CPU design: instruction set, control logic, machine program

1 Today's goal

- Learn the stack architecture of of the tinycpu.
- Learn how to determine and describe the control logic.
- Learn machine programs and assembly language programs.

2 Today's contents

Step 1 Read very carefully the instruction set (Section 3), modules (Section 4), control wires (Section 5), and logic of control wires (Section 6), of tinycpu and understand them.

Step 2 *Check 1* Write miscellaneous definitions (List 1), tinycpu (List 2), block RAM with countdown program (List 3), and its test bench (List 4). Perform the simulation and check if it works correctly.

Step 3 *Check 2* Write top module (List 5) of tinycpu and its UCF (List 6). Implement the bit file in the FPGA and confirm that it works properly.

Step 4 Write an assembly program as follows:

- n is given from input port in.
- n, $2n$, $3n$, ..., n^2 are written to the output buffer in turn, and terminate.

Step 5 Convert the assembly program in Step 4 to a machine program.

Step 6 *Check 3* Perform the simulation for the machine program in Step 5 to confirm that it is correct.

Step 7 *Check 4* Implement the bit file for the machine program in Step 5, and see if it works properly in the FPGA board.

3 Instruction set of tinycpu

Table 1 shows the instruction set of tinycpu. Each instruction works as follows:

HALT The state of state machine state0 changes to IDLE.

PUSHI I (PUSH Immediate) Operand I (Immediate value) is pushed to the stack stack0. Since the operand I has 12 bits while the width of the stack is 16 bits, the sign is extended. More specifically, if the MSB of I is 0, then 4'b0000 is added to extend to 16 bits. If the MSB of I is 1, then 4'b1111 is added no to change the sign.

PUSH A The value of address A in ram0 is pushed to the stack.

POP A The stack top qtop is written in the address A of ram 0. In the same time, stack is popped.

JMP A (JuMP) A is written in program counter pc0. In other words, the program execution jumps to address A.

JZ A (Jump Zero) The program execution jumps to address A if stack top qtop is 0. In the same time, stack0 is popped.

JNZ A (Jump Non-Zero) The program execution jumps to address A if stack top qtop is not zero. In the same time, stack0 is popped.

IN The value of input port in is pushed in stack0.

OUT The value of stack top qtop is written in output buffer obuf0. In the same time, stack0 is popped.

ADD,SUB,MUL,SHL,SHR,BAND, BOR,BXOR

AND,OR The specified binary operation is performed for qtop and qnext and the resulting value is written in stack (i.e.qtop). In the same time, stack0 is popped.

NEG,BNOT,NOT The specified unary operation is performed for qtop and the resulting value is written in stack (i.e.qtop).

Table 1: Instruction set. Mnemonic names and instruction codes

mnemonic		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Hex
1	HALT	0	0	0	0	-												0000
2	PUSHI I	0	0	0	1	I												1000+I
3	PUSH A	0	0	1	0	A												2000+A
4	POP A	0	0	1	1	A												3000+A
5	JMP A	0	1	0	0	A												4000+A
6	JZ A	0	1	0	1	A												5000+A
7	JNZ A	0	1	1	0	A												6000+A
8	IN	1	1	0	1	-												D000
9	OUT	1	1	1	0	-												E000
10	OP f	1	1	1	1	-							f					F000+f
	ADD	1	1	1	1	-							0	0	0	0	0	F000
	SUB	1	1	1	1	-							0	0	0	0	1	F001
	MUL	1	1	1	1	-							0	0	0	1	0	F002
	SHL	1	1	1	1	-							0	0	0	1	1	F003
	SHR	1	1	1	1	-							0	0	1	0	0	F004
	BAND	1	1	1	1	-							0	0	1	0	1	F005
	BOR	1	1	1	1	-							0	0	1	1	0	F006
	BXOR	1	1	1	1	-							0	0	1	1	1	F007
	AND	1	1	1	1	-							0	1	0	0	0	F008
	OR	1	1	1	1	-							0	1	0	0	1	F009
	EQ	1	1	1	1	-							0	1	0	1	0	F00A
	NE	1	1	1	1	-							0	1	0	1	1	F00B
	GE	1	1	1	1	-							0	1	1	0	0	F00C
	LE	1	1	1	1	-							0	1	1	0	1	F00D
	GT	1	1	1	1	-							0	1	1	1	0	F00E
	LT	1	1	1	1	-							0	1	1	1	1	F00F
	NEG	1	1	1	1	-							1	0	0	0	0	F010
	BNOT	1	1	1	1	-							1	0	0	0	1	F011
	NOT	1	1	1	1	-							1	0	0	1	0	F012

