RISC Verilog based 8-Bit Processor

Implementation on FPGA

Meet Doshi

1812078 KJSCE-22

Introduction:

This processor is based on RISC based ISA architecture which implements early x86 instruction set on RISC based design.

The processor’s memory is based on Harvard architecture which implements separate memory for instructions and data.

The processor is non-pipelined, meaning it can only execute one instruction at a time. Every instruction takes 2 clock cycles or t-states. As the processor is non pipelined the branch penalty is null, furthermore no WAR & RAW data dependencies exist. Not implementing pipelining takes a major toll on clock cycles in hardware this processor might even take up to 4 t-states.

Processor Features:

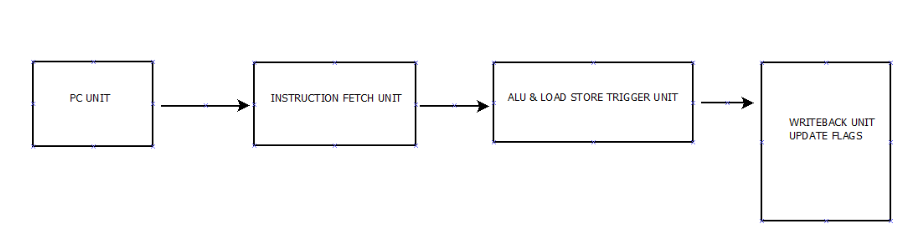
* Harvard Architecture

256 X 24 Bit Instruction Memory

16K X 8 Bit Data Memory

* 8 Bit Data Bus
* 24 Bit Instruction Bus
* 32 General Purpose Registers
* Clock Freq =
* 2.5V Voltage Supply
* Dual Edge Triggered For Lower CPI
* 2.0 Average CPI
* 4 Stage Instruction Life Cycle
* 18 Control Signals
* 4 Bit Flag Register PZBC

Instruction Life Cycle



1. PC Unit

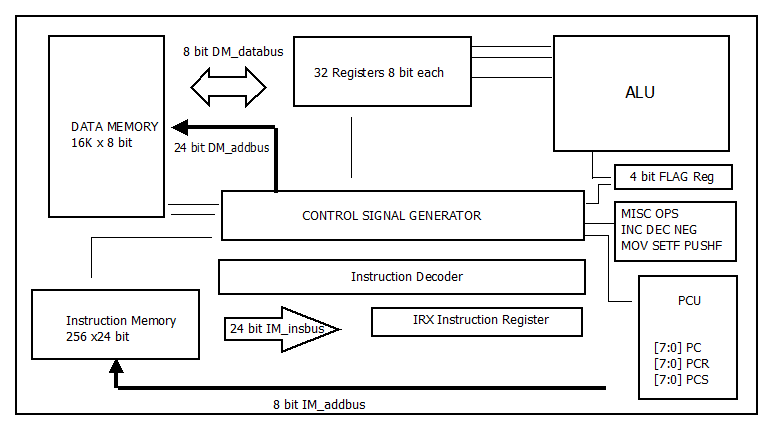
* Check reset and Enable
* Load PC with instruction address based on control signals

1. Instruction Fetch Unit
   * Load PC in to instruction register IRX
   * Fetch Instruction from Instruction memory
   * Instantiate control signals
2. ALU Load Store Unit

* Instantiate ALU if needed and release control signals
* Instantiate DATA MEMORY if needed
* Execute Non ALU operations and store them into temp registers for next stage

1. Write back and update flags

* Write back results and update flags
* Increment PCR

Processor Architecture

INTERNAL BUSES:

* 24 Bit Instruction Bus
* 8 Bit Instruction Address Bus
* 14 Bit Data Address Bus
* 8 Bit Data Bus
* 4 Bit Flag Bus
* 3 Operand Bus 8 Bit each
* 8 Bit Result Bus

CONTROL SIGNALS:

1. INSGRP
2. INSOPC
3. RDIM
4. RDDM
5. WRDM
6. OPERANDS1
7. OPERANDS2
8. OPERANDS3
9. ALU
10. RDLOAD
11. RDSTORE
12. ASSIGN
13. MOV
14. BRANCH
15. SPC
16. RSPC
17. SWRESET
18. STOP

REGISTER SET:

BASED ON MIPS REGISTER SET TYPE

32 Available to USERS

21 Registers For GPR's

11 Saved For Assembler

* RegA …. RegO (15 Registers)
* RegZero
* RegPCS1
* RegPCS2
* RegPCS3
* Regsad1
* Regsad2

7 Temporary Registers

* 2 For Decoder
* 5 For ALU

INSTRUCTION REGISTER:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| INSTRUCTION REGISTER IRX | | | | | | |
| GROUP  2 BIT | OPCODE  3 BIT | REG  5 BIT | | ADDRESS FOR DATA TRANSFER INSTRUCTIONS OR BRANCH INSTRUCTIONS  14 BITS OR 8 BITS RESP. | | |
| GG | OPC | ##### | | @@@@@@@@@@@@@@ | | |
| GG | OPC | ##### | | REG 5Bits | SEL 1 Bit | REG/IV 8 Bits |
| ------------------------------------------ | | | | ##### | 0/1 | XXX##### OR 8BIT IV |
| GG | OPC | | xxxxx | xxxxx | x | xxxxFFFF |

SEL:

0=Register

1=Immediate Value (IV)

