format: addu rd, rs, rt addu $t1, $s1, $s2;

Purpose:

Addition of a 32-bit value with another 32-bit value from two source register of register "rs" and "rt" and place result in the destination register "rd".

Description: reg[rd]←-- reg[rs] + reg[rt]

Adding two 32-bit value from two source registers and place the result in the 32-bit destination register. The value are unsigned. The carry and overflow flag is sent implicitly to the 5-bit flags register.

Restriction:

General purpose registers only to be used in the operation are register 8 to 25. As in hex are register 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF, 0x10, 0x11, 0x12, 0X13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19.

Operation:

temp[32:0] ←-- reg[rs] + reg[rt];

reg[rd] ←-- temp[31:0];

reg[flags] ←-- temp[32];

Exception:

Carry and overflow flag occur. The flag is implicitly save in 5-bit flag register.

Programming Notes:

Use a temporary variable of 33-bit to hold the carry flag if there is a carry flag. The carry flag will be implicitly sent to the 5-bit flags register.

Implementation:

The lower 31 bit from the temporary variable is saved to the destination register "rd" as the result of the operation.

sub **subtract two 32-bit signed values**

31 – 26	25 – 21	20 – 16	15 – 11	10 – 6	5 – 0
Opcode 0x22	rs	rt	rd	shamt	Funct 0x22

31 – 26	25 – 21	20 – 16	15 – 11	10 – 6	5 – 0
Opcode 0x22	rs 0x11	rt 0x12	rd 0x9	shamt 0x00	Funct 0x22

Format: sub rd, rs, rt sub $t1, $s1, $s2;

Purpose:

Substraction of a 32-bit value with another 32-bit value from two source register of register "rs" and "rt" and place result in the destination register "rd".

Description: reg[rd]←-- reg[rs] - reg[rt]

Substracting two 32-bit value from two source registers and place the result in the 32-bit destination register. The value are signed. The carry and overflow flag is sent implicitly to the 5-bit flags register.

Restriction:

General purpose registers only to be used in the operation are register 8 to 25. As in hex are register 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF, 0x10, 0x11, 0x12, 0X13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19.

Operation:

temp[32:0] ←-- reg[rs] - reg[rt];

reg[rd] ←-- temp[31:0];

reg[flags] ←-- temp[32];

Exception:

Carry and overflow flag occur. The flag is implicitly save in 5-bit flag register.

Programming Notes:

Use a temporary variable of 33-bit to hold the carry flag if there is a carry flag. The overflow flag will be implicitly sent to the 5-bit flags register.

Implementation:

The lower 31 bit from the temporary variable is saved to the destination register "rd" as the result of the operation.

subu **subtract two 32-bit unsigned values**

31 – 26	25 – 21	20 – 16	15 – 11	10 – 6	5 – 0
Opcode 0x23	rs	rt	rd	shamt	Funct 0x23

31 – 26	25 – 21	20 – 16	15 – 11	10 – 6	5 – 0
Opcode 0x23	rs 0x11	rt 0x12	rd 0x9	shamt 0x00	Funct 0x23

Format: subu rd, rs, rt subu $t1, $s1, $s2;

Purpose:

Subtraction of a 32-bit value with another 32-bit value from two source register of register "rs" and "rt" and place result in the destination register "rd".

Description: reg[rd]←-- reg[rs] - reg[rt]

Subtracting two 32-bit value from two source registers and place the result in the 32-bit destination register. The value are unsigned. The carry and overflow flag is sent implicitly to the 5-bit flags register.

Restriction:

General purpose registers only to be used in the operation are register 8 to 25. As in hex are register 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF, 0x10, 0x11, 0x12, 0X13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19.

Operation:

temp[32:0] ←-- reg[rs] - reg[rt];

reg[rd] ←-- temp[31:0];

reg[flags] ←-- temp[32];

Exception:

Carry and overflow flag occur. The flag is implicitly save in 5-bit flag register.

Programming Notes:

Use a temporary variable of 33-bit to hold the carry flag if there is a carry flag. The carry flag will be implicitly sent to the 5-bit flags register.

Implementation:

The lower 31 bit from the temporary variable is saved to the destination register "rd" as the result of the operation.

and **bit wise and two 32-bit values**

31 – 26	25 – 21	20 – 16	15 – 11	10 – 6	5 – 0
Opcode 0x24	rs	rt	rd	shamt	Funct 0x24

⇩

31 – 26	25 – 21	20 – 16	15 – 11	10 – 6	5 – 0
Opcode 0x24	rs 0x11	rt 0x12	rd 0x9	shamt 0x00	Funct 0x24

Format: and rd, rs, rt and $t1, $s1, $s2;

Purpose:

Bit wise and operation of two 32-bit values from two source register of register "rs" and "rt" and place result in the destination register "rd".

Description: reg[rd]←-- reg[rs] & reg[rt]

Adding the two 32-bit value bit with bit and place the value in the destination register. This is a bit wise operation with two operands

Restriction:

General purpose registers only to be used in the operation are register 8 to 25. As in hex are register 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF, 0x10, 0x11, 0x12, 0X13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19.

Operation:

reg[rd] ←-- reg[rs] & reg[rt];

Exception:

N/A

Programming Notes:

N/A

Implementation:

N/A

or **bit wise or two 32-bit values**

31 – 26	25 – 21	20 – 16	15 – 11	10 – 6	5 – 0
Opcode 0x25	rs	rt	rd	shamt	Funct 0x25

31 – 26	25 – 21	20 – 16	15 – 11	10 – 6	5 – 0
Opcode 0x25	rs 0x11	rt 0x12	rd 0x9	shamt 0x00	Funct 0x25

Format: or rd, rs, rt or $t1, $s1, $s2;

Purpose:

Bit wise or operation of two 32-bit values from two source register of register "rs" and "rt" and place result in the destination register "rd".

Description: reg[rd]←-- reg[rs] | reg[rt]

Orring the two 32-bit value bit with bit and place the value in the destination register. This is a bit wise operation with two operands.

Restriction:

General purpose registers only to be used in the operation are register 8 to 25. As in hex are register 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF, 0x10, 0x11, 0x12, 0X13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19.

Operation:

reg[rd] ←-- reg[rs] | reg[rt];

Exception:

N/A

Programming Notes:

N/A

Implementation:

N/A

xor **bit wise xor two 32-bit values**

31 – 26	25 – 21	20 – 16	15 – 11	10 – 6	5 – 0
Opcode 0x26	rs	rt	rd	shamt	Funct 0x26

⇩

31 – 26	25 – 21	20 – 16	15 – 11	10 – 6	5 – 0
Opcode 0x26	rs 0x11	rt 0x12	rd 0x9	shamt 0x00	Funct 0x26

Format: xor rd, rs, rt xor $t1, $s1, $s2;

Purpose:

Bit wise xor operation of two 32-bit values from two source register of register "rs" and "rt" and place result in the destination register "rd".

Description: reg[rd]←-- reg[rs] ^ reg[rt]

Xorring the two 32-bit value bit with bit and place the value in the destination register. This is a bit wise operation with two operands.

Restriction:

General purpose registers only to be used in the operation are register 8 to 25. As in hex are register 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF, 0x10, 0x11, 0x12, 0X13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19.

Operation:

reg[rd] ←-- reg[rs] ^ reg[rt];

Exception:

N/A

Programming Notes:

N/A

Implementation:

N/A

nor **bit wise nor two 32-bit values**

31 – 26	25 – 21	20 – 16	15 – 11	10 – 6	5 – 0
Opcode 0x27	rs	rt	rd	shamt	Funct 0x27

31 – 26	25 – 21	20 – 16	15 – 11	10 – 6	5 – 0
Opcode 0x27	rs 0x11	rt 0x12	rd 0x9	shamt 0x00	Funct 0x27

Format: nor rd, rs, rt nor $t1, $s1, $s2;

Purpose:

Bit wise or operation of two 32-bit values from two source register of register "rs" and "rt", negate the result and place result in the destination register "rd".

Description: reg[rd] ←-- ~(reg[rs] | reg[rt])

Orring the two 32-bit value bit with bit and nor the result and place the value in the destination register. This is a bit wise operation with two operands.

Restriction:

General purpose registers only to be used in the operation are register 8 to 25. As in hex are register 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF, 0x10, 0x11, 0x12, 0X13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19.

Operation:

reg[rd] ←-- ~(reg[rs] | reg[rt]);

Exception:

N/A

Programming Notes:

N/A

Implementation:

N/A

slt **signed comparison 32-bit value**

31 – 26	25 – 21	20 – 16	15 – 11	10 – 6	5 – 0
Opcode 0x2A	rs	rt	rd	shamt	Funct 0x2A

31 – 26	25 – 21	20 – 16	15 – 11	10 – 6	5 – 0
Opcode 0x2A	rs 0x11	rt 0x12	rd 0x9	shamt 0x00	Funct 0x2A

Format: slt rd, rs, rt slt $t1, $s1, $s2;

Purpose:

Compare two signed 32-bit values if the first value is less than the second value.

Description: reg[rd] ←-- (reg[rs] < reg[rt])? 1: 0

Signed value in register "rs" is compare to that of "rt" if "rs" < "rt", destination register "rd" get value of "1", and if not "rd" == 0.

Restriction:

General purpose registers only to be used in the operation are register 8 to 25. As in hex are register 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF, 0x10, 0x11, 0x12, 0X13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19.

Operation:

```
if (reg[rs] < reg[rt])
     reg[rd] ←-- 1;
else
     reg[rd] ←-- 0;
```

Exception:

N/A

Programming Notes:

N/A

Implementation:

N/A

sltu **unsigned comparison 32-bit value**

31 – 26	25 – 21	20 – 16	15 – 11	10 – 6	5 – 0
Opcode 0x2B	rs	rt	rd	shamt	Funct 0x2B

31 – 26	25 – 21	20 – 16	15 – 11	10 – 6	5 – 0
Opcode 0x2B	rs 0x11	rt 0x12	rd 0x9	shamt 0x00	Funct 0x2B

Format: sltu rd, rs, rt sltu $t1, $s1, $s2;

Purpose:

Compare two unsigned 32-bit values if the first value is less than the second value.

Description: reg[rd] ←-- (reg[rs] < reg[rt])? 1: 0

Unsigned value in register "rs" is compare to that of "rt" if "rs" < "rt", destination register "rd" get value of "1", and if not "rd" == 0.

Restriction:

General purpose registers only to be used in the operation are register 8 to 25. As in hex are register 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF, 0x10, 0x11, 0x12, 0X13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19.

Operation:

```
if (reg[rs] < reg[rt])
     reg[rd] ←-- 1;
else
     reg[rd] ←-- 0;
```

Exception:

N/A

Programming Notes:

N/A

Implementation:

N/A

2. I Type Instructions

Name	Opcode[31:26]	Name	Opcode[31:26]
beq	0x04	sltiu	0x0B
bne	0x05	andi	0x0C
blez	0x06	ori	0x0D
bgtz	0x07	xori	0x0E
addi	0x08	lui	0x0F
slti	0x0A	lw	0x23
		sw	0x2B

I type instruction set:

beq **conditional branch instruction**

31 – 26	25 – 21	20 – 16	15 – 0
Opcode 0x04	rs code	rt code	16-bit immediate

31 – 26	25 – 21	20 – 16	15 – 0
Opcode 0x4	rs code 0x11	rt code 0x12	16-bit immediate offset

Format: beq rs, rt, offset beq $s1, $s2, lable

Purpose:

Compare if two 32-bit values are equal than branch to the address of offset

Description: if (reg[rs] == reg[rt]) branch

A 16-bit offset is converted to a 32-bit value by concatenating two "0" to the lower bit of the 16-bit and make it to be 18-bit and signed extend the 18 bits. The value is added to the current PC(PC+4) as the address to branch to if the condition is met.

Restriction:

General purpose registers only to be used in the operation are register 8 to 25. As in hex are register 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF, 0x10, 0x11, 0x12, 0X13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19.

Operation:

```
Offset ←-- signed-extended({offset, 2'b00});

     Condition ←-- (reg[rs] == reg[rt])

if condition true

     PC ←-- PC + offset

end
```

Exception: N/A

Programming Notes:

With 16-bit signed extended, the address can go from only -32768 to 32767(-2^15 to 2^15-1).

Implementation:

N/A

bne **not equal branch instruction**

31 – 26	25 – 21	20 – 16	15 – 0
Opcode 0x05	rs code	rt code	16-bit immediate

31 – 26	25 – 21	20 – 16	15 – 0
Opcode 0x5	rs code 0x11	rt code 0x12	16-bit immediate offset

Format: bne rs, rt, offset bne $s1, $s2, lable

Purpose:

Compare if two 32-bit values are not equal than branch to the address of offset

Description: if (reg[rs] != reg[rt]) branch

A 16-bit offset is converted to a 32-bit value by concatenating two "0" to the lower bit of the 16-bit and make it to be 18-bit and signed extend the 18 bits. The value is added to the current PC(PC+4) as the address to branch to if the condition is met.

Restriction:

General purpose registers only to be used in the operation are register 8 to 25. As in hex are register 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF, 0x10, 0x11, 0x12, 0X13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19.

Operation:

```
Offset ←-- signed-extended({offset, 2'b00});

     Condition ←-- (reg[rs] != reg[rt])

if condition true

     PC ←-- PC + offset

end
```

Exception: N/A

Programming Notes:

With 16-bit signed extended, the address can go from only -32768 to 32767(-2^15 to 2^15-1).

Implementation:

N/A

blez **branch less than or equal zero**

31–26	25–21	20–16	15–0
Opcode 0x06	rs code	rt code	16-bit immediate

31–26	25–21	20–16	15–0
Opcode 0x6	rs code 0x11	rt code 0x00	16-bit immediate offset

Format: blez rs, rt, offset blez $s1, $zero, lable

Purpose:

Compare if the value in the source register is greater than zero. If it is not than branch to the offset address.

Description: if (reg[rs] <= reg[rt]) branch

A 16-bit offset is converted to a 32-bit value by concatenating two "0" to the lower bit of the 16-bit and make it to be 18-bit and signed extend the 18 bits. The value is added to the current PC(PC+4) as the address to branch to if the condition is met.

Restriction:

General purpose registers only to be used in the operation are register 8 to 25. As in hex are register 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF, 0x10, 0x11, 0x12, 0X13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19.

Operation:

```
Offset ←-- signed-extended({offset, 2'b00});

     Condition ←-- (reg[rs] <= reg[rt])

if condition true

     PC ←-- PC + offset

end
```

Exception: N/A

Programming Notes:

With 16-bit signed extended, the address can go from only -32768 to 32767(-2^15 to 2^15-1).

Implementation:

N/A

bgtz **branch greater than zero**

31–26	25–21	20–16	15–0
Opcode 0x07	rs code	rt code	16-bit immediate

31–26	25–21	20–16	15–0
Opcode 0x7	rs code 0x11	rt code 0x00	16-bit immediate offset

Format: bgtz rs, rt, offset bgtz $s1, $zero, lable

Purpose:

Compare if the value in the source register is greater than zero. If it is than branch to the offset address.

Description: if (reg[rs] > reg[rt]) branch

A 16-bit offset is converted to a 32-bit value by concatenating two "0" to the lower bit of the 16-bit and make it to be 18-bit and signed extend the 18 bits. The value is added to the current PC(PC+4) as the address to branch to if the condition is met.

Restriction:

General purpose registers only to be used in the operation are register 8 to 25. As in hex are register 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF, 0x10, 0x11, 0x12, 0X13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19.

Operation:

```
Offset ←-- signed-extended({offset, 2'b00});

      Condition ←-- (reg[rs] <= reg[rt])

if condition true

      PC ←-- PC + offset

end
```

Exception: N/A

Programming Notes:

With 16-bit signed extended, the address can go from only -32768 to 32767(-2^15 to 2^15-1).

Implementation:

N/A

addi **add immediate value**

31 – 26	25 – 21	20 – 16	15 – 0
Opcode 0x08	rs code	rt code	16-bit immediate

31 – 26	25 – 21	20 – 16	15 – 0
Opcode 0x8	rs code 0x11	rt code 0x12	16-bit immediate 4

Format: addi rt, rs, immediate value addi $s2, $s1, 4;

Purpose:

Add an immediate value to the content of the source register.

Description: reg[rt] ←-- reg[rs] + immediate value

Adding an immediate value to the content of source register and place the result to the destination register ($s2).

Restriction:

General purpose registers only to be used in the operation are register 8 to 25. As in hex are register 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF, 0x10, 0x11, 0x12, 0X13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19.

Operation:

Reg[rt] ←--- reg[rs] + immediate value

Exception: N/A

Programming Notes:

With 16-bit immediate value, the value can go from only -32768 to 32767(-2^15 to 2^15-1).

Implementation:

N/A

slti **compare with signed immediate**

31 – 26	25 – 21	20 – 16	15 – 0
Opcode 0x0A	rs code	rt code	16-bit immediate

31 – 26	25 – 21	20 – 16	15 – 0
Opcode 0xA	rs code 0x11	rt code 0x12	16-bit immediate -4

Format: slti rt, rs, immediate value slti $s2, $s1, -4;

Purpose:

Compare the signed immediate value to the content in the source register "rs"

Description: reg[rt] ←-- reg[rs]<(signedExtended_immediate)?1:0

Compare the immediate value with the content of source register if "rs" < immediate value, destination register get "1" otherwise "0".

Restriction:

General purpose registers only to be used in the operation are register 8 to 25. As in hex are register 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF, 0x10, 0x11, 0x12, 0X13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19.

Operation:

```
condition ←---  reg[rs] < immediate;

if (condition == true)

     reg[rt] ←-- 1;

else

     reg[rt] ←-- 0;
```

Exception: N/A

Programming Notes:

With 16-bit immediate value, the value can go from only -32768 to 32767(-2^15 to 2^15-1).

Implementation:

N/A

sltiu **compare with unsigned immediate**

31–26	25–21	20–16	15–0
Opcode 0x0B	rs code	rt code	16-bit immediate

31–26	25–21	20–16	15–0
Opcode 0x0B	rs code 0x11	rt code 0x12	16-bit immediate 4

Format: sltiu rt, rs, immediate value sltiu $s2, $s1, 4;

Purpose:

Compare the unsigned immediate value to the content in the source register "rs"

Description: reg[rt] ←-- reg[rs]<(signedExtended_immediate)?1:0

Compare the immediate value with the content of source register if "rs" < immediate value, destination register get "1" otherwise "0".

Restriction:

General purpose registers only to be used in the operation are register 8 to 25. As in hex are register 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF, 0x10, 0x11, 0x12, 0X13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19.

Operation:

```
condition ←---  reg[rs] < immediate;

if (condition == true)

     reg[rt] ←-- 1;

else

     reg[rt] ←-- 0;
```

Exception: N/A

Programming Notes:

With 16-bit immediate value, the value can go from only -32768 to 32767(-2^15 to 2^15-1).

Implementation:

N/A

andi **bit wise and with immediate**

31 – 26	25 – 21	20 – 16	15 – 0
Opcode 0x0C	rs code	rt code	16-bit immediate

31 – 26	25 – 21	20 – 16	15 – 0
Opcode 0x0C	rs code 0x11	rt code 0x12	16-bit immediate 0xff00

Format: andi rt, rs, immediate value andi $s2, $s1, 0xff00;

Purpose:

Bit wise and immediate value to the content in the source register “rs”.

Description: reg[rt] ←-- reg[rs] & 0xff00;

Bit wise and the immediate value with the content of source register “rs”, and place the result in the destination reg[rt].

Restriction:

General purpose registers only to be used in the operation are register 8 to 25. As in hex are register 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF, 0x10, 0x11, 0x12, 0X13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19.

Operation:

Reg[rt] ←-- reg[rs] & 0xff00;

Exception: N/A

Programming Notes:

With 16-bit immediate value, the value can go from only -32768 to 32767(-2^15 to 2^15-1).

Implementation:

N/A

ori **bit wise or with immediate**

31 – 26	25 – 21	20 – 16	15 – 0
Opcode 0x0D	rs code	rt code	16-bit immediate

31 – 26	25 – 21	20 – 16	15 – 0
Opcode 0x0D	rs code 0x11	rt code 0x12	16-bit immediate 0xff00

Format: ori rt, rs, immediate value ori $s2, $s1, 0xff00;

Purpose:

Bit wise or immediate value to the content in the source register "rs".

Description: reg[rt] ←-- reg[rs] | 0xff00;

Bit wise or the immediate value with the content of source register "rs", and place the result in the destination reg[rt].

Restriction:

General purpose registers only to be used in the operation are register 8 to 25. As in hex are register 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF, 0x10, 0x11, 0x12, 0X13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19.

Operation:

Reg[rt] ←-- reg[rs] | 0xff00;

Exception: N/A

Programming Notes:

With 16-bit immediate value, the value can go from only -32768 to 32767(-2^15 to 2^15-1).

Implementation:

N/A

xori **bit wise xor with immediate**

31 – 26	25 – 21	20 – 16	15 – 0
Opcode 0x0E	rs code	rt code	16-bit immediate

31 – 26	25 – 21	20 – 16	15 – 0
Opcode 0x0E	rs code 0x11	rt code 0x12	16-bit immediate 0xff00

Format: xori rt, rs, immediate value xori $s2, $s1, 0xff00;

Purpose:

Bit wise xor immediate value to the content in the source register "rs".

Description: reg[rt] ←-- reg[rs] ^ 0xff00;

Bit wise xor the immediate value with the content of source register "rs", and place the result in the destination reg[rt].

Restriction:

General purpose registers only to be used in the operation are register 8 to 25. As in hex are register 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF, 0x10, 0x11, 0x12, 0X13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19.

Operation:

Reg[rt] ←-- reg[rs] ^ 0xff00;

Exception: N/A

Programming Notes:

With 16-bit immediate value, the value can go from only -32768 to 32767(-2^15 to 2^15-1).

Implementation:

N/A

lui **load upper immediate**

31 – 26	25 – 21	20 – 16	15 – 0
Opcode 0x0F	rs code	rt code	16-bit immediate

31 – 26	25 – 21	20 – 16	15 – 0
Opcode 0x0F	rs code 0x00	rt code 0x12	16-bit immediate 0xAAAA

Format: lui rt, immediate value lui $s2, 0xAAAA;

Purpose:

Load the 16-bit immediate value to the upper bit of the destination register "rt".

Description: reg[rt] ←-- {immediate, 16'b0};

Make the 16-bit immediate value to 32-bit by concatenating 16 low-order zero to make it a 32-bit immediate value with its previous value at the upper 16 bits and load it to "rt", which has immediate value as the upper bit.

Restriction:

General purpose registers only to be used in the operation are register 8 to 25. As in hex are register 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF, 0x10, 0x11, 0x12, 0X13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19.

Operation:

reg[rt] ←-- {immediate, 16'b0};

Exception: N/A

Programming Notes:

With 16-bit immediate value, the value can go from only -32768 to 32767(-2^15 to 2^15-1).

Implementation:

N/A

lw **load word**

31 – 26	25 – 21	20 – 16	15 – 0
Opcode 0x23	rs code	rt code	16-bit number offset

31 – 26	25 – 21	20 – 16	15 – 0
Opcode 0x23	rs code 0x11	rt code 0x12	16-bit number 24

Format: lw rt, offset(rs) lw $s2, 24($s1);

Purpose:

Load the content value from memory addressed at offset(rs)to a destination register.

Description: reg[rt] ←-- M[offset + rs];

Register "rs" is the base address to calculate the address from which the content can be retrieved from. By adding the offset to the based address, the right address to get the value from.

Decimal 24 == 0x18. Address of $s2 = 0x12004094. The address is 0x120040ac.
reg[rt] ←-- M[0x120040ac];

Restriction:

General purpose registers only to be used in the operation are register 8 to 25. As in hex are register 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF, 0x10, 0x11, 0x12, 0X13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19.

Operation:

reg[rt] ←-- M[offset + rs];

Exception: N/A

Programming Notes:

The 16-bit offset to the base address can have value within the range of -32768 to 32767(-2^15 to 2^15-1) in decimal.

Implementation:

N/A

sw **store word**

31 – 26	25 – 21	20 – 16	15 – 0
Opcode 0x2B	rs code	rt code	16-bit number offset

31 – 26	25 – 21	20 – 16	15 – 0
Opcode 0x2B	rs code 0x11	rt code 0x12	16-bit number 24

Format: sw rt, offset(rs) sw $s2, 24($s1);

Purpose:

Store value from register to destination register "rt".

Description: M[offset + rs] ←-- reg[rt]

Register "rs" is the base address to calculate the address from which the content can be retrieved from. By adding the offset to the based address, the right address to get the value from.

Decimal 24 == 0x18. Address of $s2 = 0x12004094. The address is 0x120040ac.
M[0x120040ac] ←--- reg[rt];

Restriction:

General purpose registers only to be used in the operation are register 8 to 25. As in hex are register 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF, 0x10, 0x11, 0x12, 0X13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19.

Operation:

M[offset + rs] ←-- reg[rt]

Exception: N/A

Programming Notes:

The 16-bit offset to the base address can have value within the range of -32768 to 32767(-2^15 to 2^15-1) in decimal.

Implementation:

N/A

3. J Type Instructions

Name	Opcode[31:26]
j	0x02
jal	0x3

J-Type Instruction:

j **unconditional jump**

31 – 26	25 – 00
Opcode 0x02	26-bit Jump address label

31 – 26	25 – 21
Opcode 0x02	26-bit Jump address 0x010000c0

Format: j designated address j label

Purpose:

Go to designated address and continue executing code from there.

Description: PC ←--- jump_address

The offset is defined by the address of the label. The absolute address of destination address is formed by concatenating two zeros as the low-order bits to make a 28 bits and concatenating upper 4 bits of the current PC(PC+4) to make a 32-bit address.

26-bit of label: ~~0000_00~~00_01|00_00|00_00|00_00|00_00|11_00|00_00

PC ←-- PC+4

PC: 0000_0000_0100_0000_0000_0000_0010_1100

The new 28-bit concatenated with upper 4 bits of the PC gives address as 0x010000c0

Restriction:

General purpose registers only to be used in the operation are register 8 to 25. As in hex are register 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF, 0x10, 0x11, 0x12, 0X13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19.

Operation:

PC ←--- jum_address

Exception: N/A

Programming Notes:

The 16-bit offset to the base address can have value within the range of -32768 to 32767(-2^15 to 2^15-1) in decimal.

Implementation:

N/A

jal **jump to a procedure**

31 – 26	25 – 00
Opcode 0x03	26-bit Jump address procedure called

31 – 26	25 – 21
Opcode 0x03	26-bit Jump address ProcAddress

Format: jal procAddrs jal procAddrs

Purpose:

Go to designated address and continue executing code from there.

Description: jal procAddrs;

The absolute address of destination address is formed by concatenating two zeros as the low-order bits to make a 28 bits and concatenating upper 4 bits of the current PC(PC+4) to make a 32-bit address.

26-bit of label: ~~0000_00~~00_01|00_00|00_00|00_00|00_00|11_00|00_00

PC ←-- PC+4

PC: 0000_0000_0100_0000_0000_0000_0010_1100

The new 28-bit concatenated with upper 4 bits of the PC gives address as 0x010000c0. This is the absolute address.

PC ←-- absolute address(0x010000c0);

$ra ←--- PC + 8

Restriction:

General purpose registers only to be used in the operation are register 8 to 25. As in hex are register 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF, 0x10, 0x11, 0x12, 0X13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19. Also include the return address of $ra:0x1F, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07.

Operation:

PC ←-- PC + 4;

PC ←-- absolute address;

$ra ←-- PC + 8

Exception: N/A

Programming Notes:

The 16-bit offset to the base address can have value within the range of -32768 to 32767(-2^15 to 2^15-1) in decimal.