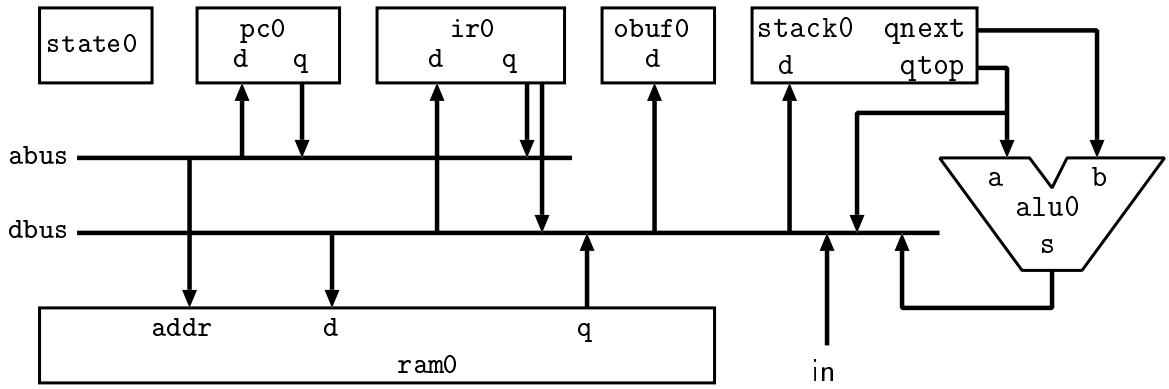

Figure 1: Architecture of tinycpu

4 Modules of tinycpu

Tinycpu uses seven modules in Table 2. The seven modules are:

Program Counter A counter to store the 12-bit address poiting the instruction code in the memory.

Instruction Register A 16-bit register to stores the instruction code to be executed.

State Machine A 3-bit register that holds the current states.

Stack An array of registers to store intermediate values for evaluating formulas.

ALU Combinational circuit to compute the resulting values for specified operation.

Memory A block RAM to store instruction codes of a program and values used by the program.

Output Buffer A 16-bit register to store the output of the program.

5 Control wires

Table 3 shows the control wires. Among them, control wires from 1 to 10 are defined in Table 2 are used to control the behavior of instantiated modules. When the value of a control wire is 1, the flip flops in the module are updated at the rising edge of the clock. Thus, these operations are synchronous.

On the other hand, control wires from 11 to 17 control write operations to either address bus abus or data bus dbus. This operations are asynchronous, that is, if a control wire becomes 1 then the corresponding write operation is performed immediately.

Usually data transfer is performed using two control wires such that one of them is from 1 to 10 and the other is from 11 to 17. For example, to transfer data from ram0 to ir0, control wires dbus2ir and ram2dbus take value 1.

6 Logic of control wires

Finally, we need to decide the logic of control wires. Their logic is determined by the values of

- current state cs (the output of state0),
- irout (the output of ir0), and
- stack top qtop.

Table 4 shows the logic of control wires. The control wires that must take value 1 are shown.

7 Verilog HDL for tinycpu

We are now ready to write the Verilog HDL for tinycpu.