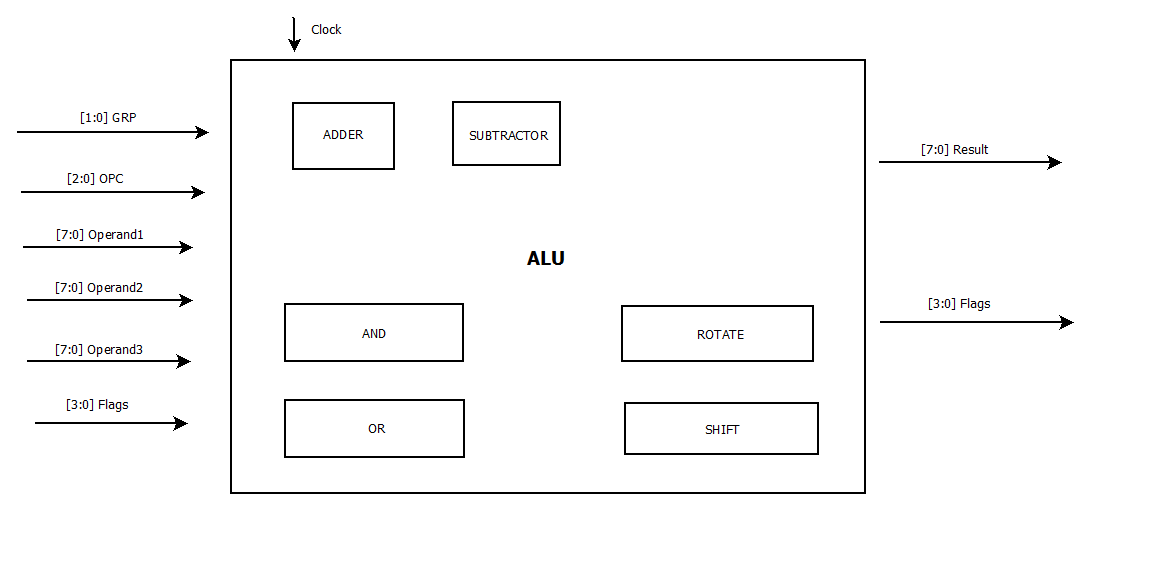
#: Register Select

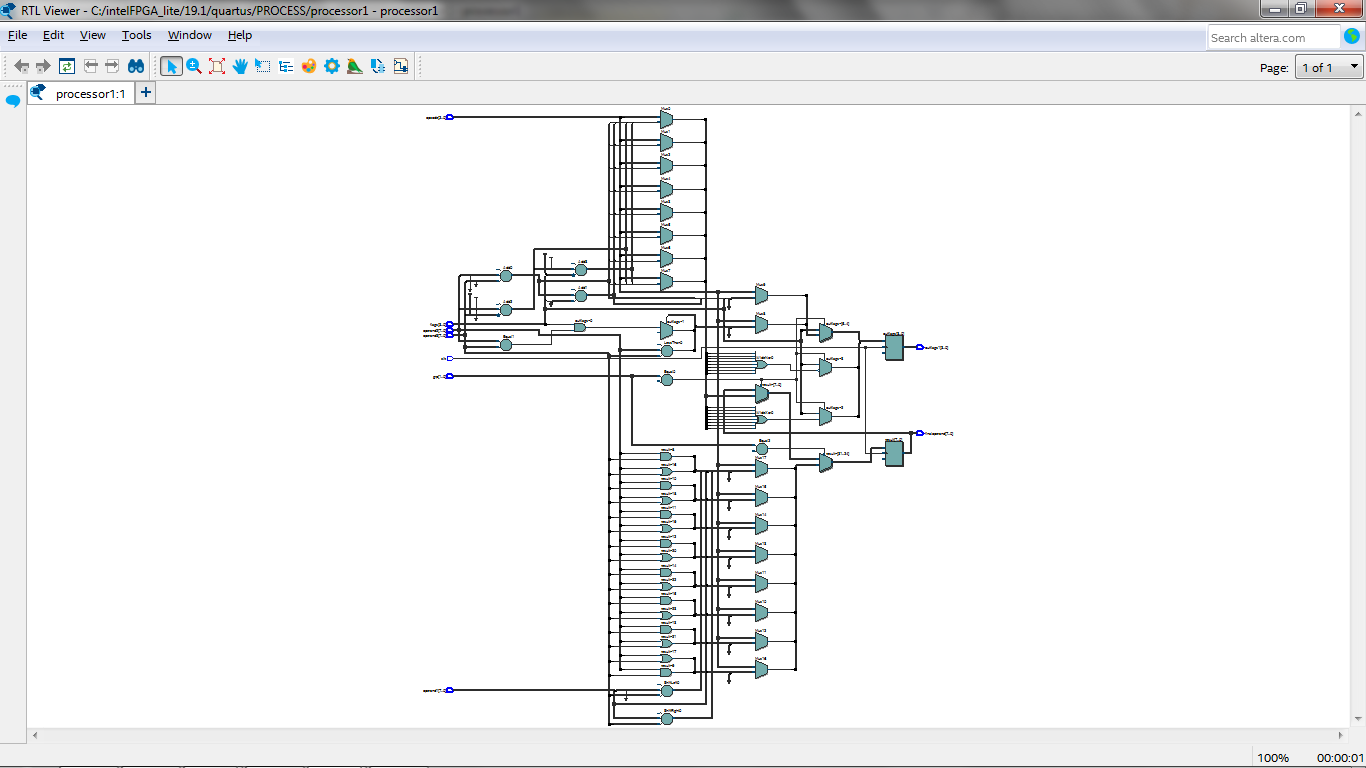
@: Address Bits

F: Flag Bits

X: Don’t Care

**ALU:**





**ALU FEATURES:**

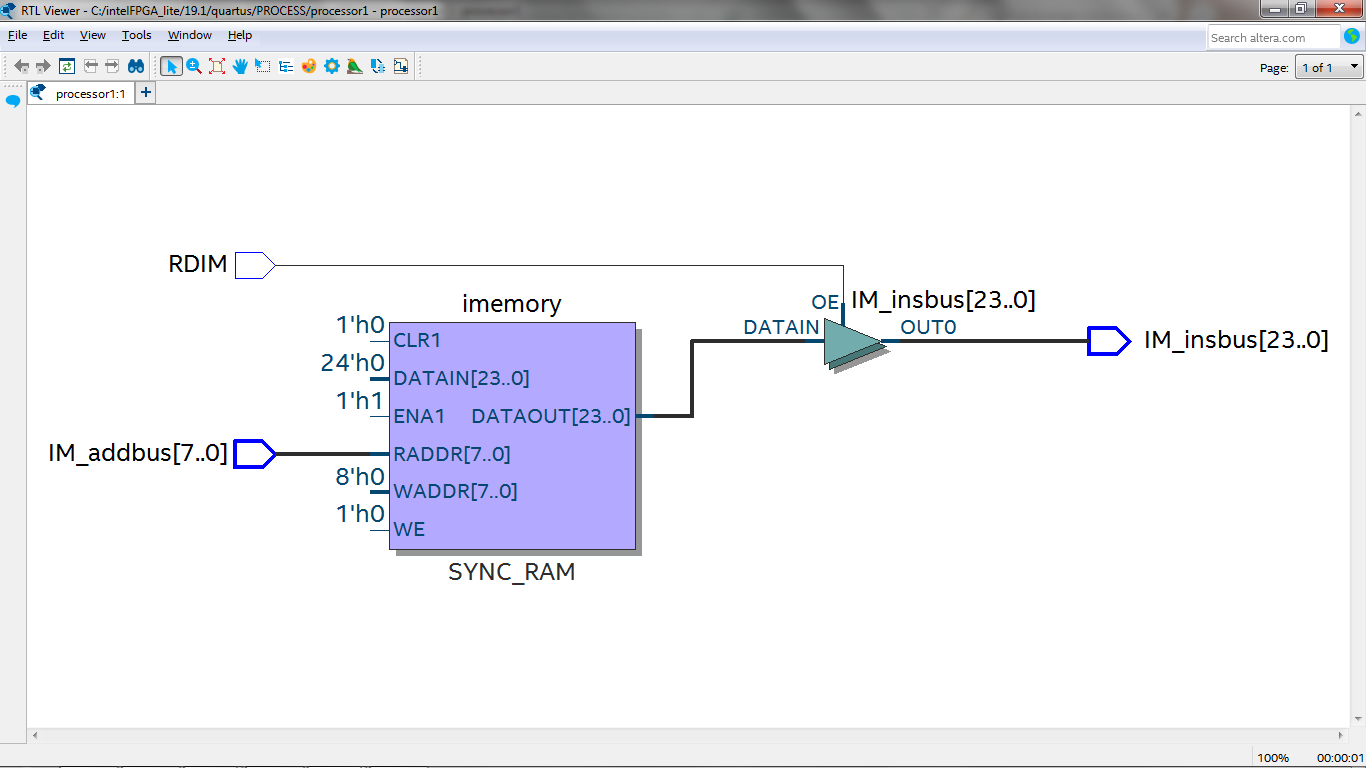
* 8 Bit Adder Subtractor with carry and borrow
* 8 Bit AND/OR
* ROTATE SHIFT (LEFT/ RIGHT)
* No Barrel Shifter
* Update Flags only on Adder Subtractor

