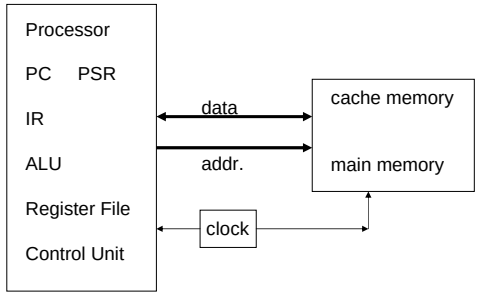
Design a Simple Processor with Verilog at Behavioral Level

Processor Architecture



Instructions

Name	Mnemonic	Opcode	Format(instr. dest/src)	
NOP	NOP	0	NOP	
BRANCH	BRA	1	BRA	mem, cc
LOAD	LD	2	LD	reg, meml
STORE	STR	3	STR	mem, src
ADD	ADD	4	ADD	reg, src
MULTIPLY	MUL	5	MUL	reg, src
COMPLEMENT	CMP	6	CMP	reg, src
SHIFT	SHF	7	SHF	reg, cnt
ROTATE	ROT	8	ROT	reg, cnt
HALT	HLT	9	HLT	

mem – memory address
meml – memory address or immediate value
reg – any register index
src – any register or immediate value
cc – condition code
cnt – shift/rotate count, >= 0 right; <=0 left +/- 16

Condition code (cc)

A	Always	0
C	Carry	1
E	Even	2
P	Parity	3
Z	Zero	4
N	Negative	5

Instruction format

IR[31:28]	Opcode
IR[27:24]	cc
IR[27]	source type 0=reg(mem), 1=imm
IR[26]	destination type 0=reg, 1=imm
IR[23:12]	source address
IR[23:12]	shift/rotate count
IR[11:0]	destination address

Processor status register

PSR[0]	carry
PSR[1]	even
PSR[2]	parity
PSR[3]	zero
PSR[4]	negative

```
// Parameter declaration

parameter       WIDTH = 32;
parameter       CYCLE = 10;
parameter       ADDRSIZE = 12;
parameter       MAXREGS = 16;
parameter       MEMSIZE = 11;

// Register Declaration

reg [WIDTH-1:0] MEM[0:MEMSIZE],
                RFILE[0:MAXREGS-1],
                ir,
                src1, src2;
reg[WIDTH:0]    result;
reg[ADDRSIZE-1:0] pc;
reg[4:0]        psr;
reg             reset;
reg             dir;  //rotate direction
integer         i; // for debugging
```

5

```
// define instruction fields
`define   OPCODE      ir[31:28];
`define   SRC         ir[23:12];
`define   DST         ir[11:0];
`define   SRCTYPE     ir[27];     //source type 0=reg, 1 = imm
`define   DSTTYPE     ir[26];     // destination type, 0= reg, 1=imm
`define   CCODE       ir[27:24];
`define   SRCNT       ir[23:12];  //shift rotate count


//define operand type

`define REGTYPE       0
`define IMMTYPE       1
`define NOP           4'b0000
`define BRA           4'b0001
`define LD            4'b0010
….
…
…
…
```

6

```
//define condition code
`define   CARRY  psr[0]
`define   EVEN   psr[1]
`define   PARITY psr[2]
`define   ZERO   psr[3]
`define   NEG    psr[4]

//define condtion code
`define   CCC    1      //result has carry
`define   CCE    2      //result is even
`define   CCP    3      //result has odd parity
`define   CCZ    4      //result is zero
`define   CCN    5      //result is negative
`define   CCA    0      //always

`define   RIGHT  0      //rotate/shift right
`define   LEFT   1      //rotate/shift left
```

7

```
function [WIDTH-1:0] getsrc;
input [WIDTH-1:0] in;
begin
        if(`SRCTYPE == `REGTYPE)
        begin  getsrc=RFILE(`SRC);
        end
        else begin  // immediate type
        getsrc = `SRC;
        end
end
endfunction

task execute;


task fetch;


task write_result;
```

8

```
task clearcondcode;
begin
        psr =0;
end
endtask


task setcondcode;

initial begin
        $readmemb("program1", MEM);
        $monitor(……..
        …….
end

always begin
        if(!reset) begin
        #CYCLE fetch
        #CYCLE execute
        #CYCLE write_result
        end
        else #CYCLE;
end
```

9

write an assembly program that counts number of 1's
in a given binary number

run the above program on the processor you are going to design.