List 1 shows miscellaneous definitions. These definitions can be imported by the following include statement.

```
`include "defs.v"
```

Table 2: Modules and ports

	module	port	i/o	bitwidth	function	connected to
1	pc0	clk	i	1	clock	clk
	(counter)	reset	i	1	reset	reset
	N=12	inc	i	1	increment pc	pcinc
	program	load	i	1	write in pc	abus2pc
	counter	d	i	12	pc input	abus
		q	o	12	pc output	pcout
2	ir0	clk	i	1	clock	clk
	(counter)	reset	i	1	reset	reset
	N=16	load	i	1	write in ir	dbus2ir
	instruction	d	i	16	ir input	dbus
	register	q	o	16	ir output	irout
3	state0	clk	i	1	clock	clk
	(state)	reset	i	1	reset	reset
	state	run	i	1	start execution	run
	machine	halt	i	1	move to IDLE	halt
		cont	i	1	move to EXECB	cont
		cs	o	3	state output	cs
4	stack0	clk	i	1	clock	clk
	(stack)	reset	i	1	reset	reset
	stack	load	i	1	write in qtop	dbus2qtop
		push	i	1	stack push	push
		pop	i	1	stack top	pop
		d	i	16	stack input	dbus
		qtop	o	16	stack top output	qtop
		qnext	o	16	stack second output	qnext
5	alu0	a	i	16	data input	qtop
	(alu)	b	i	16	data input	qnext
	ALU	f	i	5	operation selection	irout[4:0]
		s	o	16	result output	aluout
6	ram0	clk	i	1	clock	clk
	(ram)	load	i	1	write	dbus2ram
	memory	addr	i	12	address input	abus
		d	i	16	data input	dbus
		q	o	16	data output	ramout
7	obuf0	clk	i	1	clock	clk
	(counter)	reset	i	1	reset	reset
	N=16	load	i	1	write	dbus2obuf
	output	d	i	16	data input	dbus
	buffer	q	o	16	data output	out

Table 3: Control wire

	control wire	related modules/buses	operation
1	halt	state0	move from EXECA to IDLE
2	cont	state0	move from EXECA to EXECB
3	pcinc	pc0	increment pc
4	push	stack0	push stack
5	pop	stack0	pop stack
6	abus2pc	abus,pc0	write abus in pc0
7	dbus2ir	dbus,ir0,	write dbus in ir0
8	dbus2qtop	dbus,stack0	write dbus in stack0
9	dbus2ram	abus, dbus,ram0	write dbus in ram0 with address abus
10	dbus2obuf	dbus,obuf0	write dbus in obuf0
11	pc2abus	pc0, abus	write pcout in abus
12	ir2abus	ir0, abus	write irout[11:0] in abus
13	ir2dbus	ir0,dbus	write the sign extension of irout[11:0] in dbus
14	qtop2dbus	stack0,dbus	write qtop in dbus
15	alu2dbus	alu0,dbus	write aluout in dbus
16	ram2dbus	ram0,dbus	write ramout in dbus
17	in2dbus	dbus	write in in dbus

Table 4: The logic of control wire

instruction	FETCHA	FETCHB
all	pcinc, pc2abus	ram2dbus, dbus2ir

instruction	EXECA	condition
HALT	halt	-
PUSHI I	ir2dbus, dbus2qtop, push	-
PUSH A	ir2abus, cont	-
POP A	ir2aubs, qtop2dbus, dbus2ram, pop	-
JMP A	ir2abus, abus2pc	-
JZ A	ir2abus, abus2pc, pop	if qtop==0
JZ A	pop	if qtop!=0
JNZ A	ir2abus, abus2pc, pop	if qtop!=0
JNZ A	pop	if qtop==0
IN	in2dbus, dbus2qtop, push	
OUT	qtop2dbus, dbus2obuf, pop	
OP f	alu2dbus, dbus2qtop, pop	if irout[4]==0
OP f	alu2dbus, dbus2qtop	if irout[4]==1

instruction	EXECB
PUSH A	ram2dbus, dbus2qtop, push

You can also import defs.v in the Verilog HDL of the state machine state.v, and ALU alu.v.