**FLAG REGISTER**

|  |  |  |  |
| --- | --- | --- | --- |
| **P** | **Z** | **B** | **C** |

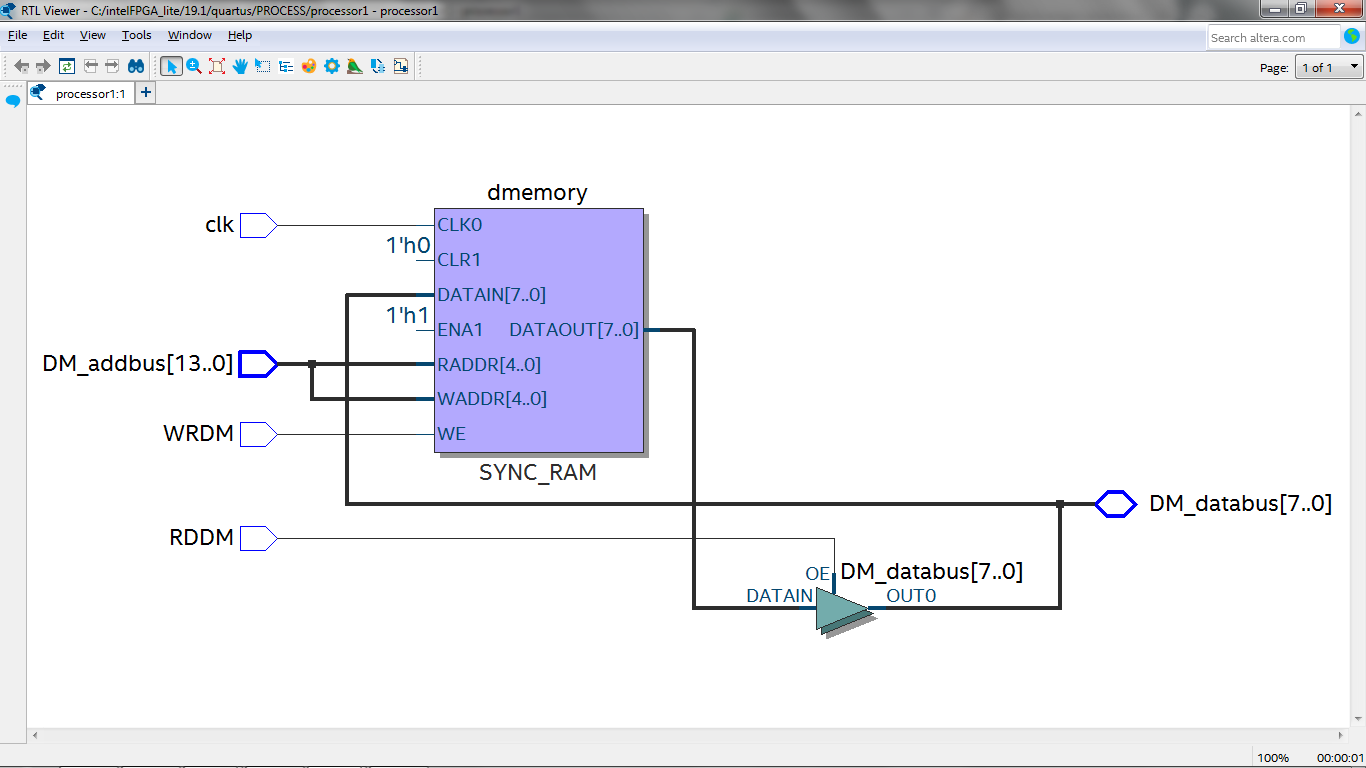
* Bit 3 Parity (0: Odd/1:Even)
* Bit 2 Zero (0:Not Zero/1:Zero)
* Bit 1 Borrow (1:Borrow)
* Bit 0 Carry (1:Carry)

**INSTRUCTION MEMORY ASYNCHRONOUS**

****

* 8 Bit IM ADDRESS BUS
* 24 Bit IM INSTRUCTION BUS
* RDIM CONTROL SIGNAL

**DATA MEMORY SYNCHRONOUS WRITE**



* 14 Bit ADDRESS BUS
* 8 Bit DATA BUS
* ASYNCHRONOUS READ
* SYNCHRONOUS WRITE

**INSTRUCTION SET 24 bit**

**Total 29 Instructions RISC Based**

|  |  |  |
| --- | --- | --- |
| **Opcode** | **Mnemonic** | **Instruction** |
| **00000** | **ADD** | **Add 2 reg and store into 3rd reg** |
| **00001** | **ADC** | **ADD with added carry** |
| **00010** | **SUB** | **Subtract 2 reg and store in 3rd reg** |
| **00011** | **SBB** | **SUB with subtracted borrow** |
| **00100** | **INC** | **Increment Reg** |
| **00101** | **DEC** | **Decrement Reg** |
| **00110** | **NEG** | **Complement Reg** |
| **01000** | **AND** | **Logical And 2 reg and store in 3rd reg** |
| **01001** | **OR** | **Logical Or 2 reg and store in 3rd reg** |
| **01010** | **ROR** | **Rotate right reg upto 8 times** |
| **01011** | **ROL** | **Rotate left reg upto 8 times** |
| **01100** | **SHR** | **Shift right reg upto 8 times** |
| **01101** | **SHL** | **Shift left reg upto 8 times** |
| **01110** | **SETF** | **Set the flag reg** |
| **01111** | **PUSHF** | **Push carry flag into reg** |
| **10000** | **JMP** | **Unconditional jump** |

|  |  |  |
| --- | --- | --- |
| **Opcode** | **Mnemonic** | **Instruction** |
| **10001** | **JC** | **Jump if carry** |
| **10010** | **JNC** | **Jump if no carry** |
| **10011** | **JZ** | **Jump if zero** |
| **10100** | **JNZ** | **Jump if not zero** |
| **10101** | **JPE** | **Jump if parity even** |
| **10110** | **JPO** | **Jump if parity odd** |
| **11000** | **RES** | **Reset** |
| **11001** | **NOP** | **No-Operation** |
| **11010** | **SPC** | **Store Program Counter** |
| **11011** | **RSPC** | **Restore Program Counter** |
| **11100** | **LOAD** | **Load memory into reg** |
| **11101** | **STOR** | **Store reg to memory** |
| **11110** | **MOV** | **Move 1 reg to another** |