Implementation:

N/A

4. E Type Instructions

Name	Opcode[31:26]
input	0x1C
output	0x1D
reti	0x1E
retpc	0x1F
retflg	0x2E
push	0x2C
Pop	0x2D

E-Type Instruction:

input **reading from external interface and write to reg**

31 – 26	25 – 21	20 – 16	15 – 0
Opcode 0x1C	rs code	rt code	16-bit number offset

31 – 26	25 – 21	20 – 16	15 – 0
Opcode 0x1C	rs code 0x11	rt code 0x12	16-bit number 24

Format: input rt, offset(rs) input $s2, 24($s1);

Purpose:

Write content from $rt to addrss of $rs, the address of I/O

Description: Reg[$rt] ←--M[$rs]

Register "rs" is the base address to calculate the address from which the content can be retrieved from. By adding the offset to the based address, the right address to get the value from.

Decimal 24 == 0x18. Address of $s2 = 0x12004094. The address is 0x120040ac.

Reg[$rt] ←-- M[0x120040ac];

Restriction:

General purpose registers only to be used in the operation are register 8 to 25. As in hex are register 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF, 0x10, 0x11, 0x12, 0X13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19.

Operation:

Reg[rt] ←-- M[offset + rs];

Exception: N/A

Programming Notes:

The 16-bit offset to the base address can have value within the range of -32768 to 32767(-2^15 to 2^15-1) in decimal.

Implementation:

N/A

output **write from Reg to I/O module**

31–26	25–21	20–16	15–0
Opcode 0x1D	rs code	rt code	16-bit number offset

31–26	25–21	20–16	15–0
Opcode 0x1D	rs code 0x11	rt code 0x12	16-bit number 24

Format: output rt, offset(rs) output $s2, 24($s1);

Purpose:

Store value from register to destination register "rt".

Description: M[offset + rs] ←--- reg[rt];

Register "rs" is the base address to calculate the address from which the content can be retrieved from. By adding the offset to the based address, the right address to get the value from.

Decimal 24 == 0x18. Address of $s2 = 0x12004094. The address is 0x120040ac.
M[0x120040ac] ←---- reg[rt];

Restriction:

General purpose registers only to be used in the operation are register 8 to 25. As in hex are register 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF, 0x10, 0x11, 0x12, 0X13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19.

Operation:

M[offset + rs] ←-- reg[rt];

Exception: N/A

Programming Notes:

The 16-bit offset to the base address can have value within the range of -32768 to 32767(-2^15 to 2^15-1) in decimal.

Implementation:

N/A

reti **pop off return address to PC**

31 – 26	25 – 0
Opcode 0x00	unused

31 – 26	25 – 0
Opcode 0x1E	unused

Format: reti ;

Purpose:

Restore the return address to the program counter by popping off the flags from stack, put back to the flag register, and pop the return address from the stack.

Description: reg[rt] ←-- M[offset + rs];

Register "rs" is the base address which hold the content value of the stack pointer. "reti" after reading the flags back it increment the $rs value by and read the PC value to the programcounter and increment the stack pointer again.

Restriction:

Restriction to use on the register $rs = 29, reserved for stack pointer register.

Operation:

reti;

Exception: N/A

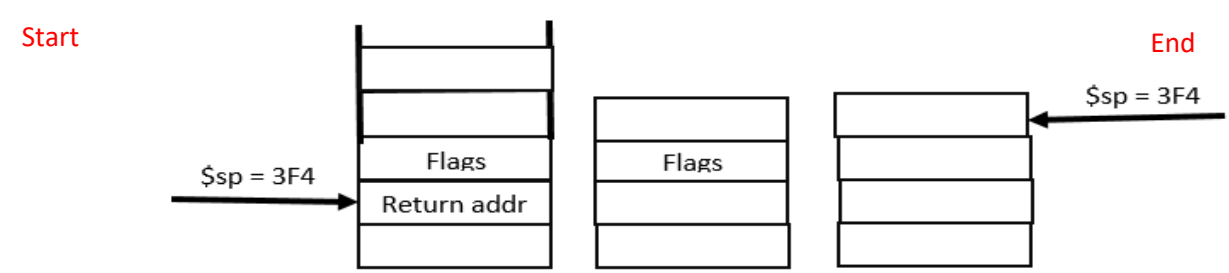
Programming Notes:

The 16-bit offset to the base address can have value within the range of -32768 to 32767(-2^15 to 2^15-1) in decimal.

Implementation:

N/A

Example of Operation



push **push Flags and PC to the stack**

31 – 26	25 – 21	20 – 16	15 – 0
Opcode 0x2C	rs code	rt code	16-bit number offset

31 – 26	25 – 21	20 – 16	15 – 0
Opcode 0x2C	rs code 5'd29	rt code 5'd31	16-bit number 00

Format: push rt, offset(rs) push $rt, 00($s1);

Purpose:

Write content of $rt to memory location of $rs. Each push the stack pointer which is $rs is decrement by 4.

Description: M[offset + rs] ←--- reg[rt];

Register "rs" is the base address to calculate the address from which the content can be retrieved from. By adding the offset to the based address, the right address to get the value from.

Decimal 24 == 0x18. Address of $s2 = 0x12004094. The address is 0x120040ac.
reg[rt] ←-- reg[0x120040ac];

Restriction:

This operation is used only on the $sp==29 as $rs, which is the stack pointer. The $rt = 31 or $rt = 8 only. Since push get PC from $31 and Flags from $8

Operation:

[offset + rs] ←--$rt(31 or 8)

Exception: N/A

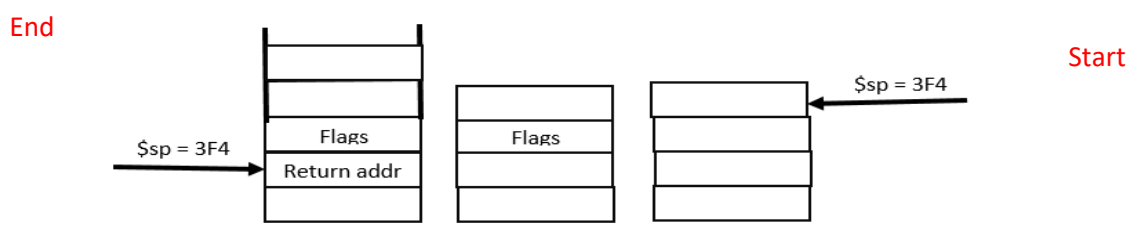
Programming Notes:

The 16-bit offset to the base address can have value within the range of -32768 to 32767(-2^15 to 2^15-1) in decimal.

Implementation:

N/A

Example of Operation



pop **read PC and Flags from Mem and write to $r8**

31 – 26	25 – 21	20 – 16	15 – 0
Opcode 0x2D	rs code	rt code	16-bit number offset

31 – 26	25 – 21	20 – 16	15 – 0
Opcode 0x2D	rs code 0x29	rt code 0x08	16-bit number 00

Format: POP rt, offset(rs) POP $t0, 00($sp);

Purpose:

Read value from memory to destination register "rt".

Description: reg[rt] ←-- reg[offset + rs];

Register "rs" is the base address to calculate the address from which the content can be retrieved from. By adding the offset to the based address, the right address to get the value from.

Decimal 00 == 0x00. Address of $t0 = 0x12004094. The address is 0x12004094.
reg[rt] ←-- reg[0x120040ac];

Restriction:

General purpose registers only to be used in the operation are register 8 to 25. As in hex are register 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF, 0x10, 0x11, 0x12, 0X13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19.

Operation:

reg[rt] ←-- reg[offset + rs];

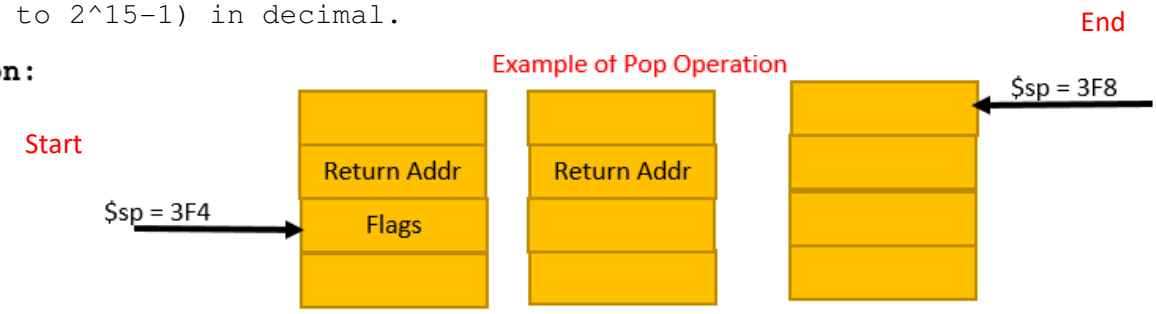
Exception: N/A

Programming Notes:

The 16-bit offset to the base address can have value within the range of -32768 to 32767(-2^15 to 2^15-1) in decimal.

Implementation:

N/A



retflg **write flags back to flag register**

31 – 26	25 – 21	20 – 00
Opcode 0x2E	rs code	Usused bit

31 – 26	25 – 21	00
Opcode 0x2E	rs code 0x08	Unused bit

Format: RETFLG $rs RETFLG $rs;

Purpose:

Output the flags which is the content of register $8 as the output of the ALU.

Description:

retflg output the content of $8 out of the datapath and send the flags signal as the output of the ALU to the control unit.

Restriction:

Only $rt = 8 is used in this instruction, because the pop instruction write the memory content to $8 only

Operation:

retflg;

Exception: N/A

Programming Notes:

The 16-bit offset to the base address can have value within the range of -32768 to 32767(-2^15 to 2^15-1) in decimal.

Implementation:

N/A

retpc **load PC with the return address**

31 – 26	25 – 21	20 – 00
Opcode 0x1F	rs code	Usused bit

31 – 26	25 – 21	00
Opcode 0x1F	rs code 0x08	Unused bit

Format: RETPC $rs RETPC $rs;

Purpose:

Output the return address which is the content of register $8 as the output of the ALU.

Description:

retflg output the content of $8 out of the datapath, the return address to the program counter PC. The instruction unit load in the input PC instead of using the increament PC.

Restriction:

Only $rt = 8 is used in this instruction, because the pop instruction write the memory content to $8 only

Operation:

Retpc $rs;

Exception: N/A

Programming Notes:

The 16-bit offset to the base address can have value within the range of -32768 to 32767(-2^15 to 2^15-1) in decimal.

Implementation:

N/A

III.Verilog Implementation/Design/Verification:

A.Source Code of Top Level:

1.Top Level for MIPS Processor:

```
`timescale 1ns / 1ps
/****************************** C E C S  4 4 0 ******************************
 *
 * File Name:  Lab6_MCU_TopLevel.v
 * Project:    CECS 440 Senior Project Customized MIPS Processor
 * Designer:   Juie Kim
 * Email:      chealy_ljh@yahoo.co.uk
 * Rev. No.:   Version 1.0
 * Rev. Date:  November 28,  2017
 *
 * Purpose:   Top Level containing 5 modules: Instruction Unit, MCU, Intgr DP,
 *            Data Memory and IO Memory modules connect to from one module to
 *            another to allow data flow
 * Notes:      N/A
 *
 ****************************************************************************/
module Lab6_MCU_TopLevel(clk, rst);
      input    clk, rst;  //system inputs

      //internal wire
      wire     c_wire, v_wire, n_wire, z_wire, intr_wire, int_ack_wire;
      wire     [1:0]  Mem_wr_Sel_wire;
      wire     [31:0] se_16_wire, ALU_OUT_wire, intgr_D_OUT_wire, DY_wire;
      wire     [31:0] PC_out_wire, IR_out_wire, dMem_D_out_wire, ioMem_D_out_wire;
      wire     [31:0] pc_mux_out_wire, mem_wr_D_in;
      wire     [4:0]  s_addr_wire, t_addr_wire, d_addr_wire;

      wire     [4:0]    Flags_to_DP_wire;
      wire     [4:0]    FS_wire, shift_amount_wire;
      wire     [2:0]    Y_Sel_wire;
      wire     [1:0]    D_Sel_wire, pc_sel_wire;
      wire              pc_ld_wire, pc_inc_wire, ir_ld_wire;
      wire              im_cs_wire, im_rd_wire, im_wr_wire;
      wire              D_En_wire, S_Sel_wire, T_Sel_wire, HILO_ld_wire, flg_wb_Sel_wire;
      wire              dm_cs_wire, dm_rd_wire, dm_wr_wire;
```

```
      wire              dm_cs_wire, dm_rd_wire, dm_wr_wire;
      wire              io_cs_wire, io_rd_wire, io_wr_wire;

      //Assignmeng for PC_Mux
      assign   pc_mux_out_wire = (pc_sel_wire == 0) ?
                                 {PC_out_wire + {se_16_wire[29:0], 2'b00}}:
                                 (pc_sel_wire == 1) ?
                                 {PC_out_wire[31:28], IR_out_wire[25:0], 2'b00}:
                                 (pc_sel_wire == 2) ? ALU_OUT_wire :  32'h0;

      Intruction_Unit   instrct_unit (
```

```
44
45      Intruction_Unit   instrct_unit (
46                  .clk(clk), .rst(rst),
47                  .pc_ld(pc_ld_wire), .pc_inc(pc_inc_wire),
48                  .im_cs(im_cs_wire), .im_wr(im_wr_wire), .im_rd(im_rd_wire),
49                  .ir_ld(ir_ld_wire),
50                  .PC_in(pc_mux_out_wire), .D_In(), .PC_out(PC_out_wire),
51                  .SE_16(se_16_wire),
52                  .IR_out(IR_out_wire));
53
54      MCU       mcu(
55                .clk(clk), .rst(rst), .intr(intr_wire),
56                .c(c_wire), .n(n_wire), .z(z_wire), .v(v_wire),
57                .IR(IR_out_wire),
58                .int_ack(int_ack_wire),
59                .FS(FS_wire),
60                .SA(shift_amount_wire),
61                .pc_sel(pc_sel_wire), .pc_ld(pc_ld_wire), .pc_inc(pc_inc_wire),
62                .ir_ld(ir_ld_wire),
63                .im_cs(im_cs_wire), .im_rd(im_rd_wire), .im_wr(im_wr_wire),
64                .D_En(D_En_wire), .D_Sel(D_Sel_wire),
65                .S_Sel(S_Sel_wire), .T_Sel(T_Sel_wire),
66                .HILO_ld(HILO_ld_wire), .Y_Sel(Y_Sel_wire),
67                .Mem_wr_Sel(Mem_wr_Sel_wire), .flg_wb_Sel(flg_wb_Sel_wire),
68                .dm_cs(dm_cs_wire), .dm_rd(dm_rd_wire), .dm_wr(dm_wr_wire),
69                .io_cs(io_cs_wire), .io_rd(io_rd_wire), .io_wr(io_wr_wire),
70                .Flags_to_DP(Flags_to_DP_wire));
71
72      //Assignming for D_Addr
73      assign   d_addr_wire = (D_Sel_wire == 0) ? IR_out_wire[15:11] :
74                             (D_Sel_wire == 1) ? IR_out_wire[20:16] :
75                             (D_Sel_wire == 2) ? 5'd31              :
76                             (D_Sel_wire == 3) ? 5'd29              :
77                                                 IR_out_wire[15:11];
78
79      //Assignment for S_Addr and T_Addr
80      assign   s_addr_wire = IR_out_wire[25:21];
81      assign   t_addr_wire = IR_out_wire[20:16];
82
```

```
80        assign   s_addr_wire = IR_out_wire[25:21];
81        assign   t_addr_wire = IR_out_wire[20:16];
82
83        //Selection input for DY between data from I/O mem or dmMem
84        assign   DY_wire = (io_cs_wire == 1)? ioMem_D_out_wire :dMem_D_out_wire;
85
86        Intgr_Path_2      intgr_p2 (
87                    .clk(clk), .reset(rst),
88                    .DT(se_16_wire),
89                    .DY(DY_wire), .DFlag({27'b0, Flags_to_DP_wire}), //Flgs_to_DP_wire = 5'b
90                    .Mem_wr_Sel(Mem_wr_Sel_wire), .flg_wb_Sel(flg_wb_Sel_wire),
91                    .PC_in(PC_out_wire),
92                    .D_En(D_En_wire),
93                    .D_Addr(d_addr_wire),
94                    .S_Addr(s_addr_wire), .S_Sel(S_Sel_wire),
95                    .T_Addr(t_addr_wire),
96                    .T_Sel(T_Sel_wire),
97                    .FS(FS_wire),
98                    .shift_amount(shift_amount_wire),
99                    .HILO_ld(HILO_ld_wire),
100                   .Y_Sel(Y_Sel_wire),
101                   .C(c_wire), .V(v_wire), .N(n_wire), .Z(z_wire),
102                   .ALU_OUT(ALU_OUT_wire),
103                   .D_OUT(intgr_D_OUT_wire));
104
105       Data_Memory       data_mem (
106                   .clk(clk), .rst(rst),
107                   .dm_cs(dm_cs_wire), .dm_wr(dm_wr_wire), .dm_rd(dm_rd_wire),
108                   .Address(ALU_OUT_wire[11:0]),
109                   .D_in(intgr_D_OUT_wire), .D_Out(dMem_D_out_wire));
110
111       IO_Memory         io_mem(
112                   .clk(clk), .rst(rst),
113                   .intr(intr_wire), .int_ack(int_ack_wire),
114                   .io_cs(io_cs_wire), .io_wr(io_wr_wire), .io_rd(io_rd_wire),
115                   .Address_io(ALU_OUT_wire[11:0]),
116                   .D_in_io(intgr_D_OUT_wire), .D_Out_io(ioMem_D_out_wire));
117   endmodule
118
```

B.Source Code of Verilog modules:

2.Instruction Unit:

```
1   `timescale 1ns / 1ps
2   /***************************** C E C S  4 4 0 ****************************
3    *
4    * File Name:  Instruction_Uniut.v
5    * Project:    CECS 440 Senior Project Customized MIPS Processor
6    * Designer:   Juie Kim
7    * Email:      chealy_ljh@yahoo.co.uk
8    * Rev. No.:   Version 1.0
9    * Rev. Date:  November 28,  2017
10   *
11   * Purpose:    Sub module Intruction Unit contains Program Counter register
12   *             Intruction Memory and IR register to hold the instruction.
13   * Notes:      N/A
14   *
15   ***************************************************************************/
16  module Intruction_Unit (clk, rst, pc_ld, pc_inc, im_cs, im_wr, im_rd, ir_ld,
17                          PC_in, D_In, PC_out, SE_16, IR_out);
```

```
module Intruction_Unit (clk, rst, pc_ld, pc_inc, im_cs, im_wr, im_rd, ir_ld,
                        PC_in, D_In, PC_out, SE_16, IR_out);

      input    clk, rst, im_cs, im_wr, im_rd, pc_ld, pc_inc, ir_ld;
      input         [31:0] PC_in, D_In;
      output wire   [31:0] SE_16;
      output wire   [31:0] PC_out, IR_out;

      //internal wires
      wire     [11:0]   Address_wire;
      wire     [31:0]   pc_out_wire, D_Out_wire, qOut_ir_wire;

      PC_Program_Counter   pc_cnter(
                  .clk(clk),
                  .rst(rst),
                  .pc_ld(pc_ld),
                  .pc_inc(pc_inc),
                  .pc_in(PC_in),
                  .pc_out(pc_out_wire));

      //assign lower 12 bit of the pc_out as address to instrct mem
      assign   Address_wire = pc_out_wire[11:0];
      assign   PC_out       = pc_out_wire;

      Instruction_Memory   instrct_mem(
                  .clk(clk),
                  .rst(rst),
                  .im_cs(im_cs),
                  .im_wr(im_wr),
                  .im_rd(im_rd),
                  .Address(Address_wire),
                  .D_in(D_In),
                  .D_Out(D_Out_wire));

      IR_Register    ir_reg(
                  .clk(clk),
                  .rst(rst),
                  .load_ir(ir_ld),
                  .din_ir(D_Out_wire),
                  .din_ir(D_Out_wire),
                  .qOut_ir(qOut_ir_wire));

       assign   IR_out = qOut_ir_wire;

       //Sign_Extend 16 function
       assign   SE_16  = {{16{qOut_ir_wire[15]}}, qOut_ir_wire[15:0]};

endmodule
```

3.Program Counter:

```
1  `timescale 1ns / 1ps
2  /****************************** C E C S  4 4 0 ******************************
3   *
4   * File Name:  PC_Program_Counter.v
5   * Project:    CECS 440 Senior Project Customized MIPS Processor
6   * Designer:   Juie Kim
7   * Email:      chealy_ljh@yahoo.co.uk
8   * Rev. No.:   Version 1.0
9   * Rev. Date:  November 5, 2017
10  *
11  * Purpose:    Sub module loadable register to, 32 bit to interface with the
12  *             Instruction memory module. It has the control to increament
13  *             the output by 4 if needed.
14  * Notes:      N/A
15  *
16  ******************************************************************************/
17 module PC_Program_Counter(clk, rst, pc_ld, pc_inc, pc_in, pc_out);
18
19    input    clk, rst;
20    input    pc_ld, pc_inc;
21    input    [31:0]   pc_in;
22    output   [31:0]   pc_out;
23
24    reg      [31:0]   pc_out;
25
26    always @(posedge clk, posedge rst)
27       if (rst)
28          pc_out <=   32'h0;
29       else
30          case ({pc_ld, pc_inc})
31             2'b00: begin
32                pc_out = pc_out;
33             end
34             2'b01: begin
35                pc_out = pc_out + 4;
36             end
37             2'b10: begin
38                pc_out = pc_in;
39             end
```

```
38                pc_out = pc_in;
39             end
40             2'b11: begin
41                pc_out = pc_out;
42             end
43       endcase  //end case statement
44 endmodule
45
```

4. Instruction Memory

```
/****************************** C E C S  4 4 0 ******************************
 *
 * File Name:  Instruction_Memory.v
 * Project:    CECS 440 Senior Project Customized MIPS Processor
 * Designer:   Juie Kim
 * Email:      chealy_ljh@yahoo.co.uk
 * Rev. No.:   Version 1.0
 * Rev. Date:  November 5, 2017
 *
 * Purpose:    Sub module memory module of a 4096x8 byte addressable memory
 *             big endian format which has chip select, read and write
 *             control.
 * Notes:      N/A
 *
 ******************************************************************************/
module Instruction_Memory(clk, rst, im_cs, im_wr, im_rd, Address, D_in, D_Out);
      input    clk, rst, im_cs, im_wr, im_rd;
      input    [11:0] Address;
      input    [31:0] D_in;
      output   [31:0] D_Out;

      reg      [7:0]  ir_Mem [0:4096]; //1K memory byte addressable

      //writing to ir_Memory is synchronous
      always @(posedge clk, posedge rst)
         if(rst)
            ir_Mem[Address] <= ir_Mem[Address];

         else if (im_cs==1 && im_wr==1) begin
            ir_Mem[Address + 0] <= D_in[31:24];
            ir_Mem[Address + 1] <= D_in[23:16];
            ir_Mem[Address + 2] <= D_in[15:8];
            ir_Mem[Address + 3] <= D_in[7:0];
            end

         else
            ir_Mem[Address[11:0]] <= ir_Mem[Address[11:0]];

```


```
         else
            ir_Mem[Address[11:0]] <= ir_Mem[Address[11:0]];
```


```

      //reading from jl_Mem is asynchronous
      assign D_Out = (im_cs==1 && im_rd==1) ?
                     {ir_Mem[Address + 0], ir_Mem[Address + 1],
                      ir_Mem[Address + 2], ir_Mem[Address + 3]} : D_Out;
endmodule

```

5.IR Register:

```
`timescale 1ns / 1ps
/****************************** C E C S  4 4 0 ******************************
 *
 * File Name:  IR_Intrct_Mem_Register.v
 * Project:    CECS 440 Senior Project Customized MIPS Processor
 * Designer:   Juie Kim
 * Email:      chealy_ljh@yahoo.co.uk
 * Rev. No.:   Version 1.0
 * Rev. Date:  November 5, 2017
 *
 * Purpose:    Sub module loadable register, 32 bit to interface with the
 *             Instruction memory module. It has the loadable control to
 *             load intruction 32 bit from the Instruction Memory module
 * Notes:      N/A
 *
 ******************************************************************************/
module IR_Register(clk, rst, load_ir, din_ir, qOut_ir);

      input    clk, rst, load_ir;
      input    [31:0]   din_ir;
      output   [31:0]   qOut_ir;

      reg      [31:0]   qOut_ir;

      always @(posedge clk, posedge rst)
         if (rst)
            qOut_ir <= 32'h0;
         else if (load_ir)
            qOut_ir <= din_ir;
endmodule
```

6.Interger Datapath:

```
`timescale 1ns / 1ps
/****************************** C E C S  4 4 0 ******************************
 *
 * File Name:  Intgr_Path_2.v
 * Project:    CECS 440 Senior Project Customized MIPS Processor
 * Designer:   Juie Kim
 * Email:      chealy_ljh@yahoo.co.uk
 * Rev. No.:   Version 1.0
 * Rev. Date:  Novembee 28,  2017
 *
 * Purpose:    Top level to tight ALU, two loadagle registers, and four
 *             register used as pipelining, and output result throught
 *             the ALU_OUT
 * Notes:      N/A.
 *
 ******************************************************************************/
module Intgr_Path_2(clk, reset, DT, DY, DFlag, Mem_wr_Sel, flg_wb_Sel,
                        PC_in, D_En, D_Addr, S_Addr, S_Sel,
                        T_Addr, T_Sel, FS, shift_amount, HILO_ld, Y_Sel,
                        C, V, N, Z,  ALU_OUT, D_OUT);