10

```
module instruction_set_model;

// Declare parameters .
parameter CYCLE = 10;              // Cycle Time
parameter WIDTH = 32 ;             // Width of datapaths
parameter ADDRSIZE = 12 ;          // Size of address fields
parameter MEMSIZE = (1<<ADDRSIZE);     // Size of max memory
parameter MAXREGS = 16;            // Maximum registers
parameter SBITS = 5;               // Status register bits

// Declare Registers and Memory
reg [WIDTH-1:0]  MEM[0:MEMSIZE-1],      // Memory
                 RFILE[0:MAXREGS-1],    // Register File
                 ir,                    // Instruction Register
                 src1, src2 ;           // ALU operation registers
reg [WIDTH: 0] result;                  // ALU result register
reg [SBITS-1:0] psr ;                   // Processor Status Register
reg [ADDRSIZE-1:0] pc ;   // Program counter
reg dir ;                 // rotate direction
reg reset ;               // System Reset
integer i ;               // useful for interactive debugging
```

11

```
// General definitions
`define TRUE      1
`define FALSE     0
`define DEBUG_ON          debug = 1
`define DEBUG_OFF         debug = 0
// Define Instruction fields
`define OPCODE ir[31:28]
`define SRC ir[23:12]

`define DST          ir[11:0]
`define SRCTYPE ir[27] //source type, 0=reg (mem for LD), 1 =imm
`define DSTTYPE ir[26] //destination type, 0=reg, 1=imm
`define CCODE     ir[27:24]
`define SRCNT     ir[23:12] //Shift/rotate count -=left, +=right


// Operand Types
`define REGTYPE           0
`define IMMTYPE           1
```

12

```
// Define opcodes for each instruction
`define NOP     4'b0000
`define BRA     4'b0001
`define LD      4'b0010
`define STR     4'b0011
`define ADD     4'b0100
`define MUL     4'b0101
`define CMP     4'b0110
`define SHF     4'b0111
`define ROT     4'b1000
`define HLT     4'b1001
// Define Condition Code fields
`define CARRY   psr[0]
`define EVEN    psr[1]
`define PARITY  psr[2]
`define ZERO    psr[3]
`define NEG     psr[4]
```

13

```
// Define Condition Codes
                                  // Condition Code set when...
`define CCC             1         // Result has carry
`define CCE             2         // Result is even
`define CCP             3         // Result has odd parity
`define CCZ             4         // Result is Zero
`define CCN             5         // Result is Negative
`define CCA             0         // Always

`define RIGHT           0         // Rotate/Shift Right
`define LEFT            1         // Rotate/Shift Left
```

14

```
// Functions for ALU operands and result
function [WIDTH-1:0] getsrc;
input [WIDTH-1:0] in ;
begin
if (`SRCTYPE === `REGTYPE)
        begin getsrc = RFILE[`SRC] ; end
else begin // immediate type
        getsrc = `SRC ;
        end
end
endfunction

 function [WIDTH-1:0] getdst;
 input [WIDTH-1:0] in ;
 begin
 if (`DSTTYPE === `REGTYPE) begin
         getdst = RFILE[`DST] ; end
 else begin // immediate type
 $display("Error:Immediate data can't be destination."); end
 end
 endfunction
```

15

```
// Functions/tasks for Condition Codes
function checkcondi // Returns 1 if condition code is set.
input [4:0] ccode ;
begin
case (ccode)
`CCC: checkcond = `CARRY ;
`CCE: checkcond = `EVEN ;
`CCP: checkcond = `PARITY ;
`CCZ : checkcond = `ZERO ;
`CCN: checkcond = `NEG ;
`CCA: checkcond = 1 ;
endcase
end
endfunction


task clearcondcode ; // Reset condition codes in PSR.
begin
psr = 0;
end
endtask
```