* x8086 Based Instruction Set

**Arithmetic Group 00**

**7 instructions**

Sel:

0=Reg

1=Immediate value(IV)

#: Register select

@: Address bits

1. Add R1🡨 R2 + (R3/IV) Update Flags **Mnemonic: ADD**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Group | Opcode | Reg | Reg | Sel | Reg/IV |
| 00 | 000 | ##### | ##### | 0/1 | xxx##### or 8bit IV |

1. Add with Carry R1🡨 R2 + (R3/IV) + C Update Flags **Mnemonic: ADC**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Group | Opcode | Reg | Reg | Sel | Reg/IV |
| 00 | 001 | ##### | ##### | 0/1 | xxx##### or 8bit IV |

1. Sub R1🡨 R2 - (R3/IV) Update Flags **Mnemonic: SUB**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Group | Opcode | Reg | Reg | Sel | Reg/IV |
| 00 | 010 | ##### | ##### | 0/1 | xxx##### or 8bit IV |

1. Sub with borrow R1🡨 R2 – (R3/IV) -B Update Flags **Mnemonic: SBB**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Group | Opcode | Reg | Reg | Sel | Reg/IV |
| 00 | 011 | ##### | ##### | 0/1 | xxx##### or 8bit IV |

1. INC Reg **Mnemonic: INC**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Group | Opcode | Reg | Reg | Sel | Reg/IV |
| 00 | 100 | ##### | xxxxx | x | xxxx xxxx |

1. DEC Reg **Mnemonic: DEC**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Group | Opcode | Reg | Reg | Sel | Reg/IV |
| 00 | 101 | ##### | xxxxx | x | xxxx xxxx |

1. COMPLEMENT Reg **Mnemonic: NEG**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Group | Opcode | Reg | Reg | Sel | Reg/IV |
| 00 | 110 | ##### | xxxxx | x | xxxx xxxx |

**Logical & Misc Group 01**

**8 instructions**

Sel:

0=Reg

1=Immediate value(IV)

#: Register select

@: Address bits

F: Flag Bits

1. AND R1🡨 R2 & (R3/IV)  **Mnemonic: AND**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Group | Opcode | Reg | Reg | Sel | Reg/IV |
| 01 | 000 | ##### | ##### | 0/1 | xxx##### or 8bit IV |

1. OR R1🡨 R2 | (R3/IV)  **Mnemonic: OR**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Group | Opcode | Reg | Reg | Sel | Reg/IV |
| 01 | 001 | ##### | ##### | 0/1 | xxx##### or 8bit IV |

1. ROTATE RIGHT Reg>>bit val **Mnemonic: ROR**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Group | Opcode | Reg | Reg | Sel | Reg/IV |
| 01 | 010 | ##### | xxxxx | x | xxxxxxxx |

1. ROTATE LEFT Reg<<bit val **Mnemonic: ROL**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Group | Opcode | Reg | Reg | Sel | Reg/IV |
| 01 | 011 | ##### | xxxxx | x | xxxxxxxx |

1. SHIFT RIGHT Reg>>bit **Mnemonic: SHR**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Group | Opcode | Reg | Reg | Sel | Reg/IV |
| 01 | 100 | ##### | xxxxx | x | xxxxxxxx |

1. SHIFT LEFT Reg<<bit **Mnemonic: SHL**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Group | Opcode | Reg | Reg | Sel | Reg/IV |
| 01 | 101 | ##### | xxxxx | x | xxxxxxxx |

1. SET FLAGS **Mnemonic: SETF**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Group | Opcode | Reg | Reg | Sel | Reg/IV |
| 01 | 110 | xxxxx | xxxxx | x | xxxxFFFF(4 flag bits) |

1. PUSH FLAG **Mnemonic: PUSHF**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Group | Opcode | Reg | Reg | Sel | Reg/IV |
| 01 | 111 | ##### | xxxxx | x | xxxxxxxx |

**Branch Group 10**

**7 instructions**

#: Register select

@: Address bits INS MEMORY

1. JUMP Instruction Memory Address **Mnemonic: JMP**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Group | Opcode | Reg | Reg | Sel | IV |
| 10 | 000 | xxxxx | xxxxx | x | @@@@@@@@ |

1. Jump If Carry **Mnemonic: JC**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Group | Opcode | Reg | Reg | Sel | IV |
| 10 | 001 | xxxxx | xxxxx | x | @@@@@@@@ |

1. Jump If No Carry **Mnemonic: JNC**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Group | Opcode | Reg | Reg | Sel | IV |
| 10 | 010 | xxxxx | xxxxx | x | @@@@@@@@ |

1. Jump If Zero **Mnemonic: JZ**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Group | Opcode | Reg | Reg | Sel | IV |
| 10 | 011 | xxxxx | xxxxx | x | @@@@@@@@ |

1. Jump If Non Zero **Mnemonic: JNZ**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Group | Opcode | Reg | Reg | Sel | IV |
| 10 | 100 | xxxxx | xxxxx | x | @@@@@@@@ |

1. Jump If Even Parity **Mnemonic: JPE**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Group | Opcode | Reg | Reg | Sel | IV |
| 10 | 101 | xxxxx | xxxxx | x | @@@@@@@@ |

1. Jump If Odd Parity **Mnemonic: JPO**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Group | Opcode | Reg | Reg | Sel | IV |
| 10 | 110 | xxxxx | xxxxx | x | @@@@@@@@ |

**Machine Control / Load Store Group 11**

**7 instructions**

#: Register select

@: Address bits DATA MEMORY

1. RESET **Mnemonic: RES**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Group | Opcode | Reg | Reg | Sel | Reg/IV |
| 11 | 000 | xxxxx | xxxxx | x | xxxx xxxx |

1. NOP **Mnemonic: NOP**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Group | Opcode | Reg | Reg | Sel | Reg/IV |
| 11 | 001 | xxxxx | xxxxx | x | xxxx xxxx |

1. Save PC **Mnemonic: SPC**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Group | Opcode | Reg | Reg | Sel | Reg/IV |
| 11 | 010 | xxxxx | xxxxx | x | xxxx xxxx |

1. Restore PC **Mnemonic: RSPC**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Group | Opcode | Reg | Reg | Sel | Reg/IV |
| 11 | 011 | xxxxx | xxxxx | x | xxxx xxxx |

1. Load Reg 🡨 Memory **Mnemonic: LOAD**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Group | Opcode | Reg | Reg | Sel | Reg/IV |
| 11 | 100 | ##### | @@@@@ | @ | @@@@@@@@ |

1. Store Reg 🡪 Memory **Mnemonic: STOR**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Group | Opcode | Reg | Reg | Sel | Reg/IV |
| 11 | 101 | ##### | @@@@@ | @ | @@@@@@@@ |

1. Move Reg 🡨 Reg **Mnemonic: MOV**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Group | Opcode | Reg | Reg | Sel | Reg/IV |
| 11 | 110 | ##### | ##### | x | xxxxxxxx |

**FLAG REGISTER**

|  |  |  |  |
| --- | --- | --- | --- |
| **Parity(0=EP/1=OP)** | **Zero** | **Borrow** | **Carry** |
| **F** | **F** | **F** | **F** |

