

Jul 03, 17 20:03

testbench.v

Page 1/1

```

/* testbench for cpu.v */
`timescale 1us/1us
module testBench;

parameter
    STEP = 1;

    reg clock,reset;
    reg [7:0] iport;
    wire [7:0] oport;

/*  cpu~^BM~^Rã,M~^Kã½M~^MãM~^CçãM~^B,ãM~^C¥ãM~^C¼ãM~^C«ãM~^A"ãM~^AM~^WãM~^A|ã@
£è"M~^@ */
    cpu mycpu(
        .clock(clock), .reset(reset),
        .inputPort(iport), .outputPort(oport)
    );

    always begin #(STEP) clock=~clock; end

    initial begin
        $dumpfile("tmp.vcd");
        $dumpvars(0,testBench);
        $display("time clk iport oport");
        $monitor("%4d %b %d %d", $time, clock, iport, oport);
    end

    initial begin
        #STEP clock=0;reset=0;iport=4;//iportæM~^M~^@ã¤$ãM~^@¤22
        #(2*STEP) reset=1;
        #(1024*STEP) $finish;//çM~^D;ém~^YM~^PãM~^C«ãM~^C¼ãM~^CM~^WæM~^YM~^BãM~^A`10
24ãM~^B¹ãM~^CM~^FãM~^CM~^CãM~^CM~^WãM~^A$çµM~^Bã°M~^F
    end
endmodule

```

Jul 27, 17 14:12

cpu.v

Page 1/4

```

/*CPU(top module)*/
#include "instructionSet.h"

module cpu(clock,reset, inputPort, outputPort);

parameter
  InstructionFetch      =      2'b00,
  InstructionDecode=    2'b01,
  Execute              =      2'b11,
  WriteBack            =      2'b10;

parameter
  IMSIZE = 15, //âM-^JêM-^YM-^P256word âM-^CM-^WâM-^C-âM-^B°âM-^C@âM-^C âM-^C;â
M-^CçâM-^C^
  DMSIZE = 255, //âM-^°â@M-^ZâM-^@256byte âM-^CM-^GâM-^C%âM-^BçâM-^C;âM-^CçâM-^C
^
  InstWIDTH = 24, //âM-^Q%â»âém-^U•24bitâM-^°â@M-^Z
  DataWIDTH = 8, //âM-^CM-^GâM-^C%âM-^BçâM-^U•8bit
  oport = 240, // F0
  iport = 241, // F1
  ixAddress = 242, // F2
  iyAddress = 243, // F3

  input clock, reset;
  input [DataWIDTH-1:0] inputPort;
  output [DataWIDTH-1:0] outputPort;

  wire clock, reset;
  wire [DataWIDTH-1:0] inputPort;
  reg [DataWIDTH-1:0] outputPort;

  reg [InstWIDTH-1:0] IM [0:IMSIZE]; //Instruction(Program) Memory
  reg [DataWIDTH-1:0] DM [0:DMSIZE]; //Data Memory

  reg [1:0] currentState,nextState;
  reg [DataWIDTH-1:0] pc;
  reg [InstWIDTH-1:0] ir;

  reg [DataWIDTH-1:0] acc;
  wire [DataWIDTH-1:0] ix,iy;
  reg [1:0] flag; // {Zero,Carry}
  wire [1:0] cz;

  reg [DataWIDTH-1:0] dataReg; //data Reg
  wire [DataWIDTH-1:0] mpxlout, mpx2out, mpx3out;
  wire [DataWIDTH-1:0] dRA,dWA; //data Read Address, dataWriteAddress

  reg [3:0] alucmd;
  wire [DataWIDTH-1:0] alub,aluout;
  reg csel;
  wire aluc;

  wire pcsel;
  wire [DataWIDTH-1:0] mpxpcout,nextpc;
  wire [DataWIDTH-1:0] d_iport,d_oport;
  wire [DataWIDTH-1:0] d_sum,d_n;
  wire [DataWIDTH-1:0] d_i,d_j,d_k;
wire [1:0] cs;

/* DebugçM-^T"âç;âM-^O•ç•M-^ZâM-^I²âM-^BM-^Jâ½M-^SâM-^A| */
  assign d_oport = DM[oport];
  assign d_iport = DM[iport];