```

```
21
22      input                   clk, reset, D_En, T_Sel, S_Sel, HILO_ld, flg_wb_Sel;
23      input          [1:0]    Mem_wr_Sel;
24      input          [2:0]    Y_Sel;
25      input          [4:0]    D_Addr, S_Addr, T_Addr, FS, shift_amount;
26      input          [31:0]   DY, DT, PC_in, DFlag;
27      output wire             C, V, N, Z;
28      output wire    [31:0]   ALU_OUT, D_OUT;
29
30      //declaring internal wires
31      wire           [31:0]   S_output, T_output, Y_hi, Y_lo, HI_out, LO_out;
32      wire                    load_hi, load_lo;
33
34      //internal wires for four pipelining registers
35      wire           C_wire, V_wire, N_wire, Z_wire;
36      wire           [4:0]    S_Addr_wire;
37      wire           [31:0]   RS_output, RT_output, ALU_Out_output, D_in_output;
38      wire           [31:0]   T_MUX_output;
39
40      //selection to load into HI and LO registers
41      assign load_hi  = (HILO_ld)? 1:0;
42      assign load_lo  = (HILO_ld)? 1:0;
43
44      //selection for T_MUX and S_Addr
45      assign T_MUX_output = (T_Sel == 1)? DT    : T_output;
46      assign S_Addr_wire  = (S_Sel == 1)? 5'd29 : S_Addr;
47
48      //Assign Flags input from D_in to each flag output when writing back to
49      //Flags register in the MCU
50      assign {C, V, N, Z} = (flg_wb_Sel == 1) ?
51                            {ALU_OUT[3], ALU_OUT[2], ALU_OUT[1], ALU_OUT[0]} :
52                            {C_wire, V_wire, N_wire, Z_wire};
53
54      RS_register    rs_rg(
55                        .clk(clk),
56                        .rst(reset),
57                        .din(S_output),
58                        .qOut(RS_output));
59
```

```
59
60       RT_register    rt_rg(
61                          .clk(clk),
62                          .rst(reset),
63                          .din(T_MUX_output),
64                          .qOut(RT_output));
65
66       ALU_OUT_reg    alu_out_rg(
67                          .clk(clk),
68                          .rst(reset),
69                          .din(Y_lo),
70                          .qOut(ALU_Out_output));
```

```
                        .qOut(ALU_Out_output));

    D_in_register   d_in_rg(
                        .clk(clk),
                        .rst(reset),
                        .din(DY),
                        .qOut(D_in_output));

    regfile32       reg_file(
                        .clk(clk),
                        .rst(reset),
                        .D(ALU_OUT),
                        .S_Addr(S_Addr_wire),
                        .T_Addr(T_Addr),
                        .D_Addr(D_Addr),
                        .D_En(D_En),
                        .S(S_output),
                        .T(T_output));

    ALU_32bit       ALU_32bit(
                        .Opcode(FS),
                        .shift_amount(shift_amount),
                        .S(RS_output),
                        .T(RT_output),
                        .y_hi(Y_hi),
                        .y_lo(Y_lo),
                        .flg_c(C_wire),
                        .flg_v(V_wire),
                        .flg_n(N_wire),
                        .flg_z(Z_wire));

    HI_Loadable_Register  HI_ld_reg(
                        .clk(clk),
                        .rst(reset),
                        .load_hi(load_hi),
                        .din_hi(Y_hi),
                        .qOut_hi(HI_out));

    LO_Loadable_Register  LO_ld_reg(
                        .clk(clk),
                        .rst(reset),
                        .load_lo(load_lo),
                        .din_lo(Y_lo),
                        .qOut_lo(LO_out));

    //selection for Y_MUX
    assign ALU_OUT =  (Y_Sel == 0) ? ALU_Out_output  :
                      (Y_Sel == 1) ? HI_out          :
                      (Y_Sel == 2) ? LO_out          :
                      (Y_Sel == 3) ? D_in_output     :
                      (Y_Sel == 4) ? PC_in  :
                      (Y_Sel == 5) ? DFlag : ALU_Out_output;

    assign D_OUT   =  (Mem_wr_Sel == 1) ? DFlag :
                      (Mem_wr_Sel == 2) ? PC_in :
                      (Mem_wr_Sel == 0) ? RT_output : RT_output;

endmodule
```

7. RS_Registger

```
`timescale 1ns / 1ps
/****************************** C E C S  4 4 0 ******************************
 *
 * File Name:  RS_register.v
 * Project:    CECS 440 Senior Project Customized MIPS Processor
 * Designer:   Juie Kim
 * Email:      chealy_ljh@yahoo.co.uk
 * Rev. No.:   Version 1.0
 * Rev. Date:  November 28,  2017
 *
 * Purpose:    Sub module to store data fromt he S_Addr register and as an
 *             input to ALU as to do operation needed
 * Notes:      N/A.
 *
 ******************************************************************************/
module RS_register(clk, rst, din, qOut);

   input    clk, rst;
   input    [31:0]   din;
   output   [31:0]   qOut;

   reg      [31:0]   qOut;

   always @(posedge clk, posedge rst)
      if (rst)
         qOut <=  32'h0;
      else
         qOut <=  din;
endmodule
```

8.RT_Register:

```
`timescale 1ns / 1ps
/****************************** C E C S  4 4 0 ******************************
 *
 * File Name:  RT_register.v
 * Project:   CECS 440 Senior Project Customized MIPS Processor
 * Designer:   Juie Kim
 * Email:      chealy_ljh@yahoo.co.uk
 * Rev. No.:   Version 1.0
 * Rev. Date:  November 28,  2017
 *
 * Purpose:    Sub module to store data fromt he T_Addr register and as an
 *             input to ALU as to do operation needed
 * Notes:      N/A.
 *
 ******************************************************************************/
module RT_register(clk, rst, din, qOut);

   input    clk, rst;
   input    [31:0]   din;
   output   [31:0]   qOut;

   reg      [31:0]   qOut;

   always @(posedge clk, posedge rst)
      if (rst)
         qOut <=  32'h0;
      else
         qOut <=  din;
endmodule
```

9.ALU_Out Register:

```
`timescale 1ns / 1ps
/****************************** C E C S  4 4 0 ******************************
 *
 * File Name:  ALU_OUT_reg.v
 * Project:    CECS 440 Senior Project Customized MIPS Processor
 * Designer:   Juie Kim
 * Email:      chealy_ljh@yahoo.co.uk
 * Rev. No.:   Version 1.0
 * Rev. Date:  Novembr 28,  2017
 *
 * Purpose:    Sub module to store data from the ALU operation output from
 *             y_lo port and to be output to the ALU_OUT
 * Notes:      N/A.
 *
 ******************************************************************************/
module ALU_OUT_reg(clk, rst, din, qOut);

   input    clk, rst;
   input    [31:0]   din;
   output   [31:0]   qOut;

   reg      [31:0]   qOut;

   always @(posedge clk, posedge rst)
      if (rst)
         qOut <=  32'h0;
      else
         qOut <=  din;
endmodule
```

10. D_in Register:

```
`timescale 1ns / 1ps
/****************************** C E C S  4 4 0 ******************************
 *
 * File Name:  D_in.v
 * Project:    Lab_Assignment_6
 * Designer:   Juie Kim
 * Email:      chealy_ljh@yahoo.co.uk
 * Rev. No.:   Version 1.0
 * Rev. Date:  Novemger 5, 2017
 *
 * Purpose:    Sub module to store the low 32-bit input and output D_in
 *             register for the purpose of output to ALU_OUT
 * Notes:      N/A.
 *
 ******************************************************************************/
module D_in_register(clk, rst, din, qOut);

   input    clk, rst;
   input    [31:0]   din;
   output   [31:0]   qOut;

   reg      [31:0]   qOut;

   always @(posedge clk, posedge rst)
      if (rst)
         qOut <=  32'h0;
      else
         qOut <=  din;
endmodule
```

11.Register File:

```
`timescale 1ns / 1ps
/****************************** C E C S  4 4 0 ******************************
 *
 * File Name:  regfile32.v
 * Project:    CECS 440 Senior Project Customized MIPS Processor
 * Designer:   Juie Kim
 * Email:      chealy_ljh@yahoo.co.uk
 * Rev. No.:   Version 1.0
 * Rev. Date:  Novemger 28, 2017
 *
 * Purpose:    Designing a dual port 32 x 32 register file. The
 *             register file is used to read data from and write
 *             data to
 * Notes:      N/A
 *
 ******************************************************************************/
module regfile32(clk, rst, D, S_Addr, T_Addr, D_Addr, D_En, S, T);
   input              clk, rst, D_En;
   input       [31:0] D;
   input       [4:0]  S_Addr, T_Addr, D_Addr;
   output      [31:0] S, T;

   reg         [31:0] reg_mem [0:31];    //32 register each 32 bit long

   always @(posedge clk, posedge rst)
      if (rst)
         reg_mem[0] <= 32'h0;
      else if (D_En == 1 && D_Addr != 0)
         reg_mem [D_Addr] <= D;
      else
         reg_mem[D_Addr] = reg_mem[D_Addr];

   assign      S  = reg_mem[S_Addr];
   assign      T  = reg_mem[T_Addr];
endmodule
```

12.ALU_32 Module:

```
`timescale 1ns / 1ps
/****************************** C E C S  4 4 0 ******************************
 *
 * File Name:  ALU_32bit.v
 * Project:    CECS 440 Senior Project
 * Designer:   Juie Kim
 * Email:      chealy_ljh@yahoo.co.uk
 * Rev. No.:   Version 1.0
 * Rev. Date:  November 28,  2017
 *
 * Purpose:    Top level of three sub module, designed to output the result
 *             and the four flags of c, v, n, and z.
 *
 * Notes:      N/A
 *
 ****************************************************************************/
module ALU_32bit(Opcode, shift_amount, S, T, y_hi, y_lo, flg_c,
                 flg_v, flg_n, flg_z);

   input        [4:0]   Opcode, shift_amount;
   input        [31:0]  S;
   input        [31:0]  T;
   output wire  [31:0]  y_hi;
   output wire  [31:0]  y_lo;
   output wire          flg_c, flg_v, flg_n, flg_z;

   //declaring internal wires
   wire   mlt_n, mlt_z, div_n, div_z, mip_c, mip_v, mip_n, mip_z;
   wire   bar_c, bar_v;
   wire   [63:32]  remndr,  mlt_hi;
   wire   [31:0]   quotn, mlt_lo, mip_lo, bar_lo;

   //Instantiate the divide multiply and mip modules
   BARRELSHIFT_32    barrel_shift_32(
            .Opcode(Opcode),
            .Shmt(shift_amount),
            .SData(S), .TData(T), .y_lo(bar_lo),
            .c(bar_c), .v(bar_v));

```

```
    MPY_32  MPY_32(
             .Opcode(Opcode),
             .SData(S), .TData(T),
             .y_hi(mlt_hi), .y_lo(mlt_lo));

    DIV_32  DIV_32(
             .Opcode(Opcode),
             .SData(S), .TData (T),
             .y_hi(remndr), .y_lo(quotn));

    MIP_32  MIP_32(
             .Opcode(Opcode),
             .SData(S), .TData(T), .y_lo(mip_lo),
             .c (mip_c), .v(mip_v));
```

```
            .SData(S), .TData(T), .y_lo(mip_lo),
            .c (mip_c), .v(mip_v));

    //equate the signal out of the multiply and divide module
    //and set the negative and zero flag the the internal wire
    //of each module of multiply and divide accordingly
    assign mlt_n        = (Opcode == 5'h1E)? mlt_hi[63]:1'bx;
    assign div_n        = (Opcode == 5'h1F)? quotn[31] :1'bx;
    assign mlt_z        = (Opcode == 5'h1E)? ~(|{mlt_hi,mlt_lo}):1'b1;
    assign div_z        = (Opcode == 5'h1F)? ~(|{remndr, quotn}):1'b1;

    //equate the output of the multiply and divide to the actual
    //output of the top level, y_hi and y_lo
    assign {y_hi, y_lo} = (Opcode == 5'h1E)     ? {mlt_hi, mlt_lo}:
                          (Opcode == 5'h1F)     ? {remndr, quotn} :
                         ((Opcode == 5'h0C)||
                          (Opcode == 5'h0D)||
                         (Opcode == 5'h0E))     ? {32'b0, bar_lo} : {32'b0, mip_lo};

    //equate the all four flags from the internal wires
    //to the actual output of the top level, flg_c, flg_v, flg_n, flg_z
    assign flg_c  =  ((Opcode == 5'h0C) || (Opcode == 5'h0D)
                                        || (Opcode == 5'h0E)) ? bar_c : mip_c;

    assign flg_v  =  ((Opcode == 5'h0C) || (Opcode == 5'h0D)
                                        || (Opcode == 5'h0E)) ? bar_v : mip_v;

    assign flg_n  = (Opcode == 5'h1E) ? mlt_n :
                    (Opcode == 5'h1F) ? div_n :
                 ((Opcode == 5'h0C)||
                  (Opcode == 5'h0D)||
                   (Opcode == 5'h0E)) ? bar_lo[31] : mip_lo[31];

    assign flg_z  = (Opcode == 5'h1E) ? mlt_z :
                    (Opcode == 5'h1F) ? div_z :
                 ((Opcode == 5'h0C)||
                  (Opcode == 5'h0D)||
                   (Opcode == 5'h0E)) ? ~(|bar_lo) : ~(|mip_lo);
endmodule
```

13. Barrel Shift Module:

```
`timescale 1ns / 1ps
/****************************** C E C S  4 4 0 ******************************
 *
 * File Name:   BARRELSHIFT_32.v
 * Project:     CECS 440 Senior Project Customized MIPS Processor
 * Designer:    Juie Kim
 * Email:       chealy_ljh@yahoo.co.uk
 * Rev. No.:    Version 1.0
 * Rev. Date:   November 28, 2017
 *
 * Purpose:     Sub module designed to calculate the multiplication and
 *              assign the 64 bit output to proper output data bus
 *
 * Notes:       N/A
 *
 ****************************************************************************/
module BARRELSHIFT_32(Opcode, Shmt, SData, TData, y_lo, c, v);

   input     [4:0]    Opcode, Shmt;
   input     [31:0]   SData, TData;

   output    [31:0]   y_lo;
   output             c, v;
   reg       [31:0]   y_lo;
   reg                c, v;
```

```
27      always @(*) begin
28         c = 0;
29         v = 0;
30         case({Opcode, Shmt})
31      /*****************************************
32      *BARREL SHIFT LEFT LOGICAL 32
33      *****************************************/
34            {5'h0C, 5'd1}:   {c, y_lo} = {TData[31], {TData[30:0], 1'b0}};          //SLL_1
35            {5'h0C, 5'd2}:   {c, y_lo} = {TData[30], {TData[29:0], 2'b0}};          //SLL_2
36            {5'h0C, 5'd3}:   {c, y_lo} = {TData[29], {TData[28:0], 3'b0}};          //SLL_3
37            {5'h0C, 5'd4}:   {c, y_lo} = {TData[28], {TData[27:0], 4'b0}};          //SLL_4
38            {5'h0C, 5'd5}:   {c, y_lo} = {TData[27], {TData[26:0], 5'b0}};          //SLL_5
39            {5'h0C, 5'd6}:   {c, y_lo} = {TData[26], {TData[25:0], 6'b0}};          //SLL_6
40            {5'h0C, 5'd7}:   {c, y_lo} = {TData[25], {TData[24:0], 7'b0}};          //SLL_7
41            {5'h0C, 5'd8}:   {c, y_lo} = {TData[24], {TData[23:0], 8'b0}};          //SLL_8
42            {5'h0C, 5'd9}:   {c, y_lo} = {TData[23], {TData[22:0], 9'b0}};          //SLL_9
43            {5'h0C, 5'd10}:  {c, y_lo} = {TData[22], {TData[21:0], 10'b0}};         //SLL_10
44            {5'h0C, 5'd11}:  {c, y_lo} = {TData[21], {TData[20:0], 11'b0}};         //SLL_11
45            {5'h0C, 5'd12}:  {c, y_lo} = {TData[20], {TData[19:0], 12'b0}};         //SLL_12
46            {5'h0C, 5'd13}:  {c, y_lo} = {TData[19], {TData[18:0], 13'b0}};         //SLL_13
47            {5'h0C, 5'd14}:  {c, y_lo} = {TData[18], {TData[17:0], 14'b0}};         //SLL_14
48            {5'h0C, 5'd15}:  {c, y_lo} = {TData[17], {TData[16:0], 15'b0}};         //SLL_15
49            {5'h0C, 5'd16}:  {c, y_lo} = {TData[16], {TData[15:0], 16'b0}};         //SLL_16
50            {5'h0C, 5'd17}:  {c, y_lo} = {TData[15], {TData[14:0], 17'b0}};         //SLL_17
51            {5'h0C, 5'd18}:  {c, y_lo} = {TData[14], {TData[13:0], 18'b0}};         //SLL_18
52            {5'h0C, 5'd19}:  {c, y_lo} = {TData[13], {TData[12:0], 19'b0}};         //SLL_19
53            {5'h0C, 5'd20}:  {c, y_lo} = {TData[12], {TData[11:0], 20'b0}};         //SLL_20
54            {5'h0C, 5'd21}:  {c, y_lo} = {TData[11], {TData[10:0], 21'b0}};         //SLL_21
55            {5'h0C, 5'd22}:  {c, y_lo} = {TData[10], {TData[9:0],  22'b0}};         //SLL_22
56            {5'h0C, 5'd23}:  {c, y_lo} = {TData[9],  {TData[8:0],  23'b0}};         //SLL_23
57            {5'h0C, 5'd24}:  {c, y_lo} = {TData[8],  {TData[7:0],  24'b0}};         //SLL_24
58            {5'h0C, 5'd25}:  {c, y_lo} = {TData[7],  {TData[6:0],  25'b0}};         //SLL_25
59            {5'h0C, 5'd26}:  {c, y_lo} = {TData[6],  {TData[5:0],  26'b0}};         //SLL_26
60            {5'h0C, 5'd27}:  {c, y_lo} = {TData[5],  {TData[4:0],  27'b0}};         //SLL_27
61            {5'h0C, 5'd28}:  {c, y_lo} = {TData[4],  {TData[3:0],  28'b0}};         //SLL_28
62            {5'h0C, 5'd29}:  {c, y_lo} = {TData[3],  {TData[2:0],  29'b0}};         //SLL_29
63            {5'h0C, 5'd30}:  {c, y_lo} = {TData[2],  {TData[1:0],  30'b0}};         //SLL_30
64            {5'h0C, 5'd31}:  {c, y_lo} = {TData[1],  {TData[0],    31'b0}};         //SLL_31
```

```
67      /*****************************************
68      *BARREL SHIFT RIGHT LOGICAL 32 bit
69      *****************************************/
70            {5'h0D, 5'd1}:   {c, y_lo} = {TData[0],  {1'b0,  TData[31:1]}};          //SRL_1
71            {5'h0D, 5'd2}:   {c, y_lo} = {TData[1],  {2'b0,  TData[31:2]}};          //SRL_2
72            {5'h0D, 5'd3}:   {c, y_lo} = {TData[2],  {3'b0,  TData[31:3]}};          //SRL_3
73            {5'h0D, 5'd4}:   {c, y_lo} = {TData[3],  {4'b0,  TData[31:4]}};          //SRL_4
74            {5'h0D, 5'd5}:   {c, y_lo} = {TData[4],  {5'b0,  TData[31:5]}};          //SRL_5
75            {5'h0D, 5'd6}:   {c, y_lo} = {TData[5],  {6'b0,  TData[31:6]}};          //SRL_6
76            {5'h0D, 5'd7}:   {c, y_lo} = {TData[6],  {7'b0,  TData[31:7]}};          //SRL_7
77            {5'h0D, 5'd8}:   {c, y_lo} = {TData[7],  {8'b0,  TData[31:8]}};          //SRL_8
78            {5'h0D, 5'd9}:   {c, y_lo} = {TData[8],  {9'b0,  TData[31:9]}};          //SRL_9
79            {5'h0D, 5'd10}:  {c, y_lo} = {TData[9],  {10'b0, TData[31:10]}};         //SRL_10
80            {5'h0D, 5'd11}:  {c, y_lo} = {TData[10], {11'b0, TData[31:11]}};         //SRL_11
81            {5'h0D, 5'd12}:  {c, y_lo} = {TData[11], {12'b0, TData[31:12]}};         //SRL_12
82            {5'h0D, 5'd13}:  {c, y_lo} = {TData[12], {13'b0, TData[31:13]}};         //SRL_13
83            {5'h0D, 5'd14}:  {c, y_lo} = {TData[13], {14'b0, TData[31:14]}};         //SRL_14
84            {5'h0D, 5'd15}:  {c, y_lo} = {TData[14], {15'b0, TData[31:15]}};         //SRL_15
85            {5'h0D, 5'd16}:  {c, y_lo} = {TData[15], {16'b0, TData[31:16]}};         //SRL_16
86            {5'h0D, 5'd17}:  {c, y_lo} = {TData[16], {17'b0, TData[31:17]}};         //SRL_17
87            {5'h0D, 5'd18}:  {c, y_lo} = {TData[17], {18'b0, TData[31:18]}};         //SRL_18
88            {5'h0D, 5'd19}:  {c, y_lo} = {TData[18], {19'b0, TData[31:19]}};         //SRL_19
89            {5'h0D, 5'd20}:  {c, y_lo} = {TData[19], {20'b0, TData[31:20]}};         //SRL_20
90            {5'h0D, 5'd21}:  {c, y_lo} = {TData[20], {21'b0, TData[31:21]}};         //SRL_21
91            {5'h0D, 5'd22}:  {c, y_lo} = {TData[21], {22'b0, TData[31:22]}};         //SRL_22
92            {5'h0D, 5'd23}:  {c, y_lo} = {TData[22], {23'b0, TData[31:23]}};         //SRL_23
93            {5'h0D, 5'd24}:  {c, y_lo} = {TData[23], {24'b0, TData[31:24]}};         //SRL_24
94            {5'h0D, 5'd25}:  {c, y_lo} = {TData[24], {25'b0, TData[31:25]}};         //SRL_25
95            {5'h0D, 5'd26}:  {c, y_lo} = {TData[25], {26'b0, TData[31:26]}};         //SRL_26
96            {5'h0D, 5'd27}:  {c, y_lo} = {TData[26], {27'b0, TData[31:27]}};         //SRL_27
97            {5'h0D, 5'd28}:  {c, y_lo} = {TData[27], {28'b0, TData[31:28]}};         //SRL_28
98            {5'h0D, 5'd29}:  {c, y_lo} = {TData[28], {29'b0, TData[31:29]}};         //SRL_29
99            {5'h0D, 5'd30}:  {c, y_lo} = {TData[29], {30'b0, TData[31:30]}};         //SRL_30
100           {5'h0D, 5'd31}:  {c, y_lo} = {TData[30], {31'b0, TData[31]}};            //SRL_31
101
```

```
101
102     /*****************************************
103     *BARREL SHIFT RIGHT ARITHMATIC 32
104     *****************************************/
105           {5'h0E, 5'd1}:  {c, y_lo} = {TData[0],  { TData[31],     TData[31:1]}};     //SRA_1
106           {5'h0E, 5'd2}:  {c, y_lo} = {TData[1],  {{2{TData[31]}},  TData[31:2]}};    //SRA_2
107           {5'h0E, 5'd3}:  {c, y_lo} = {TData[2],  {{3{TData[31]}},  TData[31:3]}};    //SRA_3
108           {5'h0E, 5'd4}:  {c, y_lo} = {TData[3],  {{4{TData[31]}},  TData[31:4]}};    //SRA_4
109           {5'h0E, 5'd5}:  {c, y_lo} = {TData[4],  {{5{TData[31]}},  TData[31:5]}};    //SRA_5
110           {5'h0E, 5'd6}:  {c, y_lo} = {TData[5],  {{6{TData[31]}},  TData[31:6]}};    //SRA_6
111           {5'h0E, 5'd7}:  {c, y_lo} = {TData[6],  {{7{TData[31]}},  TData[31:7]}};    //SRA_7
112           {5'h0E, 5'd8}:  {c, y_lo} = {TData[7],  {{8{TData[31]}},  TData[31:8]}};    //SRA_8
113           {5'h0E, 5'd9}:  {c, y_lo} = {TData[8],  {{9{TData[31]}},  TData[31:9]}};    //SRA_9
114           {5'h0E, 5'd10}: {c, y_lo} = {TData[9],  {{10{TData[31]}}, TData[31:10]}};   //SRA_10
115           {5'h0E, 5'd11}: {c, y_lo} = {TData[10], {{11{TData[31]}}, TData[31:11]}};   //SRA_11
116           {5'h0E, 5'd12}: {c, y_lo} = {TData[11], {{12{TData[31]}}, TData[31:12]}};   //SRA_12
117           {5'h0E, 5'd13}: {c, y_lo} = {TData[12], {{13{TData[31]}}, TData[31:13]}};   //SRA_13
118           {5'h0E, 5'd14}: {c, y_lo} = {TData[13], {{14{TData[31]}}, TData[31:14]}};   //SRA_14
119           {5'h0E, 5'd15}: {c, y_lo} = {TData[14], {{15{TData[31]}}, TData[31:15]}};   //SRA_15
120           {5'h0E, 5'd16}: {c, y_lo} = {TData[15], {{16{TData[31]}}, TData[31:16]}};   //SRA_16
121           {5'h0E, 5'd17}: {c, y_lo} = {TData[16], {{17{TData[31]}}, TData[31:17]}};   //SRA_17
122           {5'h0E, 5'd18}: {c, y_lo} = {TData[17], {{18{TData[31]}}, TData[31:18]}};   //SRA_18
123           {5'h0E, 5'd19}: {c, y_lo} = {TData[18], {{19{TData[31]}}, TData[31:19]}};   //SRA_19
124           {5'h0E, 5'd20}: {c, y_lo} = {TData[19], {{20{TData[31]}}, TData[31:20]}};   //SRA_20
125           {5'h0E, 5'd21}: {c, y_lo} = {TData[20], {{21{TData[31]}}, TData[31:21]}};   //SRA_21
126           {5'h0E, 5'd22}: {c, y_lo} = {TData[21], {{22{TData[31]}}, TData[31:22]}};   //SRA_22
127           {5'h0E, 5'd23}: {c, y_lo} = {TData[22], {{23{TData[31]}}, TData[31:23]}};   //SRA_23
128           {5'h0E, 5'd24}: {c, y_lo} = {TData[23], {{24{TData[31]}}, TData[31:24]}};   //SRA_24
129           {5'h0E, 5'd25}: {c, y_lo} = {TData[24], {{25{TData[31]}}, TData[31:25]}};   //SRA_25
130           {5'h0E, 5'd26}: {c, y_lo} = {TData[25], {{26{TData[31]}}, TData[31:26]}};   //SRA_26
131           {5'h0E, 5'd27}: {c, y_lo} = {TData[26], {{27{TData[31]}}, TData[31:27]}};   //SRA_27
132           {5'h0E, 5'd28}: {c, y_lo} = {TData[27], {{28{TData[31]}}, TData[31:28]}};   //SRA_28
133           {5'h0E, 5'd29}: {c, y_lo} = {TData[28], {{29{TData[31]}}, TData[31:29]}};   //SRA_29
134           {5'h0E, 5'd30}: {c, y_lo} = {TData[29], {{30{TData[31]}}, TData[31:30]}};   //SRA_30
135           {5'h0E, 5'd31}: {c, y_lo} = {TData[30], {{31{TData[31]}}, TData[31]}};      //SRA_31
136           endcase
137     end
138   endmodule
```

14.MPY_32 Module:

```
1   `timescale 1ns / 1ps
2   /******************************** C E C S  4 4 0 ******************************
3    *
4    * File Name:   MPY_32.v
5    * Project:     CECS 440 Senior Project Customized MIPS Processor
6    * Designer:    Juie Kim
7    * Email:       chealy_ljh@yahoo.co.uk
8    * Rev. No.:    Version 1.0
9    * Rev. Date:   November 28,  2017
10   *
11   * Purpose:     Sub module designed to calculate the multiplication and
12   *               assign the 64 bit output to proper output data bus
13   *
14   * Notes:       N/A
15   *
16   ******************************************************************************/
17  module MPY_32(Opcode, SData, TData, y_hi, y_lo);
18     input         [31:0] SData;
19     input         [31:0] TData;
20     input          [4:0] Opcode;
21     output reg   [63:32] y_hi;
22     output reg    [31:0] y_lo;
23
24     //declare internal wire to temperary hold the 64_bit output
25     wire          [63:0] result_64;
26
27     //declare temperary integer variable
28     integer       tempA = 0;
29     integer       tempB = 0;
30
31     //transfer input data to the integer variable
32     always @(*)
33        begin
34           tempA = SData;
35           tempB = TData;
36        end
37
38     assign result_64 = tempA * tempB;
```

```
    assign result_64 = tempA * tempB;

    //assign the 64_bit result to proper bit
    always @ (*)
    begin
        case (Opcode)
        5'h1E: begin
        y_hi        = result_64[63:32];
        y_lo        = result_64[31:0];
        end
        endcase
    end
endmodule
```