**SOURCE CODE**

**Processor:**

//TWO INPUT CLK 25% duty cycle

/\*

RegA=5'b00000,

RegB=5'b00001,

RegC=5'b00010,

RegD=5'b00011,

RegE=5'b00100,

RegF=5'b00101,

RegG=5'b00110,

RegH=5'b00111,

RegI=5'b01000,

RegJ=5'b01001,

RegK=5'b01010,

RegL=5'b01011,

RegM=5'b01100,

RegN=5'b01101,

RegO=5'b01110,

RegZero=5'b01111,

RegPCS1=5'b10000,

RegPCS2=5'b10001,

RegPCS3=5'b10010,

Regsad1=5'b10011,

Regsad2=5'b10100;

\*/

//21 registers for GPR's

//11 saved for assembler

module processor1(

input ENABLE,

input IMclk,

input ALUDMclk,

input reset,

output [1:0]insgrp,

output [2:0]insopc,

output RDIM,

output RDDM,

output WRDM,

output reg [23:0]IRX,

output reg [7:0]result1,

output reg [3:0]flagreg,

output [7:0]operands1,

output [7:0]operands2,

output [7:0]operands3,

output [7:0]DM\_databus,

output [23:0]IM\_insbus,

output [13:0]DM\_addbus,

output reg [7:0]PC,

output aluclk,

output dmclk,

output [3:0]flag,

output [7:0]result,

output RDLOAD,

output RDSTORE,

output ALU,

output ASSIGN,

output MOV,

output BRANCH,

output SPC,

output RSPC,

output SWRESET,

output reg [7:0]operandstemp

//output reg count,

//output reg IMclk,

//output reg ALUDMclk

);

reg [7:0]operands[0:31];

reg [3:0]flagreg1;

reg [7:0]PCR;

reg [7:0]PCS;

wire STOP;

//INS MEM INSTANTIATION

insmemory in(

.RDIM(RDIM),

.IM\_addbus(PC),

.IM\_insbus(IM\_insbus));

//DATA MEM INSTANTIATION

datamemory da(

.clk(dmclk),

.RDDM(RDDM),

.DM\_addbus(DM\_addbus),

.WRDM(WRDM),

.DM\_databus(DM\_databus));

//ALU INSTANTIATION

alu test(

.clk(aluclk),

.grp(insgrp),

.opcode(insopc),

.operand1(operands1),

.operand2(operands2),

.operand3(operands3),

.flags(flagreg),

.finaloperand(result),

.outflags1(flag));

initial

begin

PCR=8'b0;

operands[0]=8'b00000000;

operands[1]=8'b00000000;

operands[2]=8'b00000000;

operands[3]=8'b00000000;

operands[4]=8'b00000000;

operands[5]=8'b00000000;

operands[6]=8'b00000000;

operands[7]=8'b00000000;

operands[8]=8'b00000000;

operands[9]=8'b00000000;

operands[10]=8'b00000000;

operands[11]=8'b00000000;

operands[12]=8'b00000000;

operands[13]=8'b00000000;

operands[14]=8'b00000000;

operands[15]=8'b00000000;

operands[16]=8'b00000000;

operands[17]=8'b00000000;

operands[18]=8'b00000000;

operands[19]=8'b00000000;

operands[20]=8'b00000000;

operands[21]=8'b00000000;

operands[22]=8'b00000000;

operands[23]=8'b00000000;

operands[24]=8'b00000000;

operands[25]=8'b00000000;

operands[26]=8'b00000000;

operands[27]=8'b00000000;

operands[28]=8'b00000000;

operands[29]=8'b00000000;

operands[30]=8'b00000000;

operands[31]=8'b00000000;

end

////////////////////IM AND ALUDM CLK

always@(posedge IMclk )

begin

if((reset==1'b0) && (ENABLE==1'b1))

begin

PCS=(SPC==1'b1)?PC:PCS;

if(BRANCH==1'b1)

begin

PC=IRX[7:0];

end

if(RSPC==1'b1)

begin

PC=PCS;

end

else

PC=PCR;

end

end

always@(negedge IMclk)

begin

IRX=IM\_insbus;

end

always@(posedge ALUDMclk)

begin

case(IRX[23:19])

5'b00100://INC#

begin

operandstemp=((operands[IRX[18:14]])+8'b00000001);

end

5'b00101://DEC#

begin

operandstemp=(operands[IRX[18:14]]-8'b00000001);

end

5'b00110://NEG#

begin

operandstemp=(~operands[IRX[18:14]]);

end

5'b11110://MOV#

begin

operandstemp=(MOV==1'b1)?operands[IRX[13:9]]:1'b0;

end

5'b01110://SETFLAG

begin

flagreg1=IRX[3:0];

end

5'b01111://PUSH FLAG#

begin

operandstemp={4'b0,flagreg};

end

endcase

end

always@(negedge ALUDMclk )

begin

operands[IRX[18:14]]=(((IRX[23:19]==(5'b00000))||(IRX[23:19]==(5'b00001))||(IRX[23:19]==(5'b00010))||(IRX[23:19]==(5'b00011))||(IRX[23:19]==(5'b01000))||(IRX[23:19]==(5'b01001))||(IRX[23:19]==(5'b01010))||(IRX[23:19]==(5'b01011))||(IRX[23:19]==(5'b01100))||(IRX[23:19]==(5'b01101))))?result:operands[IRX[18:14]];

operands[IRX[18:14]]=((IRX[23:19]==5'b00100)||(IRX[23:19]==5'b00101)||(IRX[23:19]==5'b00110)||(IRX[23:19]==5'b11110)||(IRX[23:19]==5'b01111))?(operandstemp):(operands[IRX[18:14]]);

flagreg=(((IRX[23:19]==(5'b00000))||(IRX[23:19]==(5'b00001))||(IRX[23:19]==(5'b00010))||(IRX[23:19]==(5'b00011))||(IRX[23:19]==(5'b01000))||(IRX[23:19]==(5'b01001))||(IRX[23:19]==(5'b01010))||(IRX[23:19]==(5'b01011))||(IRX[23:19]==(5'b01100))||(IRX[23:19]==(5'b01101))))?flag:flagreg;

flagreg=(IRX[23:19]==5'b01110)?flagreg1:flagreg;