```

Monday April 02, 2018

Jul 27, 17 14:12

cpu.v

Page 2/4

```

  assign ix = DM[ixAddress];
  assign iy = DM[iyAddress];
  assign cs=currentState;
  assign d_i=DM[0];
  assign d_j=DM[1];
  assign d_k=DM[2];
  assign d_sum = DM[4];
  assign d_n = DM[5];

/* CPUâM-^FM-^EâM-^PM-^DâM-^CçâM-^B,âM-^C%âM-^C%âM-^CêM-^VM-^SâM-^N%çM-^Z */
  mux2to1_8bit mpx1(.sel(ir[18]), .d0(8'h00), .d1(ix), .y(mpxlout));
  adder8 dataAddress1(.a(mpxlout), .b(ir[15:8]), .c(dmy), .s(dRA));
  mux4to1_8bit mpx2(.sel(ir[19:18]), .d00(ir[15:8]), .d01(acc), .d10(dataReg),
.d11(dataReg), .y(alub));
  mux2to1_8bit mpx3(.sel(ir[16]), .d0(8'h00), .d1(iy), .y(mpx3out));
  adder8 dataAddress2(.a(mpx3out), .b(ir[7:0]), .c(dmy), .s(dWA));

  mux2to1_1bit mpxc(.sel(csel), .d0(1'b0), .d1(flag[1]), .y(aluc)); //ADD,ADC,SU
B,SUC carry-inâM-^G|çM-^PM-^F
  alu8 alu(.cmd(alucmd), .a(acc), .b(alub), .c(aluc), .czflag(cz), .f(aluout) );

  assign pcsel = (ir[23:20]==`B) & ( !ir[19] & !ir[18] | ir[19] & flag[1] | ir[1
8] & flag[0] );
  mux2to1_8bit mpxpc(.sel( pcsel ), .d0(8'b1), .d1(ir[7:0]), .y(mpxpcout));
  adder8 nextAddress(.a(pc), .b(mpxpcout), .c(dmy), .s(nextpc));

/*-----*/
/* âM-^E%âM-^JM-^°ç~âM-^PâM-^A@çM-^J%æM-^EM-^KâM-^BM-^RâM-^OM-^Vâ½M-^W,âM-^G°â
M-^JM-^°ç~âM-^PâM-^A,çM-^J%æM-^EM-^KâM-^BM-^RâM-^G°âM-^JM-^° */
  always @(posedge clock) begin
    DM[iport] <= inputPort;
    outputPort <= DM[oport];
  end

/*-----*/
/* âM-^C^âM-^B»âM-^CM-^CâM-^CM-^Hâç;âM-^O•âM-^OM-^WçM-^PM-^F resetçM-^KâM-^A;â,
M-^KâM-^AM-^LâM-^BM-^JâM-^KM-^Uâ½M-^° */
  always @(negedge reset) resetAction;

/*-----*/
/* âM-^C^âM-^B»âM-^CM-^CâM-^CM-^HâM-^G|çM-^PM-^F : Power on ResetâM-^A%âM-^AM-^
âM-^A" Resetâç;âM-^O• */
  task resetAction;
  begin
    nextState <= InstructionFetch;
    pc <= 8'h00; DM[ixAddress]<=0; DM[iyAddress]=0;
  end
endtask

/*-----*/
/* âM-^B•âM-^CM-^_âM-^C%âM-^C-âM-^C%âM-^B•âM-^C$âM-^C³çµM-^Bâ°M-^FâM-^G|çM-^P
M-^F */
  task finishAction;
  begin
    $display("inputPort(@f0): HEX %2h outputPort(@f1): HEX %2h",inputPort, outputPort);
    $display("inputPort(@f0): DEC %3d outputPort(@f1): DEC %3d", inputPort, outputPort);
    $finish;
  end
endtask

/*-----*/
/* PHASE1 : instruction Fetch âM-^Q%â»ââM-^OM-^Vâ½M-^W */

```

cpu.v

1/2

Jul 27, 17 14:12

cpu.v

Page 3/4

```

task instructionFetch;
begin
    nextState <= InstructionDecode;
    ir<=IM[pc];
end
endtask

/*-----*/
/* PHASE2 : instruction Decode */
task instructionDecode;
begin
    nextState <= Execute;
    dataReg<=DM[dRA];
    case (ir[23:20]) //ALU
    'ADD : alucmd <= 4'h0;
    'ADC : alucmd <= 4'h1;
    'SUB : alucmd <= 4'h2;
    'SUC : alucmd <= 4'h3;
    'AND : alucmd <= 4'h4;
    'OR  : alucmd <= 4'h5;
    'XOR : alucmd <= 4'h6;
    'NOT : alucmd <= 4'h7;
    'LD  : alucmd <= 4'h8;
    default : alucmd <= 4'hf;
    endcase

    case (ir[23:20])
        'ADC ,
        'SUC
        : csel = 1'b1;
        default : csel = 1'b0;
    endcase
    pc <= nextpc;
end
endtask

/*-----*/
/* PHASE3 : Execute */
task execute;
begin
    nextState <= WriteBack;
    case(ir[23:20])
        'LD ,
        'ADD, 'ADC,
        'SUB, 'SUC,
        'AND, 'OR , 'XOR, 'NOT
        : begin acc <= aluout; flag <= cz; end
        'FIN : finishAction;
    endcase
end
endtask

/*-----*/
/* PHASE4 : WriteBack */
task writeback;
begin
    nextState <= InstructionFetch;
    case(ir[23:20])
        'LD ,