15. DIV Module:

```
`timescale 1ns / 1ps
/****************************** C E C S  4 4 0 ******************************
 *
 * File Name:  DIV_32.v
 * Project:    CECS 440 Senior Project Customized MIPS Processor
 * Designer:   Juie Kim
 * Email:      chealy_ljh@yahoo.co.uk
 * Rev. No.:   Version 1.0
 * Rev. Date:  November 28,  2017
 *
 * Purpose:    Sub module to calculate the divsion and assign the
 *             output to a proper bits
 *
 * Notes:      N/A
 *
 ******************************************************************************/
module DIV_32(Opcode, SData, TData, y_hi, y_lo);
   input       [31:0] SData;
   input       [31:0] TData;
   input        [4:0] Opcode;
   output reg [63:32] y_hi;
   output reg  [31:0] y_lo;

   //Declare temporay integer variable
   integer     tempA, tempB;

   //save both input in the integer type variable
   always @(*)
      begin
         tempA = SData;
         tempB = TData;
      end

   //operating division and assign to proper bits
   always @ (*)
   begin
      case (Opcode)
      5'h1F: begin
      y_hi       = tempA % tempB;
```

```
      begin
         case (Opcode)
         5'h1F: begin
         y_hi        = tempA % tempB;
         y_lo        = tempA / tempB;
         end
         endcase
      end
   endmodule
```

16. MIP_32 Module:

```
`timescale 1ns / 1ps
/****************************** C E C S  4 4 0 ******************************
 *
 * File Name:  MIP_32.v
 * Project:    CECS 440 Senior Project Customized MIPS Processor
 * Designer:   Juie Kim
 * Email:      chealy_ljh@yahoo.co.uk
 * Rev. No.:   Version 1.0
 * Rev. Date:  November 28, 2017
 *
 * Purpose:    Sub module designed to operate unary operand and two
 *             operand. This a standard mip architecture operations
 *
 * Notes:      N/A
 *
 ******************************************************************************/
module MIP_32(Opcode, SData, TData, y_lo, c, v);
   input       [31:0] SData;
   input       [31:0] TData;
   input        [4:0] Opcode;
   output reg  [31:0] y_lo;
   output reg  c, v;

   //internal wire to hold 33 bit of the result of the addition and
   //substruction
   reg          [32:0] tempData;
   reg          [31:0] sd, td;

   //Declare temporay integer variable
   integer     tempA, tempB;

   //parameter declaration
   parameter    [4:0] Pass_S  = 5'h00,
                      Pass_T  = 5'h01,
                      ADD     = 5'h02,
                      ADDU    = 5'h03,
                      SUB     = 5'h04,
                      SUBU    = 5'h05,
                      SLT     = 5'h06,
```

```
                      SUBU    = 5'h05,
                      SLT     = 5'h06,
                      SLTU    = 5'h07,
                      AND     = 5'h08,
                      OR      = 5'h09,
                      XOR     = 5'h0A,
                      NOR     = 5'h0B,
                      SLL     = 5'h0C,
                      SRL     = 5'h0D,
                      SRA     = 5'h0E,
                      INC     = 5'h0F,
                      DEC     = 5'h10,
                      INC4    = 5'h11,
                      DEC4    = 5'h12,
                      ZEROS   = 5'h13,
                      ONES    = 5'h14,
                      SP_INIT = 5'h15,
                      ANDI    = 5'h16,
                      ORI     = 5'h17,
                      XORI    = 5'h18,
                      LUI     = 5'h19,
                      MUL     = 5'h1E,
                      DIV     = 5'h1F;
```

```
61
62      //save both input in the integer type variable
63      always @(*)
64         begin
65            tempA = SData;
66            tempB = TData;
67         end
68      //the assignment of each operation according to
69      //the opcode input
70      always @(*)
71      begin
72         c = 0;
73         v = 0;
74         tempData = 33'h0000;
75         case (Opcode)
76            Pass_S:
77               begin
78                  y_lo = SData;
79               end
80            Pass_T:
81               begin
82                  y_lo = TData;
83               end
84            ADD:
85               begin
86                  tempData = {1'b0, SData} + {1'b0, TData};
87                  y_lo     = tempData[31:0];
88                  c        = tempData[32];
89                  if (c == 0)
90                     v = 0;
91                  else
92                     v = ~tempData[32];
93               end
94            SUB:
95               begin
96                  tempData = {1'b0, SData} - {1'b0, TData};
97                  y_lo     = tempData[31:0];
98                  c        = tempData[32];
99                  if (c == 0)
```

```
99                  if (c == 0)
100                    v = 0;
101                 else
102                    v = ~tempData[32];
103              end
104           ADDU:
105              begin
106                 tempData = {1'b0, SData} + {1'b0, TData};
107                 y_lo     = tempData[31:0];
108                 c        = tempData[32];
109                 v        = tempData[32];
110              end
111           SUBU:
112              begin
113                 tempData = {1'b0, SData} - {1'b0, TData};
114                 y_lo     = tempData[31:0];
115                 c        = tempData[32];
116                 v        = tempData[32];
117              end
```

```
118         SLT:
119            begin
120               if ((tempA < tempB)|| (tempA == tempB))
121                  y_lo = 1;
122               else
123                  y_lo = 0;
124            end
125         SLTU:
126            begin
127               if (SData < TData)
128                  y_lo = 1;
129               else
130                  y_lo = 0;
131            end
132         AND:
133            begin
134               y_lo = SData & TData;
135            end
136         OR:
137            begin
138               y_lo = SData | TData;
139            end
140         XOR:
141            begin
142               y_lo = SData ^ TData;
143            end
144         NOR:
145            begin
146               y_lo = ~(SData | TData);
147            end
148         INC:
149            begin
150               tempData = {1'b0, SData} + 1;
151               y_lo     = tempData[31:0];
152            end
```

```
153         DEC:
154            begin
155               tempData = {1'b0, SData} - 1;
156               y_lo     = tempData[31:0];
157            end
158         INC4:
159            begin
160               tempData = {1'b0, SData} + 4;
161               y_lo     = tempData[31:0];
162            end
163         DEC4:
164            begin
165               tempData = {1'b0, SData} - 4;
166               y_lo     = tempData[31:0];
167            end
168         ZEROS:
169            begin
170               y_lo = 32'h0;
171            end
172         ONES:
173            begin
174               y_lo = 32'hFFFFFFFF;
175            end
```

```
176         SP_INIT:
177            begin
178               y_lo = 32'h3FC;
179            end
180         ANDI:
181            begin
182               y_lo = (SData & {16'h0, TData[15:0]});
183            end
184         ORI:
185            begin
186               y_lo = (SData | {16'h0, TData[15:0]});
187            end
188         XORI:
189            begin
190               y_lo = (SData ^ {16'h0, TData[15:0]});
191            end
192         LUI:
193            begin
194               y_lo = {TData[15:0], 16'h0};
195            end
196      endcase
197   end
198 endmodule
```

17.HILO ladable register:

```
1  `timescale 1ns / 1ps
2  /****************************** C E C S  4 4 0 ******************************
3   *
4   * File Name:  HI_Loadable_Register.v
5   * Project:    CECS 440 Senior Project Customized MIPS Processor
6   * Designer:   Juie Kim
7   * Email:      chealy_ljh@yahoo.co.uk
8   * Rev. No.:   Version 1.0
9   * Rev. Date:  November 5, 2017
10  *
11  * Purpose:    Sub module to load the MSB 32-bit output from 32-bit ALU
12  *             operations, store in the register to output MSB 32bit.
13  * Notes:      N/A
14  *
15  ****************************************************************************/
16 module HI_Loadable_Register(clk, rst, load_hi, din_hi, qOut_hi);
17
18    input    clk, rst, load_hi;
19    input    [31:0]   din_hi;
20    output   [31:0]   qOut_hi;
21
22    reg      [31:0]   qOut_hi;
23
24    always @(posedge clk, posedge rst)
25       if (rst)
26          qOut_hi <=  32'h0;
27       else if (load_hi)
28          qOut_hi <=  din_hi;
29 endmodule
```

```
`timescale 1ns / 1ps
/****************************** C E C S  4 4 0 ******************************
 *
 * File Name:  LO_Loadable_Register.v
 * Project:    CECS 440 Senior Project Customized MIPS Processor
 * Designer:   Juie Kim
 * Email:      chealy_ljh@yahoo.co.uk
 * Rev. No.:   Version 1.0
 * Rev. Date:  November 5, 2017
 *
 * Purpose:    Sub module to load the low 32-bit output from 32-bit ALU
 *             operations, store in the register to output LSB 32bit.
 * Notes:      N/A.
 *
 ******************************************************************************/
module LO_Loadable_Register(clk, rst, load_lo, din_lo, qOut_lo);

   input    clk, rst, load_lo;
   input    [31:0]   din_lo;
   output   [31:0]   qOut_lo;

   reg      [31:0]   qOut_lo;

   always @(posedge clk, posedge rst)
      if (rst)
         qOut_lo <=  32'h0;
      else if (load_lo)
         qOut_lo <=  din_lo;
endmodule
```

18. Data Memory Module:

```
`timescale 1ns / 1ps
/****************************** C E C S  4 4 0 ******************************
 *
 * File Name:  Data_Memory.v
 * Project:    CECS 440 Senior Project Customized MIPS Processor
 * Designer:   Juie Kim
 * Email:      chealy_ljh@yahoo.co.uk
 * Rev. No.:   Version 1.0
 * Rev. Date:  November 28, 2017
 *
 * Purpose:    Memory of 1K x 32-bit width. the total address is 1024 or 1K
 *             the width is 32-bit, which is asynchronous read and synchronous
 *             write with chip select to perform read and write
 * Notes:      N/A
 *
 ******************************************************************************/
module Data_Memory(clk, rst, dm_cs, dm_wr, dm_rd, Address, D_in, D_Out);

      input              clk, rst, dm_cs, dm_wr, dm_rd;
      input       [11:0] Address;
      input       [31:0] D_in;
      output      [31:0] D_Out;
      wire        [31:0] D_Out;

      reg         [7:0] jl_Mem [0:4095]; //1K memory byte addressable

      //writing to jl_Memory is synchronous
      always @(posedge clk, posedge rst)
         if (rst)
            jl_Mem[Address] <= 8'b0;
```

```
27        //writing to jl_Memory is synchronous
28        always @(posedge clk, posedge rst)
29           if (rst)
30              jl_Mem[Address] <= 8'b0;
31
32           else if (dm_cs==1 && dm_wr==1) begin
33              jl_Mem[Address + 0] <= D_in[31:24];
34              jl_Mem[Address + 1] <= D_in[23:16];
35              jl_Mem[Address + 2] <= D_in[15:8];
36              jl_Mem[Address + 3] <= D_in[7:0];
37              end
38
39           else
40              jl_Mem[Address[11:0]] <= jl_Mem[Address[11:0]];
41
42        //reading from jl_Mem is asynchronous
43        assign D_Out = (dm_cs==1 && dm_rd==1) ?
44                       {jl_Mem[Address + 0], jl_Mem[Address + 1],
45                        jl_Mem[Address + 2], jl_Mem[Address + 3]} : D_Out;
46  endmodule
47
```

19.I/O Memory Module:

```
1   `timescale 1ns / 1ps
2   /****************************** C E C S  4 4 0 ******************************
3    *
4    * File Name:  Data_Memory.v
5    * Project:    CECS 440 Senior Project Customized MIPS Processor
6    * Designer:   Juie Kim
7    * Email:      chealy_ljh@yahoo.co.uk
8    * Rev. No.:   Version 1.0
9    * Rev. Date:  November 28,  2017
10   *
11   * Purpose:    Memory of 1K x 32-bit width. the total address is 1024 or 1K
12   *             the width is 32-bit, which is asynchronous read and synchronous
13   *             write with chip select to perform read from IO module as INPUT and
14   *             write to IO module as OUPUT
15   * Notes:      N/A
16   *
17   ****************************************************************************/
18  module IO_Memory(clk, rst, intr, int_ack, io_cs, io_wr, io_rd, Address_io, D_in_io, D_Out_io);
19        input     int_ack;
20        output reg intr;
21
22        input              clk, rst, io_cs, io_wr, io_rd;
23        input       [11:0] Address_io;
24        input       [31:0] D_in_io;
25        output      [31:0] D_Out_io;
26        wire        [31:0] D_Out_io;
27
28        reg         [7:0] ioMem [0:4095]; //1K memory byte addressable
29
30        initial begin
31           intr = 0;
32  //       #1060
33           #700
34           intr = 1;
35           @(posedge int_ack)
36              intr = 0;
37        end
```

```
38
39        //writing to jl_Memory is synchronous
40        always @(posedge clk, posedge rst)
41           if (rst)
42              ioMem[Address_io] <= 8'b0;
43
44           else if (io_cs==1 && io_wr==1) begin
45              ioMem[Address_io + 0] <= D_in_io[31:24];
46              ioMem[Address_io + 1] <= D_in_io[23:16];
47              ioMem[Address_io + 2] <= D_in_io[15:8];
48              ioMem[Address_io + 3] <= D_in_io[7:0];
49              end
50           else
51              ioMem[Address_io[11:0]] <= ioMem[Address_io[11:0]];
52
53        //reading from io_Mem is asynchronous
54        assign D_Out_io = (io_cs==1 && io_rd==1) ?
55                        {ioMem[Address_io + 0], ioMem[Address_io + 1],
56                         ioMem[Address_io + 2], ioMem[Address_io + 3]} : D_Out_io;
57  endmodule
```

C.Memory Modules used for verification: Data memory and instruction memory used to test the processor

1.Data Memory Moduels:

dMem01_Fa17.dat - Notepad

File Edit Format View Help

```
@0              // 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  //0x04
24 68 AC E0     // 0x14:17
13 57 9B DF     // 0x18:1B
0F 0F 0F 0F     // 0x1C:1F
F0 F0 F0 F0     // 0x20:23  //0x08
00 00 00 09     // 0x24:27
00 00 00 0A     // 0x28:2B
00 00 00 0B     // 0x2C:2F
00 00 00 0C     // 0x30:33  //0x0C
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
```

dMem02_Fa17.dat - Notepad

File Edit Format View Help

```
@0              // 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  //0x04
24 68 AC E0     // 0x14:17
13 57 9B DF     // 0x18:1B
0F 0F 0F 0F     // 0x1C:1F
F0 F0 F0 F0     // 0x20:23  //0x08
00 00 00 09     // 0x24:27
00 00 00 0A     // 0x28:2B
00 00 00 0B     // 0x2C:2F
00 00 00 0C     // 0x30:33  //0x0C
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
```

dMem03_Fa17.dat - Notepad

File Edit Format View Help

```
@0              // 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  //0x04
24 68 AC E0     // 0x14:17
13 57 9B DF     // 0x18:1B
0F 0F 0F 0F     // 0x1C:1F
F0 F0 F0 F0     // 0x20:23  //0x08
00 00 00 09     // 0x24:27
00 00 00 0A     // 0x28:2B
00 00 00 0B     // 0x2C:2F
00 00 00 0C     // 0x30:33  //0x0C
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
```

dMem04_Fa17.dat - Notepad

File Edit Format View Help

```
@0              // 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  //0x04
24 68 AC E0     // 0x14:17
13 57 9B DF     // 0x18:1B
0F 0F 0F 0F     // 0x1C:1F
F0 F0 F0 F0     // 0x20:23  //0x08
00 00 00 09     // 0x24:27
00 00 00 0A     // 0x28:2B
00 00 00 0B     // 0x2C:2F
00 00 00 0C     // 0x30:33  //0x0C
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
```

dMem05_Fa17.dat - Notepad

File Edit Format View Help

```
@0              // 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  //0x04
24 68 AC E0     // 0x14:17
13 57 9B DF     // 0x18:1B
0F 0F 0F 0F     // 0x1C:1F
F0 F0 F0 F0     // 0x20:23  //0x08
00 00 00 09     // 0x24:27
00 00 00 0A     // 0x28:2B
00 00 00 0B     // 0x2C:2F
00 00 00 0C     // 0x30:33  //0x0C
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
```

dMem06_Fa17.dat - Notepad

File Edit Format View Help

```
@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
```

dMem07_Fa17.dat - Notepad

File Edit Format View Help

```
@0              // 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
```

dMem08_Fa17.dat - Notepad

File Edit Format View Help

```
@0              // Big Endia

00 00 00 19     // 0x00:03
00 00 03 E8     // 0x04:07
FF FF FF E7     // 0x08:0B
FF FF FC 18     // 0x0C:0F
00 00 61 A8     // 0x10:13
FF FF 9E 58     // 0x14:17
FF FF FF FF     // 0x18:1B
00 00 00 07     // 0x1C:1F
00 00 00 08     // 0x20:23
00 00 00 09     // 0x24:27
00 00 00 0A     // 0x28:2B
00 00 00 0B     // 0x2C:2F
00 00 00 0C     // 0x30:33
00 00 00 0D     // 0x34:37
00 00 00 0E     // 0x38:3B
00 00 00 0F     // 0x3C:3F

@1CC
AB CD EF 01     // 0x1CC:1CF

@3F8
00 00 00 00     // 0x3F8:3FB
```

dMem09_Fa17.dat - Notepad

File Edit Format View Help

```
@0              // Big Endian Format

00 04 09 11     // 0x00:03  //word 00 =
00 00 03 E8     // 0x04:07  //word 01 =
FF FB F6 EF     // 0x08:0B  //word 02 = -
FF FF FC 18     // 0x0C:0F  //word 03 =
00 00 01 08     // 0x10:13  //word 04 =
00 00 01 D1     // 0x14:17  //word 05 =
FF FF FE F8     // 0x18:1B  //word 06 =
FF FF FE 2F     // 0x1C:1F  //word 07 =
00 00 00 08     // 0x20:23  //word 08 =
00 00 00 09     // 0x24:27  //word 09 =
00 00 00 0A     // 0x28:2B  //word 10 =
00 00 00 0B     // 0x2C:2F  //word 11 =
00 00 00 0C     // 0x30:33  //word 12 =
00 00 00 0D     // 0x34:37  //word 13 =
00 00 00 0E     // 0x38:3B  //word 14 =
00 00 00 0F     // 0x3C:3F  //word 15 =

@1CC
AB CD EF 01     // 0x1CC:1CF

@3F8
00 00 00 00     // 0x3F8:3FB
```

dMem10_Fa17.dat - Notepad

File Edit Format View Help

```
@0              // Big Endian Format

00 04 09 11     // 0x00:03  //word 00 =  264465
00 00 03 E8     // 0x04:07  //word 01 =    1000
FF FB F6 EF     // 0x08:0B  //word 02 = -264465
FF FF FC 18     // 0x0C:0F  //word 03 =   -1000
00 00 01 08     // 0x10:13  //word 04 =     264
00 00 01 D1     // 0x14:17  //word 05 =     465
FF FF FE F8     // 0x18:1B  //word 06 =    -264
FF FF FE 2F     // 0x1C:1F  //word 07 =
00 00 00 08     // 0x20:23  //word 08 =
00 00 00 09     // 0x24:27  //word 09 =
00 00 00 0A     // 0x28:2B  //word 10 =
00 00 00 0B     // 0x2C:2F  //word 11 =
00 00 00 0C     // 0x30:33  //word 12 =
00 00 00 0D     // 0x34:37  //word 13 =
00 00 00 0E     // 0x38:3B  //word 14 =
00 00 00 0F     // 0x3C:3F  //word 15 =

@1CC
AB CD EF 01     // 0x1CC:1CF

@3F8
00 00 00 00     // 0x3F8:3FB
```

dMem11_Fa17.dat - Notepad

File Edit Format View Help

```
@0                  // Big Endian F

00 04 09 11         // 0x00:03
00 00 03 E8         // 0x04:07
FF FB F6 EF         // 0x08:0B
FF FF FC 18         // 0x0C:0F
00 00 01 08         // 0x10:13
00 00 01 D1         // 0x14:17
FF FF FE F8         // 0x18:1B
FF FF FE 2F         // 0x1C:1F
00 00 00 08         // 0x20:23
00 00 00 09         // 0x24:27
00 00 00 0A         // 0x28:2B
00 00 00 0B         // 0x2C:2F
00 00 00 0C         // 0x30:33
00 00 00 0D         // 0x34:37
00 00 00 0E         // 0x38:3B
00 00 00 0F         // 0x3C:3F

@1CC
AB CD EF 01         // 0x1CC:1CF

@3F8
00 00 00 00         // 0x3F8:3FB
```

dMem12_Fa17.dat - Notepad

File Edit Format View Help

```
@0                  // Big Endian Fo

00 04 09 11         // 0x00:03
00 00 03 E8         // 0x04:07
FF FB F6 EF         // 0x08:0B
FF FF FC 18         // 0x0C:0F
00 00 01 08         // 0x10:13
00 00 01 D1         // 0x14:17
FF FF FE F8         // 0x18:1B
FF FF FE 2F         // 0x1C:1F
00 00 00 08         // 0x20:23
00 00 00 09         // 0x24:27
00 00 00 0A         // 0x28:2B
00 00 00 0B         // 0x2C:2F
00 00 00 0C         // 0x30:33
00 00 00 0D         // 0x34:37
00 00 00 0E         // 0x38:3B
00 00 00 0F         // 0x3C:3F

@1CC
AB CD EF 01         // 0x1CC:1CF

@3F8
00 00 00 00         // 0x3F8:3FB
```

dMem13_Fa17.dat - Notepad

File Edit Format View Help

```
@0

C3 C3 C3 C3
12 34 56 78
89 AB CD EF
A5 A5 A5 A5
5A 5A 5A 5A
24 68 AC E0
13 57 9B DF
0F 0F 0F 0F
F0 F0 F0 F0
00 00 00 09
00 00 00 0A
00 00 00 0B
00 00 00 0C
00 00 00 0D
FF FF FF F8
00 00 75 CC

@1CC
AB CD EF 01

@3F8
00 00 00 00
```

dMem14_Fa17.dat - Notepad

File Edit Format View Help

```
@0

C3 C3 C3 C3
12 34 56 78
89 AB CD EF
A5 A5 A5 A5
5A 5A 5A 5A
24 68 AC E0
13 57 9B DF
0F 0F 0F 0F
F0 F0 F0 F0
00 00 00 09
00 00 00 0A
00 00 00 0B
00 00 00 0C
00 00 00 0D
FF FF FF F8
00 00 75 CC

@1CC
AB CD EF 01

@3F8
00 00 00 00

@3FC
00 00 02 00
```

2. Instruction Memory:

iMem01_Fa17.dat - Notepad

File Edit Format

```
@0
3c 01 12 34
34 21 56 78
3c 02 87 65
34 42 43 21
00 01 18 20
10 22 00 01
10 23 00 03
3c 0e ff ff
35 ce ff ff
00 00 00 0d
00 00 70 20
14 23 00 01
14 22 00 03
3c 0f ff ff
35 ef ff ff
00 00 00 0d
00 00 78 20
3c 0d 10 01
35 ad 00 c0
ad a1 00 00
00 00 00 0d
```

iMem02_Fa17.dat - Notepad

File Edit Format View

```
@0
3c 01 ff ff
34 21 ff ff
20 02 00 10
3c 0f 10 01
35 ef 00 c0
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
3c 0e 5a 5a
35 ce 3c 3c
00 00 00 0d
```

iMem03_Fa17.dat - Notepad

File Edit Format View

```
@0
3c 01 80 00
34 21 ff ff
20 02 00 10
3c 0f 10 01
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
3c 0e 5a 5a
35 ce 3c 3c
00 00 00 0d
```

iMem04_Fa17.dat - Notepad

File Edit Format View Help

```
@0
3c 01 ff ff
34 21 ff ff
20 02 00 10
3c 0f 10 01
35 ef 00 c0
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
3c 0e 5a 5a
35 ce 3c 3c
00 00 00 0d
```

iMem06_Fa17.dat - Note

File Edit Format View

```
@0
3c 0f 10 01
35 ef 00 00
3c 0e 10 01
35 ce 00 c0
20 0d 00 10
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
8d e9 00 24
8d ea 00 28
8d eb 00 2c
8d ec 00 30
8d f1 00 00
ad d1 00 00
21 ef 00 04
21 ce 00 04
21 ad ff ff
15 a0 ff fa
00 00 00 0d
```

iMem07_Fa17.dat - Notep

File Edit Format View

```
@0
3c 0f 10 01
35 ef 00 00
3c 0e 10 01
35 ce 00 c0
20 0d 00 10
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
8d e9 00 24
8d ea 00 28
8d eb 00 2c
8d ec 00 30

0c 10 00 15
3c 0f ff ff
35 ef ff ff
00 00 00 0d

8d f1 00 00
ad d1 00 00
21 ef 00 04
21 ce 00 04
21 ad ff ff
15 a0 ff fa
03 e0 00 08
00 00 00 0d
```

iMem08_Fa17.da

File Edit Format

```
@0
3c 0f 10 01
35 ef 00 00
8d e1 00 00
8d e2 00 04
8d e3 00 08
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
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
00 00 60 10
14 cb 00 10
14 ec 00 12
00 64 00 18
00 00 68 12
14 ad 00 12

3c 0e 00 00
35 ce 00 00
00 00 00 0d
3c 0e ff ff
35 ce ff ff
00 00 00 0d
3c 0e ff ff
35 ce ff fe
00 00 00 0d
3c 0e ff ff
35 ce ff fd
00 00 00 0d
3c 0e ff ff
35 ce ff fc
00 00 00 0d
3c 0e ff ff
35 ce ff fb
00 00 00 0d
3c 0e ff ff
35 ce ff fa
00 00 00 0d
```

iMem09_Fa17.dat - Notepad

File Edit Format View Help

```
@0
3c 0f 10 01
35 ef 00 c0
20 01 ff 8a
20 02 00 8a
0c 10 00 22
3c 0d 77 88
35 ad 77 88
3c 0c 88 77
35 8c 88 77
3c 0b ff ff
35 6b ff ff
01 ac 50 26
11 4b 00 02
20 0e ff fb
00 00 00 0d
01 ac 48 24
11 20 00 02
20 0e ff fa
00 00 00 0d
01 e2 48 25
3c 08 10 01
35 08 00 ca
11 09 00 02
20 0e ff f9
00 00 00 0d
01 e2 48 27
3c 08 ef fe
35 08 ff 35
```

iMem10_Fa17.da

File Edit Format

```
8d e1 00 00
8d e2 00 04
8d e3 00 08
8d e4 00 0c
8d e5 00 10
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
15 46 00 18

00 62 00 1a
00 00 48 12
00 00 50 10
15 27 00 17
15 48 00 19

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
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
00 00 00 0d

3c 0e ff ff
35 ce ff ff
00 00 00 0d
3c 0e ff ff
35 ce ff fe
00 00 00 0d
3c 0e ff ff
35 ce ff fd
00 00 00 0d
3c 0e ff ff
35 ce ff fc
00 00 00 0d
3c 0e ff ff
35 ce ff fb
00 00 00 0d
3c 0e ff ff
35 ce ff fa
00 00 00 0d
3c 0e ff ff
35 ce ff f9
00 00 00 0d
3c 0e ff ff
35 ce ff f8
00 00 00 0d
```

iMem11_Fa17.da

File Edit Format

```
@0
3c 0f 10 01
35 ef 00 c0
20 01 ff 8a
20 02 00 8a
0c 10 00 1a
3c 0d ff ff
35 ad 55 55
3c 0c ff ff
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
11 00 00 02
20 0e ff f9
00 00 00 0d
31 87 f5 fa
00 ea 40 22
11 00 00 02
20 0e ff f8
00 00 00 0d
ad e1 00 18
00 00 00 0d
00 00 00 0d
2c 23 ff 8b
14 60 00 02
20 0e ff ff
00 00 00 0d
20 04 00 c0
ad e4 00 00
2c 23 ff 89
10 60 00 02
20 0e ff fe
00 00 00 0d
20 05 00 c4
ad e5 00 04
2c 23 ff 8a
10 60 00 02
20 0e ff fd
00 00 00 0d
20 06 00 c8
ad e6 00 08
2c 43 00 8b
14 60 00 02
20 0e ff fc
00 00 00 0d
20 07 00 cc
ad e7 00 0c
2c 43 00 89
10 60 00 02
20 0e ff fb
00 00 00 0d
20 08 00 d0
ad e8 00 10
2c 43 00 8a
10 60 00 02
20 0e ff fa
00 00 00 0d
20 06 00 d4
ad e6 00 14
20 0e 00 00
03 e0 00 08
```

iMem12_Fa17.dat

File Edit Format

```
@0
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
18 20 00 02
20 0e ff ff
00 00 00 0d
20 03 00 c0
ad e3 00 00
18 40 00 03
20 04 00 c4
ad e4 00 04
08 10 00 13
20 0e ff fe
00 00 00 0d
```