16

```
task setcondcode ; // Compute the condition codes and set PSR.
input [WIDTH:0] res ;
begin
        `CARRY = res[WIDTH] ;
        `EVEN = -res[0] ;
        `PARITY = ^res ;
        `ZERO = -(|res) ;
        `NEG = res[WIDTH-1] ;
end
endtask

// Main Tasks -fetch, execute, write_result
task fetch ; // Fetch the instruction and increment PC.
begin
ir = MEM[pc] ;
pc = pc + 1 ;
end
endtask
```

17

```
task execute; //Decode and execute the instruction.
begin
        case (`OPCODE)
        `NOP : ;
        `BRA : begin
                if (checkcond(`CCODE) == 1) pc = `DST ;
               end
        `LD : begin
                clearcondcode;
                if (`SRCTYPE) RFILE[`DST] = `SRC ;
                else RFILE[`DST] = MEM[`SRC] ;
                setcondcode({1'b0,RFILE[`DST]});
              end
        `STR :begin
                clearcondcode;
                if (`SRCTYPE) MEM[`DST] = `SRC ;
                else MEM[`DST] = RFILE[`SRC] ;
              end
        `ADD :begin
                clearcondcode;
                src1 = getsrc(ir) ; src2 = getdst(ir) ;
                result = srcl + src2 ;
                setcondcode(result) ;
              end
```

18

```
`MUL :  begin
        clearcondcode;
        src1 = getsrc(ir) ; src2 = getdst(ir) ;
        result = srcl * src2 ;
        setcondcode(result) ;
        end
`CMP :  begin
        clearcondcode;
        src1 = getsrc(ir) ;
        result = -src1 ;
        setcondcode(result) ;
        end
`SHF :  begin
        clearcondcode;
        src1 = getsrc(ir) ;
        src2 = getdst(ir) ;
        i = src1[ADDRSIZE-1:0] ;
        result = (i>=0) ? (src2 >> i) : (src2 << -i);
        setcondcode(result) ;
        end
```

19

```
`ROT :begin
        clearcondcode;
        src1 = getsrc(ir) ; src2 = getdst(ir) ;
        dir = (src1[ADDRSIZE-1] ==0) ? `RIGHT: `LEFT;
        i = (src1[ADDRSIZE-1] ==0) ? src1 : -src1[ADDRSIZE-1:0];
        while (i > 0) begin
        if (dir == `RIGHT) begin result = src2 >> 1 ;
                result [WIDTH-1] = src2[0] ;
                end
        else begin
                result = src2 << 1 ;
                result [0] = src2[WIDTH-1] ;
                end
        i = i -1;
        src2 = result;
        end
        setcondcode(result) ;
        end
`HLT :begin
$display("Halt ...") ;
$stop; end
default: $display("Error : Illegal Opcode.") ;
endcase end
endtask
```

20

```
// Write the result in register file or memory.
task write_result; begin
if ((`OPCODE >= `ADD) && (`OPCODE < `HLT)) begin
if (`DSTTYPE == `REGTYPE) RFILE[`DST] = result;
else MEM[`DST] = result; end
end
endtask

// Debugging help
task apply_reset;
begin
reset = 1 ;
#CYCLE
reset = 0 ; pc = 0 ;
end
endtask
```

21

```
task disprm ;
input rm ;
input [ADDRSIZE-1:0] adrl, adr2 ;
begin
        if (rm == `REGTYPE) begin
        while (adr2 >= adr1) begin
        $display("REGFILE[%d]=%d\n",adr1,RFILE[adr1]) ;
        adr1 = adr1 + 1 ;
        end
        end
      else begin
      while (adr2 >= adr1) begin
            $display("MEM[%d]=%d\n",adr1,MEM[adr1]) ;
            adrl = adrl + 1 ;
      end
      end
 end
 endtask
```

22

```
//Initial and always blocks
initial begin: prog_load
$readmemb("sisc.prog",MEM) ;
$monitor("%d %d %h %h %h", $time, pc, RFILE[0],RFILE[1],RFILE[2]);
apply_reset;
end

always begin: main-process
if (!reset) begin
#CYCLE fetch;
#CYCLE execute;
#CYCLE write_result;
end
else #CYCLE ; end
endmodule
```

23