PCR=PC+1'b1;

result1=(((IRX[23:19]==(5'b00000))||(IRX[23:19]==(5'b00001))||(IRX[23:19]==(5'b00010))||(IRX[23:19]==(5'b00011))||(IRX[23:19]==(5'b01000))||(IRX[23:19]==(5'b01001))||(IRX[23:19]==(5'b01010))||(IRX[23:19]==(5'b01011))||(IRX[23:19]==(5'b01100))||(IRX[23:19]==(5'b01101))))?result:8'b0;

end

assign insgrp=IRX[23:22];

assign insopc=IRX[21:19];

assign RDIM=(reset==1'b0)?1'b1:1'b0;

assign RDDM=(RDLOAD==1'b1)?1'b1:1'b0;

assign WRDM=(RDSTORE==1'b1)?1'b1:1'b0;

assign operands1=(ALU==1'b1)?operands[IRX[18:14]]:8'bz;

assign operands2=(ALU==1'b1)?operands[IRX[13:9]]:8'bz;

assign operands3=(ALU==1'b1)?((IRX[8]==1'b0)?operands[IRX[4:0]]:IRX[7:0]):8'bz;

assign ALU=(((IRX[23:19]==(5'b00000))||(IRX[23:19]==(5'b00001))||(IRX[23:19]==(5'b00010))||(IRX[23:19]==(5'b00011))||(IRX[23:19]==(5'b01000))||(IRX[23:19]==(5'b01001))||(IRX[23:19]==(5'b01010))||(IRX[23:19]==(5'b01011))||(IRX[23:19]==(5'b01100))||(IRX[23:19]==(5'b01101)))&&(ALUDMclk==1'b1))?1'b1:1'b0;

assign RDLOAD=(IRX[23:19]==5'b11100)?1'b1:1'b0;

assign RDSTORE=(IRX[23:19]==5'b11101)?1'b1:1'b0;

assign ASSIGN=((IRX[23:19]==5'b00100)||(IRX[23:19]==5'b00101)||(IRX[23:19]==5'b00110)||(IRX[23:19]==5'b01110)||(IRX[23:19]==5'b01111))?1'b1:1'b0;

assign DM\_addbus=(((RDSTORE==1'b1)&&(WRDM==1'b1))||((RDLOAD==1'b1)&&(RDDM==1'b1)))?IRX[13:0]:14'bz;

assign MOV=(IRX[23:19]==5'b11110)?1'b1:1'b0;

assign BRANCH=(IRX[23:19]==5'b10000)?1'b1:((((IRX[23:19]==5'b10001)&&(flagreg[0]==1'b1))||((IRX[23:19]==5'b10010)&&(flagreg[0]==1'b0))||((IRX[23:19]==5'b10011)&&(flagreg[2]==1'b1))||((IRX[23:19]==5'b10100)&&(flagreg[2]==1'b0))||((IRX[23:19]==5'b10101)&&(flagreg[3]==1'b0))||((IRX[23:19]==5'b10110)&&(flagreg[3]==1'b1)))?1'b1:1'b0);

assign SPC=(IRX[23:19]==5'b11010)?1'b1:1'b0;

assign RSPC=(IRX[23:19]==5'b11011)?1'b1:1'b0;

assign SWRESET=(IRX[23:19]==5'b11000)?1'b1:1'b0;

assign DM\_databus=(RDSTORE==1'b1)?operands[IRX[18:14]]:8'bz;

assign aluclk=ALU;

assign dmclk=WRDM;

assign STOP=((reset==1'b1)||(SWRESET==1'b1))?1'b1:1'b0;

endmodule

**ALU**

module alu(

input clk,

input [1:0]grp,

input [2:0]opcode,

input [7:0]operand1,

input [7:0]operand2,

input [7:0]operand3,

input [3:0]flags,

output [7:0]finaloperand,

output [3:0]outflags1);

reg [8:0]result;

reg [8:0] subresult;

reg [8:0] temp;

reg [3:0]outflags;

reg[15:0]shifter;

assign finaloperand=result[7:0];

assign outflags1=outflags;

always@(posedge clk)

begin

outflags=flags;

if(grp==2'b00)

begin

case(opcode)

3'b000:begin//ADD

result=operand2+operand3;

outflags[0]=result[8];

outflags[1]=1'b0;

end

3'b001:begin//ADC

result=operand2+operand3+flags[0];

outflags[0]=result[8];

outflags[1]=1'b0;

end

3'b010:begin//SUB

result=operand2-operand3;

outflags[1]=(operand3>operand2)?1'b1:1'b0;

outflags[0]=1'b0;

end

3'b011:begin//SBB

result=operand2-operand3-flags[1];

if(operand3>operand2) begin

outflags[1]=(operand3>operand2)?1'b1:1'b0;

end

else

outflags[1]=((operand3==operand2)&&(flags[1]==1'b1))?1'b1:1'b0;

outflags[0]=1'b0;

end

default:result=9'bx;

endcase

outflags[2]=~|result[7:0];

outflags[3]=^result[7:0];

end

if(grp==2'b01)

begin

case(opcode)

3'b000:begin//AND

result=operand2 & operand3;

end

3'b001:begin//OR

result=operand2 | operand3;

end

3'b010:begin//ROR

subresult[8:1]=operand1;

temp=(subresult>>1);

result[6:0]=temp[7:1];

result[7]=temp[0];

end

3'b011:begin//ROL

subresult[7:0]=operand1;

temp=subresult<<1;

result[7:1]=temp[7:1];

result[0]=temp[8];

end

3'b100:begin//SHR

shifter[15:8]=(operand1[7]==1'b1)?8'b11111111:8'b00000000;

shifter[7:0]=operand1[7:0];

result[7:0]=shifter>>1;

end

3'b101:begin//SHL

result=operand1<<1;

end

default:result=9'bx;

endcase

end

end

endmodule

**DATA MEMORY**

//16k x 8 bit data mem

module datamemory(

input clk,

input RDDM,

input [13:0]DM\_addbus,

input WRDM,

inout [7:0] DM\_databus);

reg [7:0] dmemory [0:16383];

initial begin

$readmemb("C:\\intelFPGA\_lite\\19.1\\quartus\\PROCESS\\datacode.dat", dmemory);

end

assign DM\_databus=(RDDM==1'b1)?dmemory[DM\_addbus]: 8'bz;

always@(posedge clk)

if(WRDM==1'b1)

dmemory[DM\_addbus]=DM\_databus;

endmodule

**INSTRUCTION MEMORY**

//256 x 24 bit ins mem

module insmemory(

input RDIM,

input [7:0]IM\_addbus,

output [23:0] IM\_insbus);

reg [23:0]imemory[0:255];

initial begin

$readmemb("C:\\intelFPGA\_lite\\19.1\\quartus\\PROCESS\\instructioncode.dat", imemory);

end

assign IM\_insbus=(RDIM)?imemory[IM\_addbus]: 8'bz;