Untitled - Notepad

File Edit Format View Help

```
18 00 00 02
20 0e ff fd
00 00 00 0d
20 05 00 c8
ad e5 00 08
1c 40 00 02
20 0e ff fc
00 00 00 0d
20 06 00 cc
ad e6 00 0c
1c 20 00 03
20 07 00 d0
ad e7 00 10
08 10 00 23
20 0e ff fb
00 00 00 0d
1c 20 00 03
20 08 00 d4
ad e8 00 14
08 10 00 29
20 0e ff fa
00 00 00 0d
20 0e 00 00
03 e0 00 08
```

iMem13_Fa17_w_isr

File Edit Format V

```
@0
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
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
```

```
@200
3c 10 10 01
36 10 00 c0
3c 11 80 00
36 31 ff ff
20 12 00 10
76 11 00 00
00 11 88 43
22 10 00 04
22 52 ff ff
16 40 ff fb
3c 10 10 01
36 10 00 c0
72 13 00 00
72 14 00 04
72 15 00 08
72 16 00 0c
72 17 00 10
72 18 00 14
03 e0 00 08
```

iMem14_Fa17_w_isr.dat - N

File Edit Format View

```
@0
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
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
```

```
@200
3c 10 10 01
36 10 00 c0
3c 11 80 00
36 31 ff ff
20 12 00 10
76 11 00 00
00 11 88 43
22 10 00 04
22 52 ff ff
16 40 ff fb
3c 10 10 01
36 10 00 c0
72 13 00 00
72 14 00 04
72 15 00 08
72 16 00 0c
72 17 00 10
72 18 00 14
7B A0 00 00
```

iMem_IO_External_isr.dat - Notepad

File Edit Format View Help

```
@0

00 00 00 1F  // main:      setie
3c 10 10 01  //            lui  $16, 0x1001        # change pointer to write to dMem
36 10 00 C0  //            ori  $16, 0x00C0        #  0x100100C0 into r16
3C 0F 10 01  //            lui  $15, 0x1001
35 EF 00 00  //            ori  $15, 0x0000      # $r15 <-- 0x10010000  (source pointer)

8D E1 00 00  //            lw   $01, 00($15)      #$1 = 19 load from mem at 0 location
8D E2 00 04  //            lw   $02, 04($15)      #$2 = 03e8
8D E3 00 08  //            lw   $03, 08($15)
8D E4 00 0C  //            lw   $04, 12($15)
8D E5 00 10  //            lw   $05, 16($15)
8D E6 00 14  //            lw   $06, 20($15)
8D E7 00 18  //            lw   $07, 24($15)
8D E8 00 1C  //            lw   $08, 28($15)
8D E9 00 20  //            lw   $09, 32($15)
8D EA 00 24  //            lw   $10, 36($15)
8D EB 00 28  //            lw   $11  40($15)
8D EC 00 2C  //            lw   $12, 44($15)
8D ED 00 30  //            lw   $13, 48($15)
8D EE 00 34  //            lw   $14, 52($15)
8D EF 00 38
00 00 00 0d  //            break
```

iMem_IO_External_isr.dat - Notepad

```
@200
                          //*************************************************
                          // In this ISR, we will implement reading input
                          // to write to $r17 to $28
                          // Note: this "ISR" expects the "return address"
                          //       to have been saved on the stack and read
                          //       back from the stack
                          //*************************************************
B3 BF 00 00  // isr:        push   $31, ($sp)       #push return address PC to stack
B3 A8 00 00  //             push   $8,  ($sp)       #push the flag to the stack
3c 11 80 00  //             lui    $17, 0x8000      #initialize the pattern of
36 31 ff ff  //             ori    $17, 0xFFFF      #  0x8000FFFF into r17
20 12 00 1E  //             addi   $18, $0, 0x1E    #loop counter set to 30

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

3c 10 10 01  //             lui  $16, 0x1001        #  change pointer to write to dMem
36 10 00 C0  //             ori  $16, 0x00C0        #  0x100100C0 into r16
72 11 00 00  //             input  $17,  0($16)     #  and input from 0
72 12 00 04  //             input  $18,  4($16)     #  and input from 4
72 13 00 08  //             input  $19,  8($16)     #  and input from 8
72 14 00 0C  //             input  $20,  c($16)     #
72 15 00 10  //             input  $21, 10($16)
72 16 00 14  //             input  $22, 14($16)
72 17 00 18  //             input  $23, 18($16)
72 18 00 1C  //             input  $24, 1c($16)
72 19 00 20  //             input  $25, 20($16)     #    the IO locations,
72 1A 00 24  //             input  $26, 24($16)     #    starting from 0xC0
72 1B 00 28  //             input  $27, 28($16)
72 1C 00 2C  //             input  $28, 2c($16)
7B A0 00 00  //             reti
```

iMem15_Enhenced_Push_Pop_PC.dat - Notepad

File Edit Format View Help

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

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

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

@200
                        //*************************************************
                        // In this ISR, we will implement writing
                        // some patterns to the IO space, and then
                        // reading them back.
                        // Note: this "ISR" expects the "return address"
                        // to have been read from the stack
                        //*************************************************
B3 BF 00 00  // isr:      push   $31, ($sp)       #push return address PC to stack
B3 A8 00 00  //            push   $8,  ($sp)       #push the flag to the stack
3c 10 10 01  //            lui    $16, 0x1001      #load destination IO address
36 10 00 c0  //            ori    $16, 0x00C0      #  0x100100C0 into r16
3c 11 80 00  //            lui    $17, 0x8000      #initialize the pattern of
36 31 ff ff  //            ori    $17, 0xFFFF      #  0x8000FFFF into r17
20 12 00 10  //            addi   $18, $0, 0x10    #loop counter set to 16

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

3c 10 10 01  //            lui    $16, 0x1001      #load source IO address
36 10 00 c0  //            ori    $16, 0x00C0      #  0x100100C0 into r16
72 13 00 00  //            input  $19,  0($16)     #  and input from 6
72 14 00 04  //            input  $20,  4($16)     #    the IO locations,
72 15 00 08  //            input  $21,  8($16)     #    starting from 0xC0
72 16 00 0c  //            input  $22, 12($16)
72 17 00 10  //            input  $23, 16($16)
72 18 00 14  //            input  $24, 20($16)
//7B A0 00 00  //            reti
B7 A8 00 00  //            pop    $8,  ($sp)       #pop FLAGS from the stack, write to $8
B9 00 00 00  //            RETFL  $8               #load FLAGS from $r8
B7 A8 00 00  //            pop    $8,  ($sp)       #pop  PC from the stack, write to $8
7D 00 00 00  //            RETPC  $8               #load PC from $r8
```

D. Modules Annotation:

Module_1 and simulation result:

```
@0
main:
        lui  $01, 0x1234
        ori  $01, 0x5678          # LI   R01,  0x12345678
        lui  $02, 0x8765
        ori  $02, 0x4321          # LI   R02,  [illegible]
        add  $03, $00, $01        # COPY R03, R01

        beq  $01, $02, no_eq      # should not branch
        beq  $01, $03, yes_eq     # should branch
        no_eq:
        lui  $14, 0xFFFF
        ori  $14, 0xFFFF          # LI   R14,  0xFFFFFFFF  "fail
        break

yes_eq:
        add  $14, $0, $0          # CLR  R14  "pass flag"
        bne  $01, $03, no_ne      # should not branch
        bne  $01, $02, yes_ne     # should branch
        no_ne:     lui  $15, 0xFFFF
        ori  $15, 0xFFFF          # LI   R15,  0xFFFFFFFF  "fail flag"
        break
yes_ne:
        add  $15, $0, $0          # CLR  R15  "pass flag"
        lui  $13, 0x1001
        ori  $13, 0x00C0          # LI   R13,  0x100100C0
        sw   $01, 0($13)          # ST  [R13], R01
        break
```

$r1=12345678 is not equal $r2=87654321, so the first beq is not taken. $r1=$r3 so branch to yes_eq. $r1 is not equal $r2, so take the branch. Store $r1=1234567 at 0xC0 indicate the pass all the flags and execute the program successfully.

Isim result:

```
time= 641.0 ps, S_addr[00]=00000000 || T_addr[10]=xxxxxxxx
time= 651.0 ps, S_addr[01]=12345678 || T_addr[11]=xxxxxxxx
time= 661.0 ps, S_addr[02]=87654321 || T_addr[12]=xxxxxxxx
time= 671.0 ps, S_addr[03]=12345678 || T_addr[13]=xxxxxxxx
time= 681.0 ps, S_addr[04]=xxxxxxxx || T_addr[14]=xxxxxxxx
time= 691.0 ps, S_addr[05]=xxxxxxxx || T_addr[15]=xxxxxxxx
time= 701.0 ps, S_addr[06]=xxxxxxxx || T_addr[16]=xxxxxxxx
time= 711.0 ps, S_addr[07]=xxxxxxxx || T_addr[17]=xxxxxxxx
time= 721.0 ps, S_addr[08]=xxxxxxxx || T_addr[18]=xxxxxxxx
time= 731.0 ps, S_addr[09]=xxxxxxxx || T_addr[19]=xxxxxxxx
time= 741.0 ps, S_addr[0a]=xxxxxxxx || T_addr[1a]=xxxxxxxx
time= 751.0 ps, S_addr[0b]=xxxxxxxx || T_addr[1b]=xxxxxxxx
time= 761.0 ps, S_addr[0c]=xxxxxxxx || T_addr[1c]=xxxxxxxx
time= 771.0 ps, S_addr[0d]=100100c0 || T_addr[1d]=000003fc
time= 781.0 ps, S_addr[0e]=00000000 || T_addr[1e]=xxxxxxxx
time= 791.0 ps, S_addr[0f]=00000000 || T_addr[1f]=xxxxxxxx
DATA MEMORY FILE
time= 801.0 ps, dm_addr[000000c0]=12345678
time= 811.0 ps, dm_addr[000000c4]=xxxxxxxx
time= 821.0 ps, dm_addr[000000c8]=xxxxxxxx
time= 831.0 ps, dm_addr[000000cc]=xxxxxxxx
time= 841.0 ps, dm_addr[000000d0]=xxxxxxxx
time= 851.0 ps, dm_addr[000000d4]=xxxxxxxx
time= 861.0 ps, dm_addr[000000d8]=xxxxxxxx
time= 871.0 ps, dm_addr[000000dc]=xxxxxxxx
time= 881.0 ps, dm_addr[000000e0]=xxxxxxxx
time= 891.0 ps, dm_addr[000000e4]=xxxxxxxx
time= 901.0 ps, dm_addr[000000e8]=xxxxxxxx
time= 911.0 ps, dm_addr[000000ec]=xxxxxxxx
time= 921.0 ps, dm_addr[000000f0]=xxxxxxxx
time= 931.0 ps, dm_addr[000000f4]=xxxxxxxx
time= 941.0 ps, dm_addr[000000f8]=xxxxxxxx
time= 951.0 ps, dm_addr[000000fc]=xxxxxxxx
```

Module_2 and simulation result:

```
@0
main:
        lui  $01, 0xFFFF
        ori  $01, 0xFFFF          # LI   R01   0xFFFFFFF
        addi $02, $00, 0x10       # LI   R
        lui  $15, 0x1001
        ori  $15, 0x00C0          # LI   R15,  0x100100C0
top:
        srl  $01, $01, 1          # logical shift right 1 b
        sw   $01, 0($15)          # ST  [R15], R01
        addi $15, $15, 4          # inc memory pointer 4 bytes
        addi $02, $02, -1         # decrement the loop counter
        bne  $02, $00, top        #  and jmp to top if not finished
        j    exit                 # jump around a halt instruction
        break

exit:
        lui  $14, 0x5A5A
        ori  $14, 0x3C3C          # LI   R14,  0x5A5A3C3C
        break
```

Initial $1=ffffffff. $1 is srl and store each value in mem location oc 0xC0. $2 is loop counter $2=16. The first shift is 7fffffff. End of Program, memory is fill with 16 values.

Isim result:

```
time=3591.0 ps, S_addr[00]=00000000 || T_addr[10]=xxxxxxxx
time=3601.0 ps, S_addr[01]=0000ffff || T_addr[11]=
time=3611.0 ps, S_addr[02]=00000000 || T_addr[12]=xxxx
time=3621.0 ps, S_addr[03]=xxxxxxxx || T_addr[13]=xxxxxxxx
time=3631.0 ps, S_addr[04]=xxxxxxxx || T_addr[14]=xxxxxxxx
time=3641.0 ps, S_addr[05]=xxxxxxxx || T_addr[15]=xxxxxxxx
time=3651.0 ps, S_addr[06]=xxxxxxxx || T_addr[16]=xxxxxxxx
time=3661.0 ps, S_addr[07]=xxxxxxxx || T_addr[17]=xxxxxxxx
time=3671.0 ps, S_addr[08]=xxxxxxxx || T_addr[18]=xxxxxxxx
time=3681.0 ps, S_addr[09]=xxxxxxxx || T_addr[19]=xxxxxx
time=3691.0 ps, S_addr[0a]=xxxxxxxx || T_addr[1a]=xx xxxxx
time=3701.0 ps, S_addr[0b]=xxxxxxxx || T_addr[1 ]=xxxxxxxx
time=3711.0 ps, S_addr[0c]=xxxxxxxx || T_a r[1c]=xxxxxxxx
time=3721.0 ps, S_addr[0d]=xxxxxxxx || T_addr[1d]=000003fc
time=3731.0 ps, S_addr[0e]=5a5a3c3c || T_addr[1e]=xxxxxxxx
time=3741.0 ps, S_addr[0f]=10010100 || T_addr[1f]=xxxxxxxx
DATA MEMORY FILE
time=3751.0 ps, dm_addr[000000c0]=7fffffff
time=3761.0 ps, dm_addr[000000c4]=3fffffff
time=3771.0 ps, dm_addr[000000c8]=1fffffff
time=3781.0 ps, dm_addr[000000cc]=0fffffff
time=3791.0 ps, dm_addr[000000d0]=07ffffff
time=3801.0 ps, dm_addr[000000d4]=03ffffff
time=3811.0 ps, dm_addr[000000d8]=01ffffff
time=3821.0 ps, dm_addr[000000dc]=00ffffff
time=3831.0 ps, dm_addr[000000e0]=007fffff
time=3841.0 ps, dm_addr[000000e4]=003fffff
time=3851.0 ps, dm_addr[000000e8]=001fffff
time=3861.0 ps, dm_addr[000000ec]=000fffff
time=3871.0 ps, dm_addr[000000f0]=0007ffff
time=3881.0 ps, dm_addr[000000f4]=0003ffff
time=3891.0 ps, dm_addr[000000f8]=0001ffff
time=3901.0 ps, dm_addr[000000fc]=0000ffff
```

$r2=0 to break out of the loop. Last value store in mem is $1=0000ffff. Instruction j is successfully executed when $14=5a5a3c3c.

Module_3 and simulation result:

```
@0
main:
        lui  $01, 0x8000
        ori  $01, 0xFFFF                 # LI   R01,  0x8000FFFF
        addi $02, $00, 0x10              # LI   R02,
        lui  $15, 0x1001
        ori  $15, 0x00C0                 # LI   R15,  0x100100C0
top:
        sra  $01, $01, 1                 # logical shift right 1 bit
        sw   $01, 0($15)                 # ST  [R15], R01
        addi $15, $15, 4                 # increment the memory pointer 4 b
        bne  $02, $00, top               #  and jmp to top if not finished
        j    exit                        # jump around a halt instruction
break
exit:
        lui  $14, 0x5A5A
        ori  $14, 0x3C3C                 # LI   R14,  0x5A5A3C3C
        break
```

Initial $1=8000ffff. $1 is sra and store each value in mem location oc 0xC0. $2 is loop counter $2=16.

End of Program, memory is fill with 16 values.

Isim result:

```
time=3591.0 ps, S_addr[00]=00000000 || T_addr[10]=xxxxxxxx
time=3601.0 ps, S_addr[01] =ffff8000 || T_addr[11]=xxxxxxxx
time=3611.0 ps, S_addr[02]=00000000 || T_addr[12]=xxxxxxxx
time=3621.0 ps, S_addr[03]=xxxxxxxx || T_addr[13]=xxxxxxxx
time=3631.0 ps, S_addr[04]=xxxxxxxx || T_addr[14]=xxxxxxxx
time=3641.0 ps, S_addr[05]=xxxxxxxx || T_addr[15]=xxxxxxxx
time=3651.0 ps, S_addr[06]=xxxxxxxx || T_addr[16]=xxxxxxxx
time=3661.0 ps, S_addr[07]=xxxxxxxx || T_addr[17]=xxxxxxxx
time=3671.0 ps, S_addr[08]=xxxxxxxx || T_addr[18]=xxxxxxxx
time=3681.0 ps, S_addr[09]=xxxxxxxx || T_addr[19]=xxxxxxxx
time=3691.0 ps, S_addr[0a]=xxxxxxxx || T_addr[1a]=xxxxxxxx
time=3701.0 ps, S_addr[0b]=xxxxxxxx || T_addr[1b]=xxxxxxxx
time=3711.0 ps, S_addr[0c]=xxxxxxxx || T_addr[1c]=xxxxxxxx
time=3721.0 ps, S_addr[0d]=xxxxxxxx || T_addr[1d]=000003fc
time=3731.0 ps, S_addr[0e]=5a5a3c3c || T_addr[1e]=xxxxxxxx
time=3741.0 ps, S_addr[0f]=10010100 || T_addr[1f]=xxxxxxxx
DATA MEMORY FILE
time=3751.0 ps, dm_addr[000000c0]=c0007fff
time=3761.0 ps, dm_addr[000000c4]=e0003fff
time=3771.0 ps, dm_addr[000000c8]=f0001fff
time=3781.0 ps, dm_addr[000000cc]=f8000fff
time=3791.0 ps, dm_addr[000000d0]=fc0007ff
time=3801.0 ps, dm_addr[000000d4]=fe0003ff
time=3811.0 ps, dm_addr[000000d8]=ff0001ff
time=3821.0 ps, dm_addr[000000dc]=ff8000ff
time=3831.0 ps, dm_addr[000000e0]=ffc0007f
time=3841.0 ps, dm_addr[000000e4]=ffe0003f
time=3851.0 ps, dm_addr[000000e8]=fff0001f
time=3861.0 ps, dm_addr[000000ec]=fff8000f
time=3871.0 ps, dm_addr[000000f0]=fffc0007
time=3881.0 ps, dm_addr[000000f4]=fffe0003
time=3891.0 ps, dm_addr[000000f8]=ffff0001
time=3901.0 ps, dm_addr[000000fc]=ffff8000
```

$r2=0 to break out of the loop. Last value store in mem is $1=ffff8000. Instruction j is successfully executed when $14=5a5a3c3c.

Moduees_4 and simulation result:

```
@0
main:
        lui  $01, 0xFFFF
        ori  $01, 0xFFFF        # LI   R01,  0xFFFFFFF
        addi $02, $00, 0x10     # LI   R02,  0x10
        lui  $15, 0x1001
        ori  $15, 0x00C0        # LI   R15,  0x100100C0

top:
        sll  $01, $01, 1        # logical shift left 1 bit
        sw   $01, 0($15)        # ST  [R15], R01
        addi $15, $15, 4        # incre
        addi $02, $02, -1       # decrement the loop counter
        slt  $03, $00, $02      # r3 <--1 if r0 < r2
        bne  $03, $00, top      # jmp if r3==1

        j  exit                # jump around a halt instruction
        break
exit:

        lui  $14, 0x5A5A
        ori  $14, 0x3C3C        # LI   R14,  0x5A5A3C3C
        break
```

Load $15=0x100100C0 as the base address to write to memory. In the loop value from $1 is stored in memory location 0xC0

Program break out of the loop when $r2=ffffffff. PC jump to exit location write to $14=0x5a5a3c3c. $r3=0 when finish the loop, indicate successful executing the program.

Isim result:

```
BREAK INSTRUCTION FETCHED 4455.0 ps
REGISTER' AFTER BREAK
time=4471.0 ps, S_addr[00]=00000000 || T_addr[10]=xxxxxxxx
time=4481.0 ps, S_addr[01]=fffe0000 || T_addr[11]=xxxxxxxx
time=4491.0 ps, S_addr[02]=ffffffff || T_addr[12]=xxxxxxxx
time=4501.0 ps, S_addr[03]=00000000 || T_addr[13]=xxxxxxxx
time=4511.0 ps, S_addr[04]=xxxxxxxx || T_addr[14]=xxxxxxxx
time=4521.0 ps, S_addr[05]=xxxxxxxx || T_addr[15]=xxxxxxxx
time=4531.0 ps, S_addr[06]=xxxxxxxx || T_addr[16]=xxxxxxxx
time=4541.0 ps, S_addr[07]=xxxxxxxx || T_addr[17]=xxxxxxxx
time=4551.0 ps, S_addr[08]=xxxxxxxx || T_addr[18]=xxxxxxxx
time=4561.0 ps, S_addr[09]=xxxxxxxx || T_addr[19]=xxxxxxxx
time=4571.0 ps, S_addr[0a]=xxxxxxxx || T_addr[1a]=xxxxxxxx
time=4581.0 ps, S_addr[0b]=xxxxxxxx || T_addr[1b]=xxxxxxxx
time=4591.0 ps, S_addr[0c]=xxxxxxxx || T_addr[1c]=xxxxxxxx
time=4601.0 ps, S_addr[0d]=xxxxxxxx || T_addr[1d]=000003fc
time=4611.0 ps, S_addr[0e]=5a5a3c3c || T_addr[1e]=xxxxxxxx
time=4621.0 ps, S_addr[0f]=10010104 || T_addr[1f]=xxxxxxxx
DATA MEMORY FILE
time=4631.0 ps, dm_addr[000000c0]=fffffffe
time=4641.0 ps, dm_addr[000000c4]=ffffffffc
time=4651.0 ps, dm_addr[000000c8]=ffffffff8
time=4661.0 ps, dm_addr[000000cc]=ffffffff0
time=4671.0 ps, dm_addr[000000d0]=ffffffe0
time=4681.0 ps, dm_addr[000000d4]=ffffffc0
time=4691.0 ps, dm_addr[000000d8]=ffffff80
time=4701.0 ps, dm_addr[000000dc]=ffffff00
time=4711.0 ps, dm_addr[000000e0]=fffffe00
time=4721.0 ps, dm_addr[000000e4]=fffffc00
time=4731.0 ps, dm_addr[000000e8]=fffff800
time=4741.0 ps, dm_addr[000000ec]=fffff000
time=4751.0 ps, dm_addr[000000f0]=ffffe000
time=4761.0 ps, dm_addr[000000f4]=ffffc000
time=4771.0 ps, dm_addr[000000f8]=ffff8000
time=4781.0 ps, dm_addr[000000fc]=ffff0000
```

Module_5 and simulation result:

```
@0
main:
        lui  $01, 0xFFFF
        ori  $01, 0xFFFF        # LI   R01,  0xFFFFFFF
        addi $02, $00, -16      # LI   R02,  -16
        lui  $15, 0x1001
        ori  $15, 0x00C0        # LI   R15,  0x100100C0

top:
        sll  $01, $01, 1        # logical shift left 1 bit
        sw   $01, 0($15)        # ST  [R15], R01
        addi $15, $15, 4        # increment the memory pointer 4 bytes
        addi $02, $02, 1        # increment the loop counter
        slti $03, $02, 0        # r3 <--1 if r2 < 0
        bne  $03, $00, top      # jmp if r3==1

        j      exit              # jump around a halt instruction
        break

exit:
        lui  $14, 0x5A5A
        ori  $14, 0x3C3C        # 
        break
```

Load $15=0x100100C0 as the base address to write to memory. In the loop value from $1 is stored in memory location 0xC0

C onten of $1 after sll is stored to mem starting from location 0xC0. $r3 and $2 is the counter. Loop break when $3== 0. PC jump to location exit and write to $14=5a5a3c3c indicate the end of the program.

Isim result:

```
time=4471.0 ps, S_addr[00]=00000000 || T_addr[10]=xxxxxxxx
time=4481.0 ps, S_addr[01]=fffe0000 || T_addr[11]=xxxxxxxx
time=4491.0 ps, S_addr[02]=00000001 || T_addr[12]=xxxxxxxx
time=4501.0 ps, S_addr[03]=00000000 || T_addr[13]=xxxxxxxx
time=4511.0 ps, S_addr[04]=xxxxxxxx || T_addr[14]=xxxxxxxx
time=4521.0 ps, S_addr[05]=xxxxxxxx || T_addr[15]=xxxxxxxx
time=4531.0 ps, S_addr[06]=xxxxxxxx || T_addr[16]=xxxxxxxx
time=4541.0 ps, S_addr[07]=xxxxxxxx || T_addr[17]=xxxxxxxx
time=4551.0 ps, S_addr[08]=xxxxxxxx || T_addr[18]=xxxxxxxx
time=4561.0 ps, S_addr[09]=xxxxxxxx || T_addr[19]=xxxxxxxx
time=4571.0 ps, S_addr[0a]=xxxxxxxx || T_addr[1a]=xxxxxxxx
time=4581.0 ps, S_addr[0b]=xxxxxxxx || T_addr[1b]=xxxxxxxx
time=4591.0 ps, S_addr[0c]=xxxxxxxx || T_addr[1c]=xxxxxxxx
time=4601.0 ps, S_addr[0d]=xxxxxxxx || T_addr[1d]=000003fc
time=4611.0 ps, S_addr[0e]=5a5a3c3c || T_addr[1e]=xxxxxxxx
time=4621.0 ps, S_addr[0f]=10010104 || T_addr[1f]=xxxxxxxx
DATA MEMORY FILE
time=4631.0 ps, dm_addr[000000c0]=fffffffe
time=4641.0 ps, dm_addr[000000c4]=fffffffc
time=4651.0 ps, dm_addr[000000c8]=fffffff8
time=4661.0 ps, dm_addr[000000cc]=fffffff0
time=4671.0 ps, dm_addr[000000d0]=ffffffe0
time=4681.0 ps, dm_addr[000000d4]=ffffffc0
time=4691.0 ps, dm_addr[000000d8]=ffffff80
time=4701.0 ps, dm_addr[000000dc]=ffffff00
time=4711.0 ps, dm_addr[000000e0]=fffffe00
time=4721.0 ps, dm_addr[000000e4]=fffffc00
time=4731.0 ps, dm_addr[000000e8]=fffff800
time=4741.0 ps, dm_addr[000000ec]=fffff000
time=4751.0 ps, dm_addr[000000f0]=ffffe000
time=4761.0 ps, dm_addr[000000f4]=ffffc000
time=4771.0 ps, dm_addr[000000f8]=ffff8000
time=4781.0 ps, dm_addr[000000fc]=ffff0000
```

Module_6 and simulation result:

```
@0
        lui  $15, 0x1001
        ori  $15, 0x0000        # LI   R15,  0x10010000  dest data pointer
        lui  $14, 0x1001
        ori  $14, 0x00C0        # LI   R14,  0x100100C0  dest data pointer
        addi $13, $00, 16       # LI   R13,  16          loop counter
        lw   $01, 04($15)       # Load
        lw   $02, 08($15)       #   R01
        lw   $03, 12($15)       #     to
        lw   $04, 16($15)       #      R12
        lw   $05, 20($15)
        lw   $06, 24($15)
        lw   $07, 28($15)
        lw   $08, 32($15)
        lw   $09, 36($15)
        lw   $10, 40($15)
        lw   $11, 44($15)
        lw   $12, 48($15)
mem2mem:
        lw   $17, 00($15)       # do mem to
        sw   $17, 00($14)       #   mem transfer
        addi $15, $15, 04       # bump both source
        addi $14, $14, 04       #   and dest pointers
        addi $13, $13, -1       # dec the loop counter
        bne  $13, $00, mem2mem  #   and con[illegible] done
        break
```

Load from memory with offset 4 with the content of 12345678 into $r1 to $r12. When execute **jal** instruction, it jump to mem2mem location.

In the loop, instruction dMem from location 0x00 to 0xC0 16 location. $r13 is the pointer. When break out of the loop $r3==0 and $14==10010100 after inc by 4 16 time. Mem location from 0XC0 is store with data from location 0x00 to 3C.

Isim result:

```
BREAK INSTRUCTION FETCHED 4865.0 ps
REGISTER' AFTER BREAK
time=4881.0 ps, S_addr[00]=00000000 || T_addr[10]=xxxxxxxx
time=4891.0 ps, S_addr[01]=12345678 || T_addr[11]=000075cc
time=4901.0 ps, S_addr[02]=89abcdef || T_addr[12]=xxxxxxxx
time=4911.0 ps, S_addr[03]=a5a5a5a5 || T_addr[13]=xxxxxxxx
time=4921.0 ps, S_addr[04]=5a5a5a5a || T_addr[14]=xxxxxxxx
time=4931.0 ps, S_addr[05]=2468ace0 || T_addr[15]=xxxxxxxx
time=4941.0 ps, S_addr[06]=13579bdf || T_addr[16]=xxxxxxxx
time=4951.0 ps, S_addr[07]=0f0f0f0f || T_addr[17]=xxxxxxxx
time=4961.0 ps, S_addr[08]=f0f0f0f0 || T_addr[18]=xxxxxxxx
time=4971.0 ps, S_addr[09]=00000009 || T_addr[19]=xxxxxxxx
time=4981.0 ps, S_addr[0a]=0000000a || T_addr[1a]=xxxxxxxx
time=4991.0 ps, S_addr[0b]=0000000b || T_addr[1b]=xxxxxxxx
time=5001.0 ps, S_addr[0c]=0000000c || T_addr[1c]=xxxxxxxx
time=5011.0 ps, S_addr[0d]=00000000 || T_addr[1d]=000003fc
time=5021.0 ps, S_addr[0e]=10010100 || T_addr[1e]=xxxxxxxx
time=5031.0 ps, S_addr[0f]=10010040 || T_addr[1f]=xxxxxxxx
DATA MEMORY FILE
time=5041.0 ps, dm_addr[000000c0]=00c3c3c3
time=5051.0 ps, dm_addr[000000c4]=12345678
time=5061.0 ps, dm_addr[000000c8]=89abcdef
time=5071.0 ps, dm_addr[000000cc]=a5a5a5a5
time=5081.0 ps, dm_addr[000000d0]=5a5a5a5a
time=5091.0 ps, dm_addr[000000d4]=2468ace0
time=5101.0 ps, dm_addr[000000d8]=13579bdf
time=5111.0 ps, dm_addr[000000dc]=0f0f0f0f
time=5121.0 ps, dm_addr[000000e0]=f0f0f0f0
time=5131.0 ps, dm_addr[000000e4]=00000009
time=5141.0 ps, dm_addr[000000e8]=0000000a
time=5151.0 ps, dm_addr[000000ec]=0000000b
time=5161.0 ps, dm_addr[000000f0]=0000000c
time=5171.0 ps, dm_addr[000000f4]=0000000d
time=5181.0 ps, dm_addr[000000f8]=fffffff8
time=5191.0 ps, dm_addr[000000fc]=000075cc
```

Module_7 and simulation result:

```
@0
main:       lui  $15, 0x1001
        ori  $15, 0x0000        # LI   R15,  0x10010000  dest data pointer
        lui  $14, 0x1001
        ori  $14, 0x00C0        # LI   R14,  0x100100C0  dest data pointer
        addi $13, $00, 16       # LI   R13,  16          loop counter
        lw   $01, 04($15)       # Load
        lw   $02, 08($15)       #   R01
        lw   $03, 12($15)       #     to
        lw   $04, 16($15)       #      R12
        lw   $05, 20($15)
        lw   $06, 24($15)
        lw   $07, 28($15)
        lw   $08, 32($15)
        lw   $09, 36($15)
        lw   $10, 40($15)
        lw   $11, 44($15)
        lw   $12, 48($15)

jal     mem2memx
        lui  $15, 0xFFFF
        ori  $15, 0xFFFF        # LI   R15,  0xFFFFFFFF  "pass flag"
        break
mem2mem:
        lw   $17, 00($15)       # do
        sw   $17, 00($14)       #   mem t  fer
        addi $15, $15, 04       # bump both s
        addi $14, $14, 04       #   and dest poi
        addi $13, $13, -1       # dec the loop counte
        bne  $13, $00, mem2mem  #   and continue till do
        jr   $31                # return to calling code
        break                   # safety net
```

Load from memory with offset 4 with the content of 12345678 into $r1 to $r12. When execute **jal** instruction, it jump to mem2mem location.

Program executes lw by loading from mem location 00 to location 0xC0. Value of c3c3c3c3 from location 00 is copied to location 0xC0.

Isim result:

```
BREAK INSTRUCTION FETCHED 5025.0 ps
REGISTER' AFTER BREAK
time=5041.0 ps, S_addr[00]=00000000 || T_addr[10]=xxxxxxxx
time=5051.0 ps, S_addr[01]=12345678 || T_addr[11]=000075cc
time=5061.0 ps, S_addr[02]=89abcdef || T_addr[12]=xxxxxxxx
time=5071.0 ps, S_addr[03]=a5a5a5a5 || T_addr[13]=xxxxxxxx
time=5081.0 ps, S_addr[04]=5a5a5a5a || T_addr[14]=xxxxxxxx
time=5091.0 ps, S_addr[05]=2468ace0 || T_addr[15]=xxxxxxxx
time=5101.0 ps, S_addr[06]=13579bdf || T_addr[16]=xxxxxxxx
time=5111.0 ps, S_addr[07]=0f0f0f0f || T_addr[17]=xxxxxxxx
time=5121.0 ps, S_addr[08]=f0f0f0f0 || T_addr[18]=xxxxxxxx
time=5131.0 ps, S_addr[09]=00000009 || T_addr[19]=xxxxxxxx
time=5141.0 ps, S_addr[0a]=0000000a || T_addr[1a]=xxxxxxxx
time=5151.0 ps, S_addr[0b]=0000000b || T_addr[1b]=xxxxxxxx
time=5161.0 ps, S_addr[0c]=0000000c || T_addr[1c]=xxxxxxxx
time=5171.0 ps, S_addr[0d]=00000000 || T_addr[1d]=000003fc
time=5181.0 ps, S_addr[0e]=10010100 || T_addr[1e]=xxxxxxxx
time=5191.0 ps, S_addr[0f]=ffffffff || T_addr[1f]=00000048
DATA MEMORY FILE
time=5201.0 ps, dm_addr[000000c0]=00c3c3c3
time=5211.0 ps, dm_addr[000000c4]=12345678
time=5221.0 ps, dm_addr[000000c8]=89abcdef
time=5231.0 ps, dm_addr[000000cc]=a5a5a5a5
time=5241.0 ps, dm_addr[000000d0]=5a5a5a5a
time=5251.0 ps, dm_addr[000000d4]=2468ace0
time=5261.0 ps, dm_addr[000000d8]=13579bdf
time=5271.0 ps, dm_addr[000000dc]=0f0f0f0f
time=5281.0 ps, dm_addr[000000e0]=f0f0f0f0
time=5291.0 ps, dm_addr[000000e4]=00000009
time=5301.0 ps, dm_addr[000000e8]=0000000a
time=5311.0 ps, dm_addr[000000ec]=0000000b
time=5321.0 ps, dm_addr[000000f0]=0000000c
time=5331.0 ps, dm_addr[000000f4]=0000000d
time=5341.0 ps, dm_addr[000000f8]=fffffff8
time=5351.0 ps, dm_addr[000000fc]=000075cc
```

Program successfully executes when PC counter jump back to return address and execute the lui and ori write to $r15 0xFFFFFFFF

Module_8 and simulation result:

```
@0
main:
        lui  $15, 0x1001
        ori  $15, 0x0000        # $r15 <-- 0x10010000 (src pointer)
        lw   $01, 00($15)       # $r01 <--     25
        lw   $02, 04($15)       # $r02 <--   1000
        lw   $03, 08($15)       # $r03 <--    -25
        lw   $04, 12($15)       # $r04 <--  -1000
        lw   $05, 16($15)       # $r05 <--  25000
        lw   $06, 20($15)       # $r06 <-- -25000
        lw   $07, 24($15)       # $r07 <--     -1
        mult $01, $02
        mflo $08                # rs=pos rt=pos rd=pos
        bne  $05, $08, fail1
        mult $03, $02
        mflo $09                # rs=neg rt=pos rd=neg
        mfhi $10
        bne  $06, $09, fail2L
        bne  $07, $10, fail2H
        mult $01, $04
        mflo $11                # rs=pos rt=neg rd=neg
        mfhi $12
        bne  $06, $11, fail3L
        bne  $07, $12, fail3H
        mult $03, $04
        mflo $13                # rs=neg rt=neg rd=pos
        bne  $05, $13, fail4
pass:
        lui  $14, 0x0000
        ori  $14, 0x0000        # $r14 <-- 0x00000000  (Pass flag)
        break
fail1:
        lui  $14, 0xFFFF
        ori  $14, 0xFFFF        # $r14 <-- 0xFFFFFFFF  (Fail flag 1)
        break
fail2L:
        ui  $14, 0xFFFF
        ori  $14, 0xFFFE        # $r14 <-- 0xFFFFFFFE  (Fail flag 2L)
        break
fail2H:
        lui  $14, 0xFFFF
        ori  $14, 0xFFFD        # $r14 <-- 0xFFFFFFFD  (Fail flag 2H)
         break
fail3L:
        lui  $14, 0xFFFF
        ori  $14, 0xFFFC        # $r14 <-- 0xFFFFFFFC  (Fail flag 3L)
        break
fail3H:
        lui  $14, 0xFFFF
        ori  $14, 0xFFFB        # $r14 <-- 0xFFFFFFFB  (Fail flag 3H)
        break
fail4:
        lui  $14, 0xFFFF
        ori  $14, 0xFFFA        # $r14 <-- 0xFFFFFFFA  (Fail flag 4)
        break
```

Last instrunction to execute when pass all the flag is write $14 =0

Isim result:

```
time=1131.0 ps, S_addr[00]=00000000 || T_addr[10]=xxxxxxxx
time=1141.0 ps, S_addr[01]=00000019 || T_addr[11]=xxxxxxxx
time=1151.0 ps, S_addr[02]=000003e8 || T_addr[12]=xxxxxxxx
time=1161.0 ps, S_addr[03]=ffffffe7 || T_addr[13]=xxxxxxxx
time=1171.0 ps, S_addr[04]=fffffc18 || T_addr[14]=xxxxxxxx
time=1181.0 ps, S_addr[05]=000061a8 || T_addr[15]=xxxxxxxx
time=1191.0 ps, S_addr[06]=ffff9e58 || T_addr[16]=xxxxxxxx
time=1201.0 ps, S_addr[07]=ffffffff || T_addr[17]=xxxxxxxx
time=1211.0 ps, S_addr[08]=000061a8 || T_addr[18]=xxxxxxxx
time=1221.0 ps, S_addr[09]=ffff9e58 || T_addr[19]=xxxxxxxx
time=1231.0 ps, S_addr[0a]=ffffffff || T_addr[1a]=xxxxxxxx
time=1241.0 ps, S_addr[0b]=ffff9e58 || T_addr[1b]=xxxxxxxx
time=1251.0 ps, S_addr[0c]=ffffffff || T_addr[1c]=xxxxxxxx
time=1261.0 ps, S_addr[0d]=000061a8 || T_addr[1d]=000003fc
time=1271.0 ps, S_addr[0e]=00000000 || T_addr[1e]=xxxxxxxx
time=1281.0 ps, S_addr[0f]=10010000 || T_addr[1f]=xxxxxxxx
```

$r1 to $r7 has the memory value load in. and the result of $14 is 000

```
DATA MEMORY FILE
time=1291.0 ps, dm_addr[000000c0]=xxxxxxxx
time=1301.0 ps, dm_addr[000000c4]=xxxxxxxx
time=1311.0 ps, dm_addr[000000c8]=xxxxxxxx
time=1321.0 ps, dm_addr[000000cc]=xxxxxxxx
time=1331.0 ps, dm_addr[000000d0]=xxxxxxxx
time=1341.0 ps, dm_addr[000000d4]=xxxxxxxx
time=1351.0 ps, dm_addr[000000d8]=xxxxxxxx
time=1361.0 ps, dm_addr[000000dc]=xxxxxxxx
time=1371.0 ps, dm_addr[000000e0]=xxxxxxxx
time=1381.0 ps, dm_addr[000000e4]=xxxxxxxx
time=1391.0 ps, dm_addr[000000e8]=xxxxxxxx
time=1401.0 ps, dm_addr[000000ec]=xxxxxxxx
time=1411.0 ps, dm_addr[000000f0]=xxxxxxxx
time=1421.0 ps, dm_addr[000000f4]=xxxxxxxx
time=1431.0 ps, dm_addr[000000f8]=xxxxxxxx
time=1441.0 ps, dm_addr[000000fc]=xxxxxxxx
```

Module_9 and simulation result:

```
@0
main:
        lui  $15, 0x1001
        ori  $15, 0x00C0        # $r15 <-- 0x100100C0  (dest pointer)
        addi $01, $00, -118     # $r01 <-- 0xFFFFFF8A
        addi $02  $00,  138     # $r02 <-- 0x0000008A
        jal  slt_tests
        lui  $13, 0x7788
        ori  $13, 0x7788        # $r13 <-- 0x77887788  (pattern1)
        lui  $12, 0x8877
        ori  $12, 0x8877        # $r12 <-- 0x88778877  (pattern2)
        lui  $11, 0xFFFF
        ori  $11, 0xFFFF        # $r11 <-- 0xFFFFFFFF  (pattern3)
        xor  $10, $13, $12      # $r10 <-- 0xFFFFFFFF
        beq  $10, $11, xor_pass
        addi $14, $00, -5       # fail flag5 r14 <-- 0xFFFF_FFFB
        break
xor_pass:
        and  $09, $13, $12      # $r09 <-- 0x00000000
        beq  $09, $00, and_pass
        addi $14, $00, -6       # fail flag6 r14 <-- 0xFFFF_FFFA
        break
and_pass:
        or   $09, $15, $02      # $r09 <-- 0x100100CA
        lui  $08, 0x1001
        ori  $08, 0x00CA        # $r08 <-- 0x100100CA
        beq  $08, $09, or_pass
        addi $14, $00, -7       # fail flag7 r14 <-- 0xFFFF_FFF9
        break
or_pass:
        nor  $09, $15, $02      # $r09 <-- 0xEFFEFF35
        lui  $08, 0xEFFE
        ori  $08, 0xFF35        # $r08 <-- 0xEFFEFF35
        beq  $08, $09, nor_pass
        addi $14, $00, -8       # fail flag8 r14 <-- 0xFFFF_FFF8
        break
nor_pass:
        sw   $08, 0x10($15)     # M[D0] <-- 0xEFFEFF35
        add  $14, $00, $00      # clear r14 indicating "passed all"
        break                   # should stop here, having # completed all the test
slt_tests:
        slt  $03, $01, $02      # for signed# r01 < r02
        bne  $03, $00, slt1     #   thus, we should branch
        addi $14, $00, -1       # fail flag1 r14 <-- FFFF_FFFF
        break
slt1:
        addi $04, $00, 0xC0   # pass flag1 M[C0] <-- C0
        sw   $04, 0x00($15)
        sltu $03, $02, $01      # for unsigned# r02 < r01
        bne  $03, $00, slt2     #   thus, we should branch
        addi $14, $00, -2       # fail flag2 r14 <-- FFFF_FFFE
        break
slt2:
        addi $05, $00, 0xC4   # pass flag1 M[C4] <-- C4
        sw   $05, 0x04($15)
        slt  $03, $02, $01      # for signed# r02 !< r01
        beq  $03, $00, slt3     #   thus, we should branch
        addi $14, $00, -3       # fail flag3 r14 <-- FFFF_FFFD
        break
slt3:
        addi $06, $00, 0xC8   # pass flag3 M[C8] <-- C8
        sw   $06, 0x08($15)
        sltu $03, $01, $02      # for unsigned# r01 !< r02
        beq  $03, $00, slt4     #   thus, we should branch
        addi $14, $00, -4       # fail flag4 r14 <-- FFFF_FFFC
        break
slt4:
        addi $07, $00, 0xCC     # pass flag4 M[CC] <-- CC
```

The last instruction to execute when all flag is passed is sw to D0 with value

When slt test is pass a value is stored to M[c0] with C0. The same pattern is done, fill mem with value of C0, c4, C8, and CC

```
        sw   $07, 0x0C($15)
        jr   $31                # return from subroutine
```

Isim result:

```
BREAK INSTRUCTION FETCHED 1745.0 ps
REGISTER' AFTER BREAK
time=1761.0 ps, S_addr[00]=00000000 || T_addr[10]=xxxxxxxx
time=1771.0 ps, S_addr[01]=ffffff8a || T_addr[11]=xxxxxxxx
time=1781.0 ps, S_addr[02]=0000008a || T_addr[12]=xxxxxxxx
time=1791.0 ps, S_addr[03]=00000000 || T_addr[13]=xxxxxxxx
time=1801.0 ps, S_addr[04]=000000c0 || T_addr[14]=xxxxxxxx
time=1811.0 ps, S_addr[05]=000000c4 || T_addr[15]=xxxxxxxx
time=1821.0 ps, S_addr[06]=000000c8 || T_addr[16]=xxxxxxxx
time=1831.0 ps, S_addr[07]=000000cc || T_addr[17]=xxxxxxxx
time=1841.0 ps, S_addr[08]=effeff35 || T_addr[18]=xxxxxxxx
time=1851.0 ps, S_addr[09]=effeff35 || T_addr[19]=xxxxxxxx
time=1861.0 ps, S_addr[0a]=ffffffff || T_addr[1a]=xxxxxxxx
time=1871.0 ps, S_addr[0b]=ffffffff || T_addr[1b]=xxxxxxxx
time=1881.0 ps, S_addr[0c]=88778877 || T_addr[1c]=xxxxxxxx
time=1891.0 ps, S_addr[0d]=77887788 || T_addr[1d]=000003fc
time=1901.0 ps, S_addr[0e]=00000000 || T_addr[1e]=xxxxxxxx
time=1911.0 ps, S_addr[0f]=100100c0 || T_addr[1f]=00000014

DATA MEMORY FILE
time=1921.0 ps, dm_addr[000000c0]=000000c0
time=1931.0 ps, dm_addr[000000c4]=000000c4
time=1941.0 ps, dm_addr[000000c8]=000000c8
time=1951.0 ps, dm_addr[000000cc]=000000cc
time=1961.0 ps, dm_addr[000000d0]=effeff35
time=1971.0 ps, dm_addr[000000d4]=xxxxxxxx
time=1981.0 ps, dm_addr[000000d8]=xxxxxxxx
time=1991.0 ps, dm_addr[000000dc]=xxxxxxxx
time=2001.0 ps, dm_addr[000000e0]=xxxxxxxx
time=2011.0 ps, dm_addr[000000e4]=xxxxxxxx
time=2021.0 ps, dm_addr[000000e8]=xxxxxxxx
time=2031.0 ps, dm_addr[000000ec]=xxxxxxxx
time=2041.0 ps, dm_addr[000000f0]=xxxxxxxx
time=2051.0 ps, dm_addr[000000f4]=xxxxxxxx
time=2061.0 ps, dm_addr[000000f8]=xxxxxxxx
time=2071.0 ps, dm_addr[000000fc]=xxxxxxxx
```

slt test

Module_10 and simulation result:

```
@0
main:   lui  $15, 0x1001
        ori  $15, 0x0000        # $r15 <-- 0x10010000  (source pointer)
        lw   $01, 00($15)       # $r01 <--  264465
        lw   $02, 04($15)       # $r02 <--    1000
        lw   $03, 08($15)       # $r03 <-- -264465
        lw   $04, 12($15)       # $r04 <--   -1000
        lw   $05, 16($15)       # $r05 <--     264   Quot1,4    w01 div w02, w03 div w04
        lw   $06, 20($15)       # $r06 <--     465   Rem 1,3    w01 rem w02, w01 rem w04
        lw   $07, 24($15)       # $r07 <--    -264   Quot2,3    w03 div w02, w01 div w04
        lw   $08, 28($15)       # $r08 <--    -465   Rem 2,4    w03 re0 w02, w03 rem w04
        div  $01, $02
        mflo $09                # rs=pos / rt=pos, rem=pos quot=pos
        mfhi $10
        bne  $09, $05, fail1Q
        bne  $10, $06, fail1R
        div  $03, $02
        mflo $09                # rs=neg / rt=pos, rem=neg quot=neg
        mfhi $10
        bne  $09, $07, fail2Q
        bne  $10, $08, fail2R
        div  $01, $04
        mflo $09                # rs=pos / rt=neg, rem=pos quot=neg
        mfhi $10
        bne  $09, $07, fail3Q
        bne  $10, $06, fail3R
        div  $03, $04
        mflo $09                # rs=neg / rt=neg, rem=neg quot=pos
        mfhi $10
        bne  $09, $05, fail4Q
        bne  $10, $08, fail4R
pass:
        lui  $11, 0x0000
        ori  $11, 0x0000        # $r11 <-- 0x00000000  (Pass flag)
        add  $12, $00, $11      # $r12 <-- Pass
        add  $13, $00, $11      # $r13 <-- Pass
        add  $14, $00, $11      # $r14 <-- Pass
        break
fail1Q:
        lui  $14, 0xFFFF
        ori  $14, 0xFFFF        # $r14 <-- 0xFFFFFFFF  (Fail flag 1 Quot)
        break
fail1R:
        lui  $14, 0xFFFF
        ori  $14, 0xFFFE        # $r14 <-- 0xFFFFFFFE  (Fail flag 1 Rem)
        break
fail2Q:
        lui  $14, 0xFFFF
        ori  $14, 0xFFFD        # $r14 <-- 0xFFFFFFFD  (Fail flag 2 Quot)
        break
fail2R:
        lui  $14, 0xFFFF
        ori  $14, 0xFFFC        # $r14 <-- 0xFFFFFFFC  (Fail flag 2 Rem)
        break
fail3Q:
        lui  $14, 0xFFFF
        ori  $14, 0xFFFB        # $r14 <-- 0xFFFFFFFB  (Fail flag 3 Quot)
        break
fail3R:
        lui  $14, 0xFFFF
        ori  $14, 0xFFFA        # $r14 <-- 0xFFFFFFFA  (Fail flag 3 Rem)
        break
fail4Q:
        lui  $14, 0xFFFF
        ori  $14, 0xFFF9        # $r14 <-- 0xFFFFFFF9  (Fail flag 4 Quot)
        break
fail4R:
        lui  $14, 0xFFFF
        ori  $14, 0xFFF8        # $r14 <-- 0xFFFFFFF8  (Fail flag 4 Rem)
        break
```

Registe $1 to $8 has the values from mem start from 0.

All flags are passed when $11, $12, $13, $14 are all 000

Isim result:

```
BREAK INSTRUCTION FETCHED 1425.0 ps
```

```
REGISTER' AFTER BREAK
time=1441.0 ps, S_addr[00]=00000000 || T_addr[10]=xxxxxxxx
time=1451.0 ps, S_addr[01]=00040911 || T_addr[11]=xxxxxxxx
time=1461.0 ps, S_addr[02]=000003e8 || T_addr[12]=xxxxxxxx
time=1471.0 ps, S_addr[03]=fffbf6ef || T_addr[13]=xxxxxxxx
time=1481.0 ps, S_addr[04]=fffffc18 || T_addr[14]=xxxxxxxx
time=1491.0 ps, S_addr[05]=00000108 || T_addr[15]=xxxxxxxx
time=1501.0 ps, S_addr[06]=000001d1 || T_addr[16]=xxxxxxxx
time=1511.0 ps, S_addr[07]=fffffef8 || T_addr[17]=xxxxxxxx
time=1521.0 ps, S_addr[08]=fffffe2f || T_addr[18]=xxxxxxxx
time=1531.0 ps, S_addr[09]=00000108 || T_addr[19]=xxxxxxxx
time=1541.0 ps, S_addr[0a]=fffffe2f || T_addr[1a]=xxxxxxxx
time=1551.0 ps, S_addr[0b]=00000000 || T_addr[1b]=xxxxxxxx
time=1561.0 ps, S_addr[0c]=00000000 || T_addr[1c]=xxxxxxxx
time=1571.0 ps, S_addr[0d]=00000000 || T_addr[1d]=000003fc
time=1581.0 ps, S_addr[0e]=00000000 || T_addr[1e]=xxxxxxxx
time=1591.0 ps, S_addr[0f]=10010000 || T_addr[1f]=xxxxxxxx

DATA MEMORY FILE
time=1601.0 ps, dm_addr[000000c0]=xxxxxxxx
time=1611.0 ps, dm_addr[000000c4]=xxxxxxxx
time=1621.0 ps, dm_addr[000000c8]=xxxxxxxx
time=1631.0 ps, dm_addr[000000cc]=xxxxxxxx
time=1641.0 ps, dm_addr[000000d0]=xxxxxxxx
time=1651.0 ps, dm_addr[000000d4]=xxxxxxxx
time=1661.0 ps, dm_addr[000000d8]=xxxxxxxx
time=1671.0 ps, dm_addr[000000dc]=xxxxxxxx
time=1681.0 ps, dm_addr[000000e0]=xxxxxxxx
time=1691.0 ps, dm_addr[000000e4]=xxxxxxxx
time=1701.0 ps, dm_addr[000000e8]=xxxxxxxx
time=1711.0 ps, dm_addr[000000ec]=xxxxxxxx
time=1721.0 ps, dm_addr[000000f0]=xxxxxxxx
time=1731.0 ps, dm_addr[000000f4]=xxxxxxxx
time=1741.0 ps, dm_addr[000000f8]=xxxxxxxx
time=1751.0 ps, dm_addr[000000fc]=xxxxxxxx
```

Module_11 and result of simulation:

```
@0
main:   lui  $15, 0x1001
        ori  $15, 0x00C0          # $r15 <-- 0x100100C0  (dest pointer)
        addi $01, $00, -118       # $r01 <-- 0xFFFFFF8A
        addi $02  $00,  138       # $r02 <-- 0x0000008A
        jal  sltiu_tests
        lui  $13, 0xFFFF
        ori  $13, 0x5555          # $r13 <-- 0xFFFF5555  (pattern1)
        lui  $12, 0xFFFF
        ori  $12, 0xFAF5          # $r12 <-- 0xFFFFFAF5  (pattern2)
        lui  $11, 0xFFFF
        ori  $11, 0xFFFF          # $r11 <-- 0xFFFFFFFF  (pattern3)
        lui  $10, 0x0000
        ori  $10, 0xF0F0          # $r10 <-- 0x0000F0F0  (pattern4)
        xori $09, $13, 0xAAAA     # $r09 <-- 0xFFFFFFFF
        sub  $08, $09, $11        # $r00 <-- 0
        beq  $08, $00, xor_p1     # should branch
        addi $14, $00, -7         # fail flag7 r14 <-- FFFF_FFF9
        break
xor_p1:
        andi $07, $12, 0xF5FA     # $r07 <-- 0x0000F0F0
        beq  $08, $00, xor_p2     # should branch
        addi $14, $00, -8         # fail flag8 r14 <-- FFFF_FFF8
        break
xor_p2:
        sw   $01, 0x18($15)       # M[D8] <-- FFFFFF8A
        break                     # should stop here, having
                                  #   completed all the tests

sltiu_tests:
        sltiu  $03, $01, -117     # for unsigned# r01 < se(0xFF8B)
        bne    $03, $00, slt1_p1  #   thus, we should branch
        addi   $14, $00, -1       # fail flag1 r14 <-- FFFF_FFFF
        break
slt1_p1:
        addi   $04, $00, 0xC0     # pass flag1 M[C0] <-- C0
        sw     $04, 0x00($15)
        sltiu  $03, $01, -119     # for unsigned# r01 !< se(0xFF89)
        beq    $03, $00, slt_p2   #   thus, we should branch
        addi   $14, $00, -2       # fail flag2 r14 <-- FFFF_FFFE
        break
slt_p2:
         addi   $05, $00, 0xC4     # pass flag2 M[C4] <-- C4
        sw     $05, 0x04($15)
        sltiu  $03, $01, -118     # for unsigned# r01 !< se(0xFF8A)
        beq    $03, $00, slt_p3   #   thus, we should branch
        addi   $14, $00, -3       # fail flag3 r14 <-- FFFF_FFFD
        break
slt_p3:
        addi   $06, $00, 0xC8     # pass flag3 M[C8] <-- C8
        sw     $06, 0x08($15)
        sltiu  $03, $02, 0x008B   # for unsigned# r02 < se(0x008B)
        bne    $03, $00, slt1_p4  #   thus, we should branch
        addi   $14, $00, -4       # fail flag4 r14 <-- FFFF_FFFC
        break
slt1_p4:
        addi   $07, $00, 0xCC     # pass flag4 M[CC] <-- CC
        sw     $07, 0x0C($15)
        sltiu  $03, $02, 0x0089   # for unsigned# r02 !< se(0x0089)
        beq    $03, $00, slt_p5   #   thus, we should branch
        addi   $14, $00, -5       # fail flag5 r14 <-- FFFF_FFFB
        break
slt_p5:
        addi   $08, $00, 0xD0     # pass flag5 M[D0] <-- D0
        sw     $08 0x10($15)
        sltiu  $03, $02, 0x008A   # for unsigned# r02 !< se(0x008A)
         beq    $03, $00, slt_p6   #   thus, we should branch
        addi   $14, $00, -6       # fail flag6 r14 <-- FFFF_FFFA
        break
slt_p6:
        addi   $06, $00, 0xD4     # pass flag6 M[D4] <-- D4
        sw     $06, 0x14($15)
        addi   $14, $00, 0        # set $r14 to 0000_0000
```

$r9==ffffffff and $r11==ffffffff, so $r8==0. Beq is taken

When pass all flag, the last instruction to excute is to store ffffff8a at loacation d8

When pass a flag a value is store in mem location starting from M[c0] to M[d4] with the last value as D0.