```

Jul 27, 17 14:12

cpu.v

Page 4/4

```

        'ADD, 'ADC,
        'SUB, 'SUC,
        'AND, 'OR , 'XOR, 'NOT
        : DM[dWA] <= ir[17] ? acc : DM[dWA];
    endcase
end
endtask

/*-----*/
/* CPU */
always @(posedge clock)
    if (reset) begin currentState<=nextState; end

always @(currentState) begin
    case (currentState)
        InstructionFetch : instructionFetch;
        InstructionDecode : instructionDecode;
        Execute : execute;
        WriteBack : writeback;
        default : resetAction;
    endcase
end

/* sample.bin */
initial $readmemb("sample.bin", IM);
endmodule

```

Apr 02, 18 4:04	submodule.v	Page 1/2
<pre> /* 8bit ALU*/ module alu8(cmd, a,b, c, czflag, f); input [3:0] cmd; input [7:0] a,b; input c; output [1:0] czflag; output [7:0] f; function [8:0] aluCore; // {carry,f} input [3:0] cmd; input [7:0] a,b; casex (cmd) 4'h0: aluCore = a+b; // A+B 4'h1: aluCore = a+b+{7'b0,c}; // A+B+C 4'h2: aluCore = a-b; // A-B 4'h3: aluCore = a-b-{7'b0,c}; // A-B-C 4'h4: aluCore = a&b; // A and B 4'h5: aluCore = a b; // A or B 4'h6: aluCore = a^b; // A xor B 4'h7: aluCore = ~a ; // not A 4'h8: aluCore = b; // acc <- b default: aluCore = 9'bzzzzzzzz; //ãM-^@M-^@reserverd endcase endfunction assign {czflag[1],f} = aluCore(cmd,a,b); assign czflag[0] = (f==8'b0); endmodule module mux4to1_8bit(sel,d00,d01,d10,d11,y); input [1:0] sel; input [7:0] d00,d01,d10,d11; output [7:0] y; function [7:0] mux41; input [1:0] s; input [7:0] x0,x1,x2,x3; case (s) 2'b00 : mux41=x0; 2'b01 : mux41=x1; 2'b10 : mux41=x2; 2'b11 : mux41=x3; endcase endfunction assign y = mux41(sel, d00,d01,d10,d11); endmodule module mux2to1_8bit(sel,d0,d1,y); input sel; input [7:0] d0,d1; output [7:0] y; assign y = sel ? d1:d0; endmodule module mux2to1_1bit(sel,d0,d1,y); input sel; input d0,d1; output y; assign y = sel ? d1:d0; endmodule module adder8(a,b,s,c); input [7:0] a,b; </pre>		

Apr 02, 18 4:04	submodule.v	Page 2/2
<pre> output [7:0] s; output c; assign {c,s}=a+b; endmodule </pre>		

May 01, 17 10:43

instructionSet.h

Page 1/1

```
/* instructionSet.v */
/*
    3210 98 76 5432 1098 7654 3210
    iiii mm nn ssss ssss dddd dddd
*/
#define ADD 4'b0000
#define ADC 4'b0001
#define SUB 4'b0010
#define SUC 4'b0011
#define AND 4'b0100
#define OR 4'b0101
#define XOR 4'b0110
#define NOT 4'b0111
#define LD 4'b1000
#define B 4'b1011
#define FIN 4'b1111
```

Apr 02, 18 4:04		fibonacci.asm		Page 1/1
ADR	HEX			
0	820101	LD #1 @j		
1	820102	LD #1 @k		
2	8A01F0	LOOP: LD @j @oport		
3	8A0100	LD @j @i		
4	8A0201	LD @k @j		
5	0A0002	ADD @i @k		
6	B80002	B C EXIT		
7	B000FB	B A LOOP		
8	8A01F0	EXIT: LD @j @oport		
9	850000	LD acc acc		
10	850000	LD acc acc		
11	850000	LD acc acc		
12	850000	LD acc acc		
13	F00000	FIN		

Apr 02, 18 4:04	fibonacci.bin	Page 1/1
<pre>1000001000000000100000001 1000001000000000100000010 1000101000000000111110000 1000101000000000100000000 1000101000000000100000001 0000101000000000000000010 1011100000000000000000010 1011000000000000011111011 1000101000000000111110000 1000010100000000000000000 1000010100000000000000000 1000010100000000000000000 1000010100000000000000000 1111000000000000000000000 1111000000000000000000000 0</pre>		