endmodule

**SAMPLE INSTRUCTION FILE**

//Arithmetic operations

000000000000000100000001 // ADD A=A+1 A=1

000010000000000111111111 //ADC A=A+255+Car A=0 C=1

000010000100000100000001 //ADC B=A+1+Car B=2 C=0

000100000000000100000001 //SUB A=A-1 A=255 Bor=1

000110000000000000000001 //SBB A=A-B-Bor A=252

001000000100000000000000 //INC B B=3

001010000100000000000000 //DEC B B=2

001100000100000000000000 //NEG B

//Logical MISC ops

000000000001111100001010 //ADD A=Zeroreg + 10

000000000101111100000101 //ADD B=Zeroreg + 5

010000001000000000000001 //AND C= A & B

010010001100000000000001 //OR D= A || B

010100001100000000000000 //ROR D Reg Ans 10000111

010110001100000000000000 //ROL D Reg Ans 00001111

011000001100000000000000 //SHR D Reg Ans 00000111

011010001100000000000000 //SHL D Reg Ans 00001110

011100000000000000000100 //SET FLAG 0100

011110010000000000000000 //PUSH FLAF INTO E REG

//MACHINE CONTROL

110100000000000000010000 // SPC

000000000000000100000001 // ADD A=A+1 A=1

000010000000000111111111 //ADC A=A+255+Car A=0 C=1

000010000100000100000001 //ADC B=A+1+Car B=2 C=0

000100000000000100000001 //SUB A=A-1 A=255 Bor=1

000110000000000000000001 //SBB A=A-B-Bor A=252

001000000100000000000000 //INC B B=3

001010000100000000000000 //DEC B B=2

001100000100000000000000 //NEG B

110110000000000000010000 // Restore PC with PCS

**STIMULUS FOR PROCESSOR**

module ptest;

reg ENABLE;

reg IMclk;

reg ALUDMclk;

reg reset;

wire [1:0]insgrp;

wire [2:0]insopc;

wire RDIM;

wire RDDM;

wire WRDM;

wire [23:0]IRX;

wire [7:0]result1;

wire [3:0]flagreg;

wire [7:0]operands1;

wire [7:0]operands2;

wire [7:0]operands3;

wire [7:0]DM\_databus;

wire [23:0]IM\_insbus;

wire [13:0]DM\_addbus;

wire [7:0]PC;

wire aluclk;

wire dmclk;

wire [3:0]flag;

wire RDLOAD;

wire RDSTORE;

wire ALU;

wire ASSIGN;

wire MOV;

wire BRANCH;

wire SPC;

wire RSPC;

wire SWRESET;

wire [7:0] result;

wire [7:0]operandstemp;

processor1 check(

.ENABLE(ENABLE),

.IMclk(IMclk),

.ALUDMclk(ALUDMclk),

.reset(reset),

.insgrp(insgrp),

.insopc(insopc),

.RDIM(RDIM),

.RDDM(RDDM),

.WRDM(WRDM),

.IRX(IRX),

.result1(result1),

.flagreg(flagreg),

.operands1(operands1),

.operands2(operands2),

.operands3(operands3),

.DM\_databus(DM\_databus),

.IM\_insbus(IM\_insbus),

.DM\_addbus(DM\_addbus),

.PC(PC),

.aluclk(aluclk),

.dmclk(dmclk),

.flag(flag),

.result(result),

.RDLOAD(RDLOAD),

.RDSTORE(RDSTORE),

.ALU(ALU),

.ASSIGN(ASSIGN),

.MOV(MOV),

.BRANCH(BRANCH),

.SPC(SPC),

.RSPC(RSPC),

.SWRESET(SWRESET),

.operandstemp(operandstemp)

);

initial

begin

reset=1'b0;

ENABLE=1'b1;

end

always

begin

#10 IMclk=1'b1;

#10 IMclk=1'b0;

#10 ALUDMclk=1'b1;

#10 ALUDMclk=1'b0;

end

initial begin

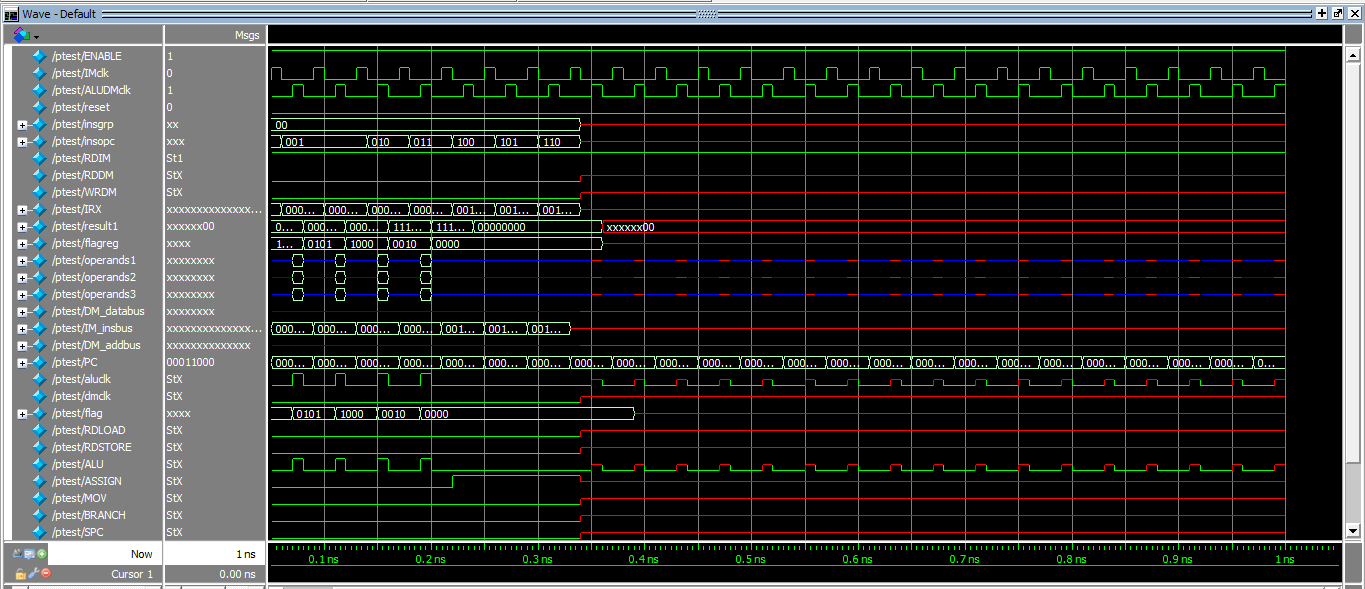
#2000 $stop;