```
        jr   $31                    # return from subroutine
```

Isim result:

```
BREAK INSTRUCTION FETCHED 1905.0 ps
REGISTER' AFTER BREAK
time=1921.0 ps, S_addr[00]=00000000 || T_addr[10]=xxxxxxxx
time=1931.0 ps, S_addr[01]=ffffff8a || T_addr[11]=xxxxxxxx
time=1941.0 ps, S_addr[02]=0000008a || T_addr[12]=xxxxxxxx
time=1951.0 ps, S_addr[03]=00000000 || T_addr[13]=xxxxxxxx
time=1961.0 ps, S_addr[04]=000000c0 || T_addr[14]=xxxxxxxx
time=1971.0 ps, S_addr[05]=000000c4 || T_addr[15]=xxxxxxxx
time=1981.0 ps, S_addr[06]=000000d4 || T_addr[16]=xxxxxxxx
time=1991.0 ps, S_addr[07]=0000f0f0 || T_addr[17]=xxxxxxxx
time=2001.0 ps, S_addr[08]=00000000 || T_addr[18]=xxxxxxxx
time=2011.0 ps, S_addr[09]=ffffffff || T_addr[19]=xxxxxxxx
time=2021.0 ps, S_addr[0a]=0000f0f0 || T_addr[1a]=xxxxxxxx
time=2031.0 ps, S_addr[0b]=ffffffff || T_addr[1b]=xxxxxxxx
time=2041.0 ps, S_addr[0c]=fffffaf5 || T_addr[1c]=xxxxxxxx
time=2051.0 ps, S_addr[0d]=ffff5555 || T_addr[1d]=000003fc
time=2061.0 ps, S_addr[0e]=00000000 || T_addr[1e]=xxxxxxxx
time=2071.0 ps, S_addr[0f]=100100c0 || T_addr[1f]=00000014

DATA MEMORY FILE
time=2081.0 ps, dm_addr[000000c0]=000000c0
time=2091.0 ps, dm_addr[000000c4]=000000c4
time=2101.0 ps, dm_addr[000000c8]=000000c8
time=2111.0 ps, dm_addr[000000cc]=000000cc
time=2121.0 ps, dm_addr[000000d0]=000000d0
time=2131.0 ps, dm_addr[000000d4]=000000d4
time=2141.0 ps, dm_addr[000000d8]=ffffff8a
time=2151.0 ps, dm_addr[000000dc]=xxxxxxxx
time=2161.0 ps, dm_addr[000000e0]=xxxxxxxx
time=2171.0 ps, dm_addr[000000e4]=xxxxxxxx
time=2181.0 ps, dm_addr[000000e8]=xxxxxxxx
time=2191.0 ps, dm_addr[000000ec]=xxxxxxxx
time=2201.0 ps, dm_addr[000000f0]=xxxxxxxx
time=2211.0 ps, dm_addr[000000f4]=xxxxxxxx
time=2221.0 ps, dm_addr[000000f8]=xxxxxxxx
time=2231.0 ps, dm_addr[000000fc]=xxxxxxxx
```

Value stored in Mem
when a flag is passed.

Module_12 and result of simulation:

```
@0
main:
            lui  $15, 0x1001
            ori  $15, 0x00C0        # $r15 <-- 0x100100C0  (dest pointer)

            addi $01, $00, -118     # $r01 <-- 0xFFFFFF8A
            addi $02  $00,  138     # $r02 <-- 0x0000008A
            jal  blt_tests
            sw   $01, 0x18($15)     # M[D8] <-- 0xFFFFFF8A
            sw   $02, 0x1C($15)     # M[DC] <-- 0x0000008A
            break

blt_tests:  blez $01, blez_p1       # this should branch
            addi $14, $00, -1       # fail flag1 r14 <-- FFFF_FFFF
            break
blez_p1:    addi $03, $00, 0xC0     # pass flag1 M[C0] <-- C0
            sw   $03, 0x00($15)
            blez $02, blez_f2       # this should not branch
            addi $04, $00, 0xC4     # pass flag2 M[C4] <-- C4
            sw   $04, 0x04($15)
            j    blez_p2
blez_f2:    addi $14, $00, -2       # fail flag2 r14 <-- FFFF_FFFE
            break
blez_p2:    blez $0, blez_p3        # this should branch
            addi $14, $00, -3       # fail flag3 r14 <-- FFFF_FFFD
            break
blez_p3:    addi $05, $00, 0xC8     # pass flag3 M[C8] <-- C8
            sw   $05, 0x08($15)

            bgtz $02, bgtz_p1       # this should pass
            addi $14, $00, -4       # fail flag3 r14 <-- FFFF_FFFC
            break
bgtz_p1:    addi $06, $00, 0xCC     # pass flag4 M[C0] <-- CC
            sw   $06, 0x0C($15)
            bgtz $01, bgtz_f2       # this should not branch
            addi $07, $00, 0xD0     # pass flag5 M[D0] <-- D0
            sw   $07, 0x10($15)
            j    bgtz_p2
bgtz_f2:    addi $14, $00, -5       # fail flag5 r14 <-- FFFF_FFFB
            break
bgtz_p2:    bgtz $01, bgtz_f3       # this should not branch
            addi $08, $00, 0xD4     # pass flag6 M[D0] <-- D4
            sw   $08, 0x14($15)
            j    bgtz_p3
bgtz_f3:    addi $14, $00, -6       # fail flag6 r14 <-- FFFF_FFFA
            break
bgtz_p3:    addi $14, $00, 0        # set $r14 to 0000_0000
            jr   $31                # return from subroutine
```

$01 <0 so branch. and store "c0" to c"c0"

The same parttern of execution if pass the flag store value: C4, C8, CC, d0 and d4 is the last store.

Before goes to return address, $r14=0. At the return address store two more value ffffff8a and 0000008a to location d8 and dc

Isim result:

```
BREAK INSTRUCTION FETCHED 1235.0 ps
REGISTER' AFTER BREAK
time=1251.0 ps, S_addr[00]=00000000 || T_addr[10]=xxxxxxxx
time=1261.0 ps, S_addr[01]=ffffff8a || T_addr[11]=xxxxxxxx
time=1271.0 ps, S_addr[02]=0000008a || T_addr[12]=xxxxxxxx
time=1281.0 ps, S_addr[03]=000000c0 || T_addr[13]=xxxxxxxx
time=1291.0 ps, S_addr[04]=000000c4 || T_addr[14]=xxxxxxxx
time=1301.0 ps, S_addr[05]=000000c8 || T_addr[15]=xxxxxxxx
time=1311.0 ps, S_addr[06]=000000cc || T_addr[16]=xxxxxxxx
time=1321.0 ps, S_addr[07]=000000d0 || T_addr[17]=xxxxxxxx
time=1331.0 ps, S_addr[08]=000000d4 || T_addr[18]=xxxxxxxx
time=1341.0 ps, S_addr[09]=xxxxxxxx || T_addr[19]=xxxxxxxx
time=1351.0 ps, S_addr[0a]=xxxxxxxx || T_addr[1a]=xxxxxxxx
time=1361.0 ps, S_addr[0b]=xxxxxxxx || T_addr[1b]=xxxxxxxx
time=1371.0 ps, S_addr[0c]=xxxxxxxx || T_addr[1c]=xxxxxxxx
time=1381.0 ps, S_addr[0d]=xxxxxxxx || T_addr[1d]=000003fc
time=1391.0 ps, S_addr[0e]=00000000 || T_addr[1e]=xxxxxxxx
time=1401.0 ps, S_addr[0f]=100100c0 || T_addr[1f]=00000014
```

```
DATA MEMORY FILE
time=1411.0 ps, dm_addr[000000c0]=000000c0
time=1421.0 ps, dm_addr[000000c4]=000000c4
time=1431.0 ps, dm_addr[000000c8]=000000c8
time=1441.0 ps, dm_addr[000000cc]=000000cc
time=1451.0 ps, dm_addr[000000d0]=000000d0
time=1461.0 ps, dm_addr[000000d4]=000000d4
time=1471.0 ps, dm_addr[000000d8]=ffffff8a
time=1481.0 ps, dm_addr[000000dc]=0000008a
time=1491.0 ps, dm_addr[000000e0]=xxxxxxxx
time=1501.0 ps, dm_addr[000000e4]=xxxxxxxx
time=1511.0 ps, dm_addr[000000e8]=xxxxxxxx
time=1521.0 ps, dm_addr[000000ec]=xxxxxxxx
time=1531.0 ps, dm_addr[000000f0]=xxxxxxxx
time=1541.0 ps, dm_addr[000000f4]=xxxxxxxx
time=1551.0 ps, dm_addr[000000f8]=xxxxxxxx
time=1561.0 ps, dm_addr[000000fc]=xxxxxxxx
```

Module_13 and result of simulation:

```
@0
main:
setie
lui  $01, 0x1234
ori  $01, 0x5678      # LI  R01,  0x12345678
lui  $02, 0x8765
ori  $02, 0x4321      # LI  R02,  0x87654321
lui  $03, 0xABCD
ori  $03, 0xEF01      # LI  R03,  0xABCDEF01
lui  $04, 0x01FE
ori  $04, 0xDCBA      # LI  R04,  0x01FEDCBA
lui  $05, 0x5A5A
ori  $05, 0x5A5A      # LI  R05,  0x5A5A5A5A
lui  $06, 0xFFFF
ori  $06, 0xFFFF      # LI  R06,  0xFFFFFFFF
lui  $07, 0xFFFF
ori  $07, 0xFF00      # LI  R07,  0xFFFFFF00

add  $08, $06, $07
add  $09, $06, $08
add  $10, $06, $09
add  $11, $06, $10
add  $12, $06, $11
add  $13, $06, $12
add  $14, $06, $13
add  $15, $06, $14

lui  $07, 0x1001
ori  $07, 0x03F0      # LI   R07,  0x100103F0
sw   $15, 0($07)      # ST  [R07], R15
break

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

out_IO:
output $17, 0($16)      # output  [R16], R17
sra    $17, $17, 2      # change the pattern by shifting twice
addi   $16, $16, 4      # increment the memory pointer 4 bytes
addi   $18, $18, -1     # decrement the loop counter
bne    $18, $00, out_IO #  and jmp to top if not finished

lui    $16, 0x1001      #load source IO address
ori    $16, 0x00C0      #  0x100100C0 into r16
input  $19,  0($16)     #  and input from 6
input  $20,  4($16)     #    the IO locations,
input  $21,  8($16)     #    starting from 0xC0
input  $22, 12($16)
input  $23, 16($16)
input  $24, 20($16)
jr     $31              # return from interrupt (v1, using $ra)
```

After finish writing to IO, $r18 == 0. The last instruction is sw $r15=fffffef8 to location 3F0.

Isim result:

```
BREAK INSTRUCTION FETCHED 5005.0 ps
REGISTER' AFTER BREAK
time=5021.0 ps, S_addr[00]=00000000 || T_addr[10]=100100c0
time=5031.0 ps, S_addr[01]=12345678 || T_addr[11]=ffffffff
```

```
time=5041.0 ps, S_addr[02]=87654321 || T_addr[12]=00000000
time=5051.0 ps, S_addr[03]=abcdef01 || T_addr[13]=8000ffff
time=5061.0 ps, S_addr[04]=01fedcba || T_addr[14]=e0003fff
time=5071.0 ps, S_addr[05]=5a5a5a5a || T_addr[15]=f8000fff
time=5081.0 ps, S_addr[06]=ffffffff || T_addr[16]=fe0003ff
time=5091.0 ps, S_addr[07]=100103f0 || T_addr[17]=ff8000ff
time=5101.0 ps, S_addr[08]=ffffffeff || T_addr[18]=ffe0003f
time=5111.0 ps, S_addr[09]=fffffefe || T_addr[19]=xxxxxxxx
time=5121.0 ps, S_addr[0a]=fffffefd || T_addr[1a]=xxxxxxxx
time=5131.0 ps, S_addr[0b]=fffffefc || T_addr[1b]=xxxxxxxx
time=5141.0 ps, S_addr[0c]=fffffefb || T_addr[1c]=xxxxxxxx
time=5151.0 ps, S_addr[0d]=fffffefa || T_addr[1d]=000003f4
time=5161.0 ps, S_addr[0e]=fffffef9 || T_addr[1e]=xxxxxxxx
time=5171.0 ps, S_addr[0f]=fffffef8 || T_addr[1f]=00000044
```

Return address is not read from the stack, so $sp pointer keep the address. PC is stored both in $ra(31) and on the stack based on this CPU architecture.

```
DATA MEMORY FILE
time=5181.0 ps, dm_addr[00000000]=00c3c3c3
time=5191.0 ps, dm_addr[00000004]=12345678
time=5201.0 ps, dm_addr[00000008]=89abcdef
time=5211.0 ps, dm_addr[0000000c]=a5a5a5a5
time=5221.0 ps, dm_addr[00000010]=5a5a5a5a
time=5231.0 ps, dm_addr[00000014]=2468ace0
time=5241.0 ps, dm_addr[00000018]=13579bdf
time=5251.0 ps, dm_addr[0000001c]=0f0f0f0f
time=5261.0 ps, dm_addr[00000020]=f0f0f0f0
time=5271.0 ps, dm_addr[00000024]=00000009
time=5281.0 ps, dm_addr[00000028]=0000000a
time=5291.0 ps, dm_addr[0000002c]=0000000b
time=5301.0 ps, dm_addr[00000030]=0000000c
time=5311.0 ps, dm_addr[00000034]=0000000d
time=5321.0 ps, dm_addr[00000038]=fffffff8
time=5331.0 ps, dm_addr[0000003c]=000075cc
time=5341.0 ps, dm_addr[00000040]=xxxxxxxx
time=5351.0 ps, dm_addr[00000044]=xxxxxxxx
time=5361.0 ps, dm_addr[00000048]=xxxxxxxx
time=5371.0 ps, dm_addr[0000004c]=xxxxxxxx

DISPLAY MEMORY AT LOCATION [3F0:3F3] =
time=5371.0 ps M[3F0]=fffffef8
M[3F8:3FB]=0000001a
M[3F4:3F7]=00000044

DISPLAY I/O module from 0C0 to 104:
time=5381.0 ps, io_addr[000000b0]=xxxxxxxx
time=5391.0 ps, io_addr[000000b4]=xxxxxxxx
time=5401.0 ps, io_addr[000000b8]=xxxxxxxx
time=5411.0 ps, io_addr[000000bc]=xxxxxxxx
time=5421.0 ps, io_addr[000000c0]=8000ffff
time=5431.0 ps, io_addr[000000c4]=e0003fff
time=5441.0 ps, io_addr[000000c8]=f8000fff
time=5451.0 ps, io_addr[000000cc]=fe0003ff
time=5461.0 ps, io_addr[000000d0]=ff8000ff
time=5471.0 ps, io_addr[000000d4]=ffe0003f
time=5481.0 ps, io_addr[000000d8]=fff8000f
time=5491.0 ps, io_addr[000000dc]=fffe0003
time=5501.0 ps, io_addr[000000e0]=ffff8000
time=5511.0 ps, io_addr[000000e4]=ffffe000
time=5521.0 ps, io_addr[000000e8]=fffff800
time=5531.0 ps, io_addr[000000ec]=fffffe00
time=5541.0 ps, io_addr[000000f0]=ffffff80
time=5551.0 ps, io_addr[000000f4]=fffffffe0
time=5561.0 ps, io_addr[000000f8]=fffffff8
time=5571.0 ps, io_addr[000000fc]=ffffffffe
time=5581.0 ps, io_addr[00000100]=xxxxxxxx
```

Module_14 and result of simulation:

```
@0
main:
setie
lui  $01, 0x1234
ori  $01, 0x5678      # LI  R01,  0x12345678
lui  $02, 0x8765
ori  $02, 0x4321      # LI  R02,  0x87654321
lui  $03, 0xABCD
ori  $03, 0xEF01      # LI  R03,  0xABCDEF01
lui  $04, 0x01FE
ori  $04, 0xDCBA      # LI  R04,  0x01FEDCBA
lui  $05, 0x5A5A
ori  $05, 0x5A5A      # LI  R05,  0x5A5A5A5A
lui  $06, 0xFFFF
ori  $06, 0xFFFF      # LI  R06,  0xFFFFFFFF
lui  $07, 0xFFFF
ori  $07, 0xFF00      # LI  R07,  0xFFFFFF00

add  $08, $06, $07
add  $09, $06, $08
add  $10, $06, $09
add  $11, $06, $10
add  $12, $06, $11
add  $13, $06, $12
add  $14, $06, $13
add  $15, $06, $14

lui  $07, 0x1001
ori  $07, 0x03F0      # LI   R07,  0x100103F0
sw   $15, 0($07)      # ST  [R07], R15
break
@200
//***********************************************
// In this ISR, is implement writing
// some patterns to the IO space, and then
// reading them back.
// Note: this "ISR" expects the "return address"
// to have been saved on the stack (not $ra)
//***********************************************
isr:
lui    $16, 0x1001      #load destination IO address
ori    $16, 0x00C0      #0x100100C0 into r16
lui    $17, 0x8000      #initialize the pattern of
ori    $17, 0xFFFF      #0x8000FFFF into r17
addi   $18, $0, 0x10    #loop counter set to 16

out_IO:
output $17, 0($16)      # output  [R16], R17
sra    $17, $17, 2      # change the pattern by shifting twice
addi   $16, $16, 4      # increment the memory pointer 4 bytes
addi   $18, $18, -1     # decrement the loop counter
bne    $18, $00, out_IO # and jmp to top if not finished

lui    $16, 0x1001      #load source IO address
ori    $16, 0x00C0      #0x100100C0 into r16
input  $19,  0($16)     #and input from 6
input  $20,  4($16)     #the IO locations,
input  $21,  8($16)     #starting from 0xC0
input  $22, 12($16)
input  $23, 16($16)
input  $24, 20($16)
reti                    # return from interrupt (v2, using M[sp] as saved PC)
                        # Note opcode=0x1E and $rs=0x1D, i.e. $sp
```

After finish writing to IO, $r18 == 0. The last instruction is sw $r15=fffffef8 to location 3F0.

Isim result:

```
BREAK INSTRUCTION FETCHED 5055.0 ps
REGISTER' AFTER BREAK
time=5071.0 ps, S_addr[00]=00000000 || T_addr[10]=100100c0
time=5081.0 ps, S_addr[01]=12345678 || T_addr[11]=ffffffff
time=5091.0 ps, S_addr[02]=87654321 || T_addr[12]=00000000
time=5101.0 ps, S_addr[03]=abcdef01 || T_addr[13]=8000ffff
time=5111.0 ps, S_addr[04]=01fedcba || T_addr[14]=e0003fff
time=5121.0 ps, S_addr[05]=5a5a5a5a || T_addr[15]=f8000fff
time=5131.0 ps, S_addr[06]=ffffffff || T_addr[16]=fe0003ff
time=5141.0 ps, S_addr[07]=100103f0 || T_addr[17]=ff8000ff
time=5151.0 ps, S_addr[08]=ffffffff || T_addr[18]=ffe0003f
time=5161.0 ps, S_addr[09]=fffffefe || T_addr[19]=xxxxxxxx
time=5171.0 ps, S_addr[0a]=fffffefd || T_addr[1a]=xxxxxxxx
time=5181.0 ps, S_addr[0b]=fffffefc || T_addr[1b]=xxxxxxxx
time=5191.0 ps, S_addr[0c]=fffffefb || T_addr[1c]=xxxxxxxx
time=5201.0 ps, S_addr[0d]=fffffefa || T_addr[1d]=000003fc
time=5211.0 ps, S_addr[0e]=fffffef9 || T_addr[1e]=xxxxxxxx
time=5221.0 ps, S_addr[0f]=fffffef8 || T_addr[1f]=00000044

DATA MEMORY FILE
time=5231.0 ps, dm_addr[00000000]=00c3c3c3
time=5241.0 ps, dm_addr[00000004]=12345678
time=5251.0 ps, dm_addr[00000008]=89abcdef
time=5261.0 ps, dm_addr[0000000c]=a5a5a5a5
time=5271.0 ps, dm_addr[00000010]=5a5a5a5a
time=5281.0 ps, dm_addr[00000014]=2468ace0
time=5291.0 ps, dm_addr[00000018]=13579bdf
time=5301.0 ps, dm_addr[0000001c]=0f0f0f0f
time=5311.0 ps, dm_addr[00000020]=f0f0f0f0
time=5321.0 ps, dm_addr[00000024]=00000009
time=5331.0 ps, dm_addr[00000028]=0000000a
time=5341.0 ps, dm_addr[0000002c]=0000000b
time=5351.0 ps, dm_addr[00000030]=0000000c
time=5361.0 ps, dm_addr[00000034]=0000000d
time=5371.0 ps, dm_addr[00000038]=ffffffff8
time=5381.0 ps, dm_addr[0000003c]=000075cc
time=5391.0 ps, dm_addr[00000040]=xxxxxxxx

DISPLAY MEMORY AT LOCATION [3F0:3F3] =
time=5861.0 ps M[3F0]=fffffef8
M[3F8:3FB]=0000001a
M[3F4:3F7]=00000044

DISPLAY I/O module from 0C0 to 104:
time=5871.0 ps, io_addr[000000b0]=xxxxxxxx
time=5881.0 ps, io_addr[000000b4]=xxxxxxxx
time=5891.0 ps, io_addr[000000b8]=xxxxxxxx
time=5901.0 ps, io_addr[000000bc]=xxxxxxxx
time=5911.0 ps, io_addr[000000c0]=8000ffff
time=5921.0 ps, io_addr[000000c4]=e0003fff
time=5931.0 ps, io_addr[000000c8]=f8000fff
time=5941.0 ps, io_addr[000000cc]=fe0003ff
time=5951.0 ps, io_addr[000000d0]=ff8000ff
time=5961.0 ps, io_addr[000000d4]=ffe0003f
time=5971.0 ps, io_addr[000000d8]=fff8000f
time=5981.0 ps, io_addr[000000dc]=fffe0003
time=5991.0 ps, io_addr[000000e0]=ffff8000
time=6001.0 ps, io_addr[000000e4]=ffffe000
time=6011.0 ps, io_addr[000000e8]=fffff800
time=6021.0 ps, io_addr[000000ec]=ffffffe00
time=6031.0 ps, io_addr[000000f0]=ffffff80
time=6041.0 ps, io_addr[000000f4]=fffffffe0
time=6051.0 ps, io_addr[000000f8]=ffffffff8
time=6061.0 ps, io_addr[000000fc]=ffffffffe
time=6071.0 ps, io_addr[00000100]=xxxxxxxx
```

After returning from interrupt $07 is changed from ffffffff00 to 100103f0. And the stack kpointer is restored back to 3FC.

Enhenced Capability module:

```
@0
main:
setie
lui  $01, 0x1234
ori  $01, 0x5678       # LI  R01,  0x12345678
lui  $02, 0x8765
ori  $02, 0x4321       # LI  R02,  0x87654321
lui  $03, 0xABCD
ori  $03, 0xEF01       # LI  R03,  0xABCDEF01
lui  $04, 0x01FE
ori  $04, 0xDCBA       # LI  R04,  0x01FEDCBA
lui  $05, 0x5A5A
ori  $05, 0x5A5A       # LI  R05,  0x5A5A5A5A
lui  $06, 0xFFFF
ori  $06, 0xFFFF       # LI  R06,  0xFFFFFFFF
lui  $07, 0xFFFF
ori  $07, 0xFF00       # LI  R07,  0xFFFFFF00
add  $08, $06, $07
add  $09, $06, $08
add  $10, $06, $09
add  $11, $06, $10
add  $12, $06, $11
add  $13, $06, $12
add  $14, $06, $13
add  $15, $06, $14
lui  $07, 0x1001
ori  $07, 0x03F0       # LI   R07,  0x100103F0
sw   $15, 0($07)       # ST  [R07], R15
break
@200