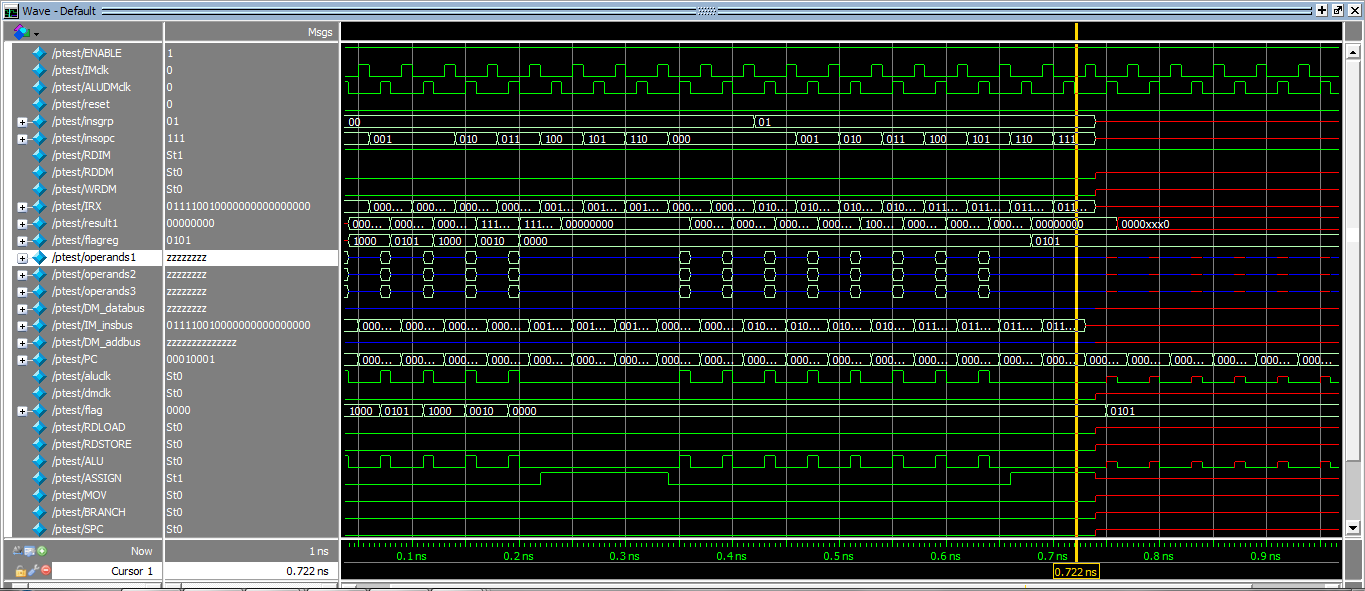
end

endmodule

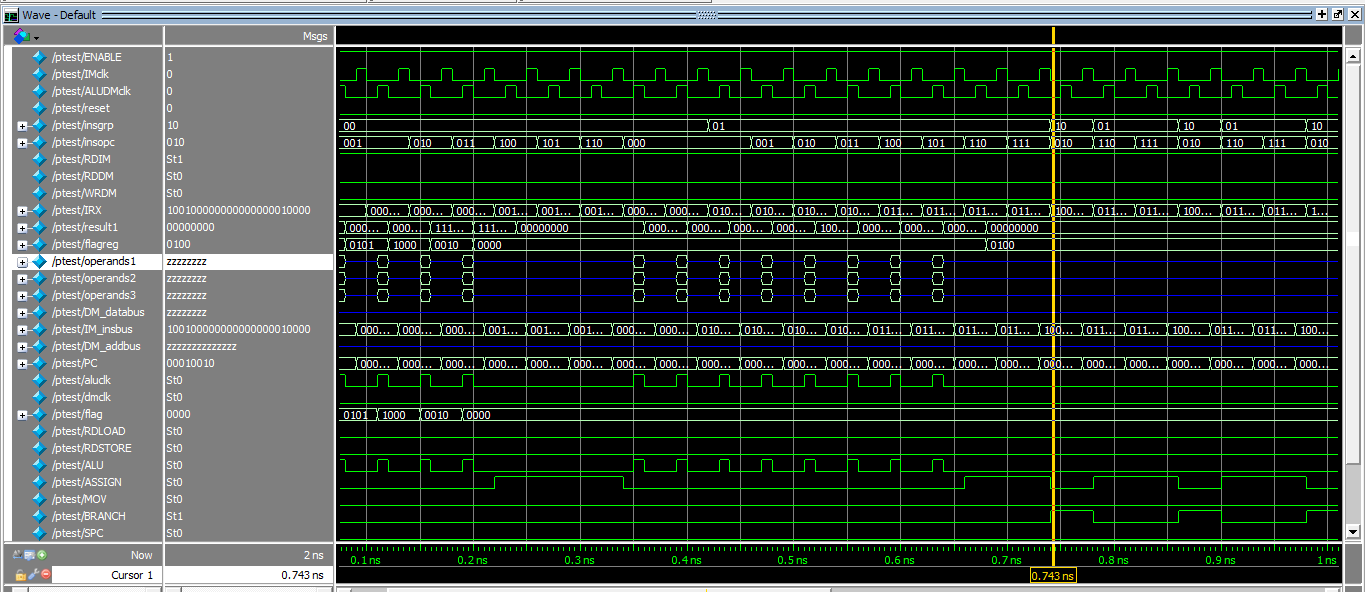
**STIMULUS RESULTS**

ARITHMETIC

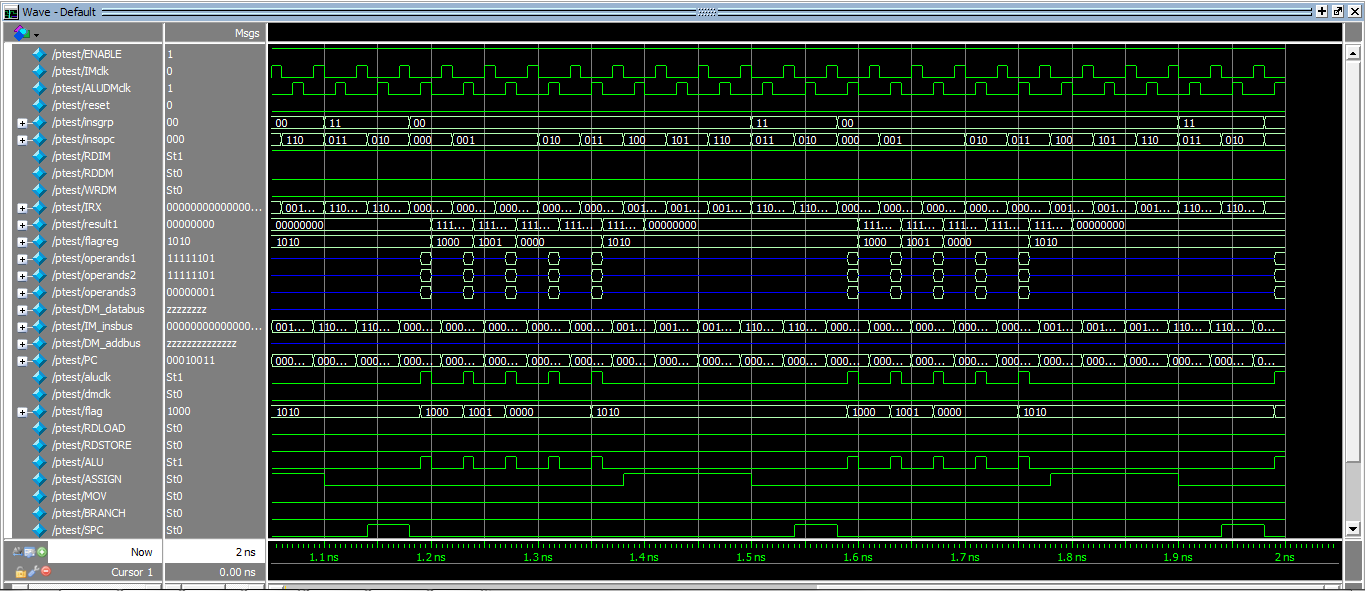


LOGICAL & MISC

BRANCH



MACHINE CONTROL



IMPLEMENTATION ON FPGA

zzzzz