//***********************************************
// In this ISR, we will implement writing
// some patterns to the IO space, and then reading them back.
//Note: this "ISR" expects the "return address"
// to have been read from the stack
//***********************************************
Isr:
push   $31, ($sp)          #push return address PC to stack
push   $8,  ($sp)          #push the flag to the stack
lui    $16, 0x1001         #load destination IO address
ori    $16, 0x00C0         #  0x100100C0 into r16
lui    $17, 0x8000         #initialize the pattern of
ori    $17, 0xFFFF         #  0x8000FFFF into r17
addi   $18, $0, 0x10       #loop counter set to 16
out_IO:
output $17, 0($16)         # output  [R16], R17
sra    $17, $17, 2         # change the pattern by shifting twice
addi   $16, $16, 4         # increment the memory pointer 4 bytes
addi   $18, $18, -1        # decrement the loop counter
bne    $18, $00, out_IO    # and jmp to top if not finished
lui    $16, 0x1001         # load source IO address
ori    $16, 0x00C0         # 00100C0 into r16
input  $19,  0($16)        # and input from 6
input  $20,  4($16)        # the IO locations,
input  $21,  8($16)        # starting from 0xC0
input  $22, 12($16)
input  $23, 16($16)
input  $24, 20($16)
pop    $8,  ($sp)          #pop FLAGS from the stack, write to $8
RETFL  $8                  #load FLAGS from $r8
pop    $8,  ($sp)          #pop  PC from the stack, write to $8
RETPC  $8                  #load PC from $r8
```

After returning from interrupt $07 is changed from ffffffff00 to 100103f0

When interrupt occure, flag is push to stack by push instruction. The reture Pc is 14

At the end of isr, Flags and PC is pop of stack so $sp=3FC. Initial value

Isim result:

```
BREAK INSTRUCTION FETCHED 5205.0 ps
REGISTER' AFTER BREAK
time=5221.0 ps, S_addr[00]=00000000 || T_addr[10]=100100c0
time=5231.0 ps, S_addr[01]=12345678 || T_addr[11]=ffffffff
time=5241.0 ps, S_addr[02]=87654321 || T_addr[12]=00000000
time=5251.0 ps, S_addr[03]=abcdef01 || T_addr[13]=8000ffff
time=5261.0 ps, S_addr[04]=01fedcba || T_addr[14]=e0003fff
```

```
time=5271.0 ps, S_addr[05]=5a5a5a5a || T_addr[15]=f8000fff
time=5281.0 ps, S_addr[06]=ffffffff || T_addr[16]=fe0003ff
time=5291.0 ps, S_addr[07]=100103f0 || T_addr[17]=ff8000ff
time=5301.0 ps, S_addr[08]=ffffffeff || T_addr[18]=ffe0003f
time=5311.0 ps, S_addr[09]=ffffffefe || T_addr[19]=xxxxxxxx
time=5321.0 ps, S_addr[0a]=ffffffefd || T_addr[1a]=xxxxxxxx
time=5331.0 ps, S_addr[0b]=ffffffefc || T_addr[1b]=xxxxxxxx
time=5341.0 ps, S_addr[0c]=ffffffefb || T_addr[1c]=xxxxxxxx
time=5351.0 ps, S_addr[0d]=ffffffefa || T_addr[1d]=000003fc
time=5361.0 ps, S_addr[0e]=ffffffef9 || T_addr[1e]=xxxxxxxx
time=5371.0 ps, S_addr[0f]=ffffffef8 || T_addr[1f]=00000014
```

$sp(29) = 3FC

```
DATA MEMORY FILE
time=5381.0 ps, dm_addr[00000000]=00c3c3c3
time=5391.0 ps, dm_addr[00000004]=12345678
time=5401.0 ps, dm_addr[00000008]=89abcdef
time=5411.0 ps, dm_addr[0000000c]=a5a5a5a5
time=5421.0 ps, dm_addr[00000010]=5a5a5a5a
time=5431.0 ps, dm_addr[00000014]=2468ace0
time=5441.0 ps, dm_addr[00000018]=13579bdf
time=5451.0 ps, dm_addr[0000001c]=0f0f0f0f
time=5461.0 ps, dm_addr[00000020]=f0f0f0f0
time=5471.0 ps, dm_addr[00000024]=00000009
time=5481.0 ps, dm_addr[00000028]=0000000a
time=5491.0 ps, dm_addr[0000002c]=0000000b
time=5501.0 ps, dm_addr[00000030]=0000000c
time=5511.0 ps, dm_addr[00000034]=0000000d
time=5521.0 ps, dm_addr[00000038]=fffffff8
time=5531.0 ps, dm_addr[0000003c]=000075cc
time=5541.0 ps, dm_addr[00000040]=xxxxxxxx
time=5551.0 ps, dm_addr[00000044]=xxxxxxxx

DISPLAY MEMORY AT LOCATION [3F0:3F3] =
time=6021.0 ps M[3F0]=fffffef8
PC(return addrss) M[3F8:3FB]=00000014
FLAGS address M[3F4:3F7]=00000010

DISPLAY I/O module from 0C0 to 104:
time=6031.0 ps, io_addr[000000b0]=xxxxxxxx
time=6041.0 ps, io_addr[000000b4]=xxxxxxxx
time=6051.0 ps, io_addr[000000b8]=xxxxxxxx
time=6061.0 ps, io_addr[000000bc]=xxxxxxxx
time=6071.0 ps, io_addr[000000c0]=8000ffff
time=6081.0 ps, io_addr[000000c4]=e0003fff
time=6091.0 ps, io_addr[000000c8]=f8000fff
time=6101.0 ps, io_addr[000000cc]=fe0003ff
time=6111.0 ps, io_addr[000000d0]=ff8000ff
time=6121.0 ps, io_addr[000000d4]=ffe0003f
time=6131.0 ps, io_addr[000000d8]=fff8000f
time=6141.0 ps, io_addr[000000dc]=fffe0003
time=6151.0 ps, io_addr[000000e0]=ffff8000
time=6161.0 ps, io_addr[000000e4]=fffffe000
time=6171.0 ps, io_addr[000000e8]=fffff800
time=6181.0 ps, io_addr[000000ec]=ffffffe00
time=6191.0 ps, io_addr[000000f0]=ffffff80
time=6201.0 ps, io_addr[000000f4]=fffffffe0
time=6211.0 ps, io_addr[000000f8]=fffffff8
time=6221.0 ps, io_addr[000000fc]=ffffffffe
time=6231.0 ps, io_addr[00000100]=xxxxxxxx
time=6241.0 ps, io_addr[00000104]=xxxxxxxx
```

CPU respond to external interrupt module:

```
@0
main:
setie
lui  $16, 0x1001        # change pointer to write to dMem
ori  $16, 0x00C0        #  0x100100C0 into $r16
lui  $15, 0x1001
ori  $15, 0x0000        # $r15 <-- 0x10010000  (source pointer)
lw   $01, 00($15)       #$1 = 19 load from mem at 0 location
lw   $02, 04($15)       #$2 = 03e8
lw   $03, 08($15)
lw   $04, 12($15)
lw   $05, 16($15)
lw   $06, 20($15)
lw   $07, 24($15)
lw   $08, 28($15)
lw   $09, 32($15)
lw   $10, 36($15)
lw   $11  40($15)
lw   $12, 44($15)
lw   $13, 48($15)
lw   $14, 52($15)
lw   $15, 56($15)
break

@200
//*************************************************
// In this ISR, we will implement reading input
// to write to $r17 to $r28
// Note: this "ISR" expects the "return address"
//       to have been saved on the stack and read
//       back from the stack
//*************************************************
isr:
push   $31, ($sp)         #push return address PC to stack
push   $8,  ($sp)       #push the flag to the stack
lui    $17, 0x8000      #initialize the pattern of
ori    $17, 0xFFFF      #  0x8000FFFF into r17
addi   $18, $0, 0x1E    #loop counter set to 30

out_IO:
output $17, 0($16)      # output  [R16], R17
sra    $17, $17, 2      # change the pattern by shifting twice
addi   $16, $16, 4      # increment the memory pointer 4 bytes
addi   $18, $18, -1     # decrement the loop counter
bne    $18, $00, out_IO #  and jmp to top if not finished

input  $17,  0($16)     #  and input from 0
input  $18,  4($16)     #  and input from 4
input  $19,  8($16)     #  and input from 8
input  $20,  c($16)     #
input  $21, 10($16)
input  $22, 14($16)
input  $23, 18($16)
input  $24, 1c($16)
input  $25, 20($16)     #    the IO locations,
input  $26, 24($16)     #    starting from 0xC0
input  $27, 28($16)
input  $28, 2c($16)
reti                    #       return from interrupt
```

When interrupt occure push PC and flag to stack.

Writing to IO sra of $17, the first value is e0003fff. Write from IO to $17 to $28.

Isim result:

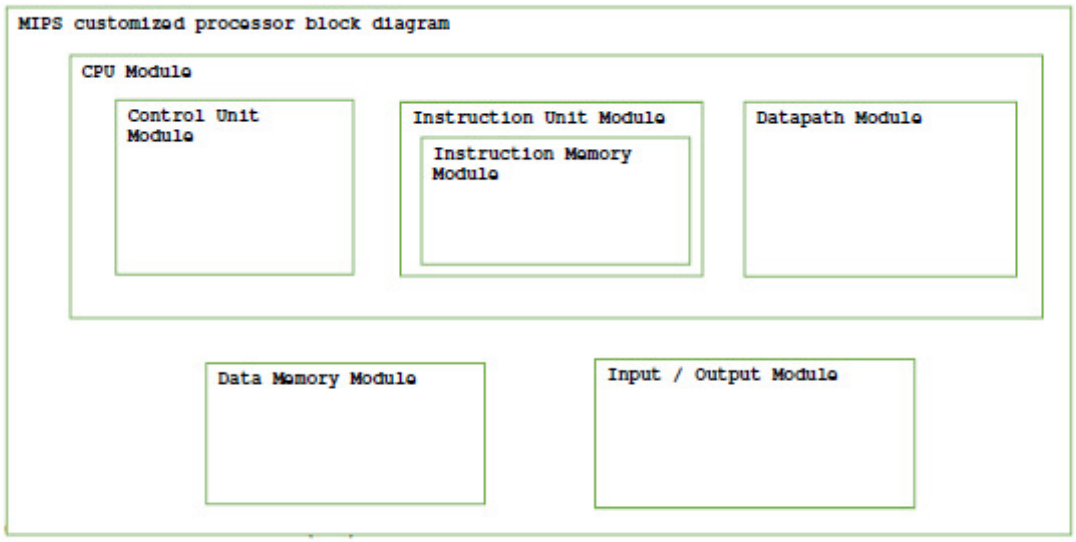
```
BREAK INSTRUCTION FETCHED 8055.0 ps
REGISTER' AFTER BREAK
time=8151.0 ps, S_addr[00]=00000000 || T_addr[10]=100100c0
time=8161.0 ps, S_addr[01]=00000019 || T_addr[11]=8000ffff
time=8171.0 ps, S_addr[02]=000003e8 || T_addr[12]=e0003fff
time=8181.0 ps, S_addr[03]=fffffffe7 || T_addr[13]=f8000fff
time=8191.0 ps, S_addr[04]=fffffc18 || T_addr[14]=fe0003ff
time=8201.0 ps, S_addr[05]=000061a8 || T_addr[15]=ff8000ff
time=8211.0 ps, S_addr[06]=ffff9e58 || T_addr[16]=ffe0003f
time=8221.0 ps, S_addr[07]=ffffffff || T_addr[17]=fff8000f
time=8231.0 ps, S_addr[08]=00000007 || T_addr[18]=fffe0003
time=8241.0 ps, S_addr[09]=00000008 || T_addr[19]=ffff8000
```

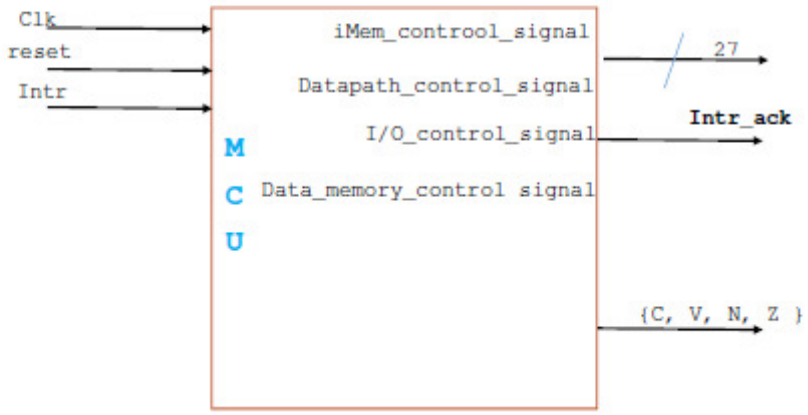
$r0 to $r15 are load from dMem_8. And the first value to read from IO is to $17 which is 8000fff. $17 not change because it is writted to IO before it is sra

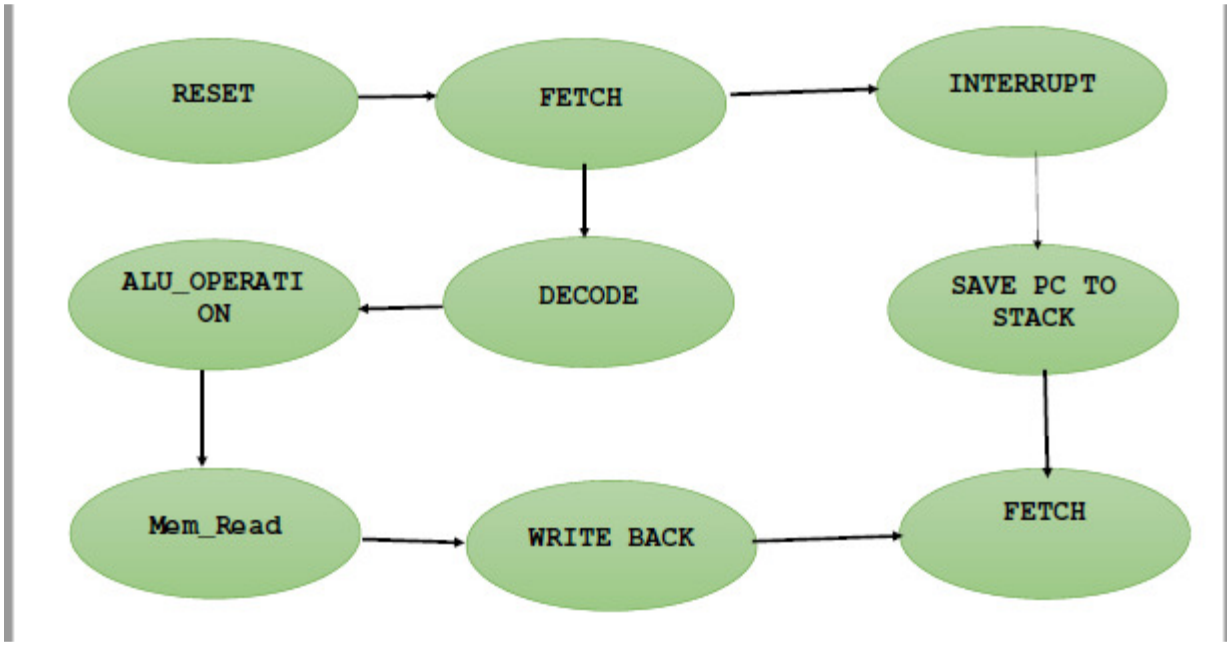
```
time=8251.0 ps, S_addr[0a]=00000009 || T_addr[1a]=ffffe000
time=8261.0 ps, S_addr[0b]=0000000a || T_addr[1b]=fffff800
time=8271.0 ps, S_addr[0c]=0000000b || T_addr[1c]=fffffe00
time=8281.0 ps, S_addr[0d]=0000000c || T_addr[1d]=000003fc
time=8291.0 ps, S_addr[0e]=0000000d || T_addr[1e]=xxxxxxxx
time=8301.0 ps, S_addr[0f]=0000000e || T_addr[1f]=00000014

DATA MEMORY FILE
time=8231.0 ps, dm_addr[00000000]=00000019
time=8241.0 ps, dm_addr[00000004]=000003e8
time=8251.0 ps, dm_addr[00000008]=ffffffe7
time=8261.0 ps, dm_addr[0000000c]=fffffc18
time=8271.0 ps, dm_addr[00000010]=000061a8
time=8281.0 ps, dm_addr[00000014]=ffff9e58
time=8291.0 ps, dm_addr[00000018]=ffffffff
time=8301.0 ps, dm_addr[0000001c]=00000007
time=8311.0 ps, dm_addr[00000020]=00000008
time=8321.0 ps, dm_addr[00000024]=00000009
time=8331.0 ps, dm_addr[00000028]=0000000a
time=8341.0 ps, dm_addr[0000002c]=0000000b
time=8351.0 ps, dm_addr[00000030]=0000000c
time=8361.0 ps, dm_addr[00000034]=0000000d
time=8371.0 ps, dm_addr[00000038]=0000000e
time=8381.0 ps, dm_addr[0000003c]=0000000f
time=8391.0 ps, dm_addr[00000040]=xxxxxxxx
time=8401.0 ps, dm_addr[00000044]=xxxxxxxx
time=8411.0 ps, dm_addr[00000048]=xxxxxxxx
time=8421.0 ps, dm_addr[0000004c]=xxxxxxxx

DISPLAY MEMORY AT LOCATION [3F0:3F3] =
time=8871.0 ps M[3F0]=xxxxxxxx
PC(return addrss) M[3F8:3FB]=00000014
FLAGS address M[3F4:3F7]=00000010

DISPLAY I/O module from 0C0 to 104:
time=8881.0 ps, io_addr[000000b0]=xxxxxxxx
time=8891.0 ps, io_addr[000000b4]=xxxxxxxx
time=8901.0 ps, io_addr[000000b8]=xxxxxxxx
time=8911.0 ps, io_addr[000000bc]=xxxxxxxx
time=8921.0 ps, io_addr[000000c0]=8000ffff
time=8931.0 ps, io_addr[000000c4]=e0003fff
time=8941.0 ps, io_addr[000000c8]=f8000fff
time=8951.0 ps, io_addr[000000cc]=fe0003ff
time=8961.0 ps, io_addr[000000d0]=ff8000ff
time=8971.0 ps, io_addr[000000d4]=ffe0003f
time=8981.0 ps, io_addr[000000d8]=fff8000f
time=8991.0 ps, io_addr[000000dc]=fffe0003
time=9001.0 ps, io_addr[000000e0]=ffff8000
time=9011.0 ps, io_addr[000000e4]=ffffe000
time=9021.0 ps, io_addr[000000e8]=fffff800
time=9031.0 ps, io_addr[000000ec]=fffffe00
time=9041.0 ps, io_addr[000000f0]=ffffff80
time=9051.0 ps, io_addr[000000f4]=fffffe0
time=9061.0 ps, io_addr[000000f8]=fffffff8
time=9071.0 ps, io_addr[000000fc]=ffffffffe
time=9081.0 ps, io_addr[00000100]=ffffffff
time=9091.0 ps, io_addr[00000104]=ffffffff
time=9101.0 ps, io_addr[00000108]=ffffffff
time=9111.0 ps, io_addr[0000010c]=ffffffff
time=9121.0 ps, io_addr[00000110]=ffffffff
time=9131.0 ps, io_addr[00000114]=ffffffff
time=9141.0 ps, io_addr[00000118]=ffffffff
time=9151.0 ps, io_addr[0000011c]=ffffffff
time=9161.0 ps, io_addr[00000120]=ffffffff
time=9171.0 ps, io_addr[00000124]=ffffffff
time=9181.0 ps, io_addr[00000128]=ffffffff
time=9191.0 ps, io_addr[0000012c]=ffffffff
time=9201.0 ps, io_addr[00000130]=ffffffff
time=9211.0 ps, io_addr[00000134]=ffffffff
time=9221.0 ps, io_addr[00000138]=xxxxxxxx
```

$r28 is read from location 2c($16)==ec, with the conten of fffffe00. $sp==3FC after the stack pointer has been restore

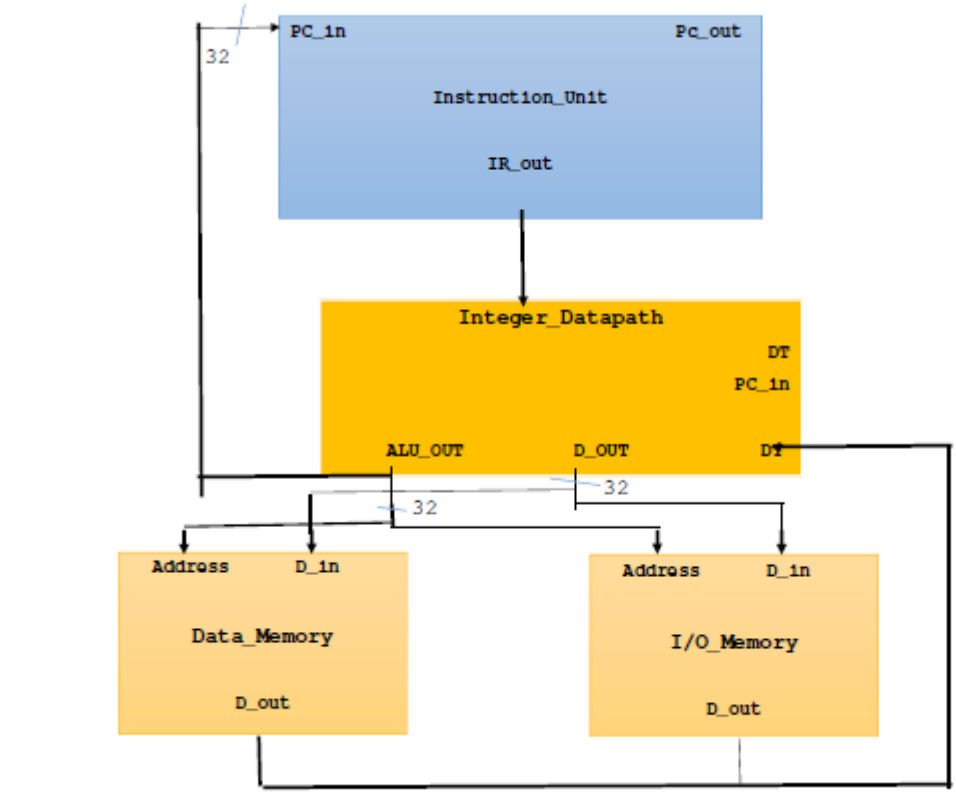
IV. Hadware Implementation:



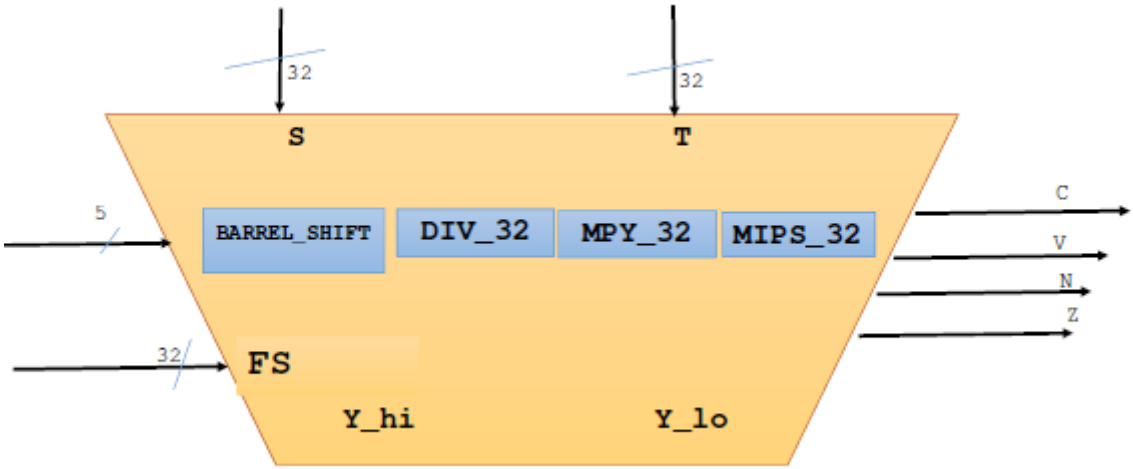




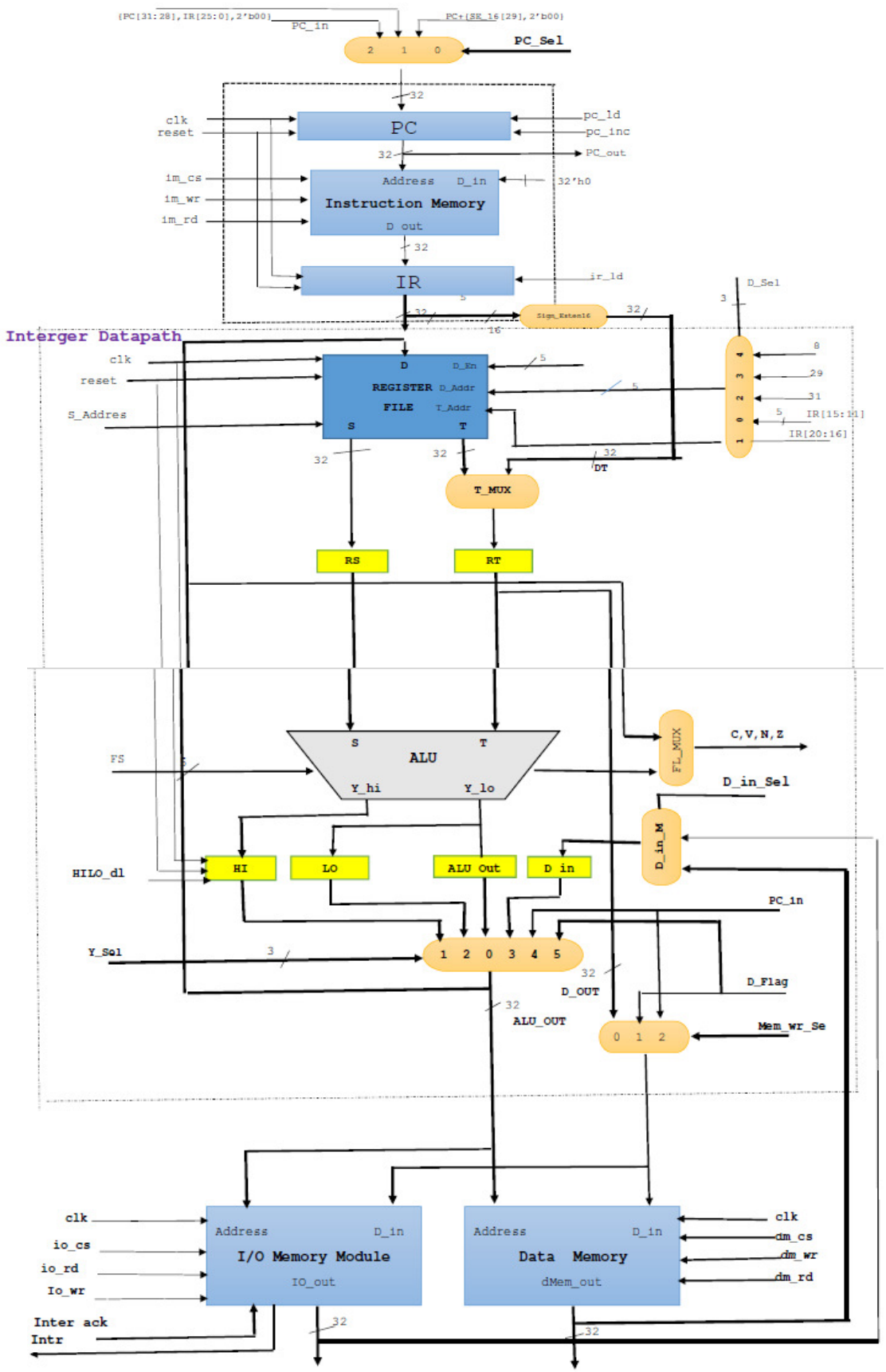
General Block of Customized MIPS CPU:



Detail Block of Internal ALU Module:



Detail Diagram:



V. Discussion and Comments:

MIPS architecture is simple and give way to learn how to design a processor. The fix instruction set make it easy to implement more instruction. The two separate memories of instruction and data memory reduces the complication reading instruction and data from data memory. Pipelining can be easily implemented with two memory architecture.
The add on enhanced instructions are pop, push, input, output and reti make a way to communicate with the outside peripheral beside the data memeory.