List 1: Miscellaneous definitoionsdefs.v

```
1  `define IDLE 3'b000
2  `define FETCHA 3'b001
3  `define FETCHB 3'b010
4  `define EXECA 3'b011
5  `define EXECB 3'b100
6
7  `define ADD 5'b00000
8  `define SUB 5'b00001
9  `define MUL 5'b00010
10 `define SHL 5'b00011
11 `define SHR 5'b00100
12 `define BAND 5'b00101
13 `define BOR 5'b00110
14 `define BXOR 5'b00111
15 `define AND 5'b01000
16 `define OR 5'b01001
17 `define EQ 5'b01010
18 `define NE 5'b01011
19 `define GE 5'b01100
20 `define LE 5'b01101
21 `define GT 5'b01110
22 `define LT 5'b01111
23 `define NEG 5'b10000
24 `define BNOT 5'b10001
25 `define NOT 5'b10010
26
27 `define HALT 4'b0000
28 `define PUSHI 4'b0001
29 `define PUSH 4'b0010
30 `define POP 4'b0011
31 `define JMP 4'b0100
32 `define JZ 4'b0101
33 `define JNZ 4'b0110
34 `define IN 4'b1101
35 `define OUT 4'b1110
36 `define OP 4'b1111
```

List 2 is the Verilog HDL of tinycpu.v. Seven modules are instantiated. The value of every control wire is detemined by a single always statement. Initially all values of control wires are 0. These values are changed to one according to the values of cs, irout, and qout.

List 2: Verilog HDL tinycpu.v

```
1  `include "defs.v"
2
3  module tinycpu(clk, reset, run, in, cs, pcout, irout, qtop, abus,
       dbus, out);
4
5    input clk,reset,run;
6    input [15:0] in;
7    output [2:0] cs;
8    output [15:0] irout, qtop, dbus, out;
9    output [11:0] pcout, abus;
10   wire [15:0] qnext, ramout, aluout;
11   reg [11:0] abus;
12   reg halt, cont, pcinc, push, pop, abus2pc, dbus2ir,
       dbus2qtop, dbus2ram, dbus2obuf, pc2abus, ir2abus,
       ir2dbus, qtop2dbus, alu2dbus, ram2dbus, in2dbus;
13
14   counter #(12) pc0(.clk(clk),.reset(reset),.load(abus2pc),.inc(
       pcinc),.d(abus),.q(pcout));
15   counter #(16) ir0(.clk(clk),.reset(reset),.load(dbus2ir),.inc(0)
       ,.d(dbus),.q(irout));
16   state state0(.clk(clk),.reset(reset),.run(run),.cont(cont),.halt(
       halt),.cs(cs));
17   stack stack0(.clk(clk),.reset(reset),.load(dbus2qtop),.push(
       push),.pop(pop),.d(dbus),.qtop(qtop),.qnext(qnext));
18   alu alu0(.a(qtop),.b(qnext),.f(irout[4:0]),.s(aluout));
19   ram #(16,12,4096) ram0(.clk(clk),.load(dbus2ram),.addr(
       abus),.d(dbus),.q(ramout));
20   counter #(16) obuf0(.clk(clk),.reset(reset),.load(dbus2obuf),.
       inc(0),.d(dbus),.q(out));
21
22   always @(pc2abus or ir2abus or pcout or irout)
23     if(pc2abus) abus = pcout;
24     else if(ir2abus) abus = irout[11:0];
25     else abus = 12'hxxx;
26
27   assign dbus = ir2dbus ? {{4{irout[11]}},irout[11:0]} : 16'
       hzzzz;
28   assign dbus = qtop2dbus ? qtop : 16'hzzzz;
29   assign dbus = alu2dbus ? aluout : 16'hzzzz;
30   assign dbus = ram2dbus ? ramout : 16'hzzzz;
31   assign dbus = in2dbus ? in : 16'hzzzz;
32
33   always @(cs or irout or qtop)
34   begin
35     halt = 0; cont = 0; pcinc = 0; push = 0; pop = 0;
         abus2pc = 0; dbus2ir = 0; dbus2qtop = 0; dbus2ram
          = 0; dbus2obuf = 0; pc2abus = 0; ir2abus = 0;
         ir2dbus = 0; qtop2dbus = 0; alu2dbus = 0;
         ram2dbus = 0; in2dbus = 0;
36     if(cs == `FETCHA)
37       begin
38         pcinc = 1; pc2abus = 1;
39       end
40     else if(cs == `FETCHB)
41       begin
42         ram2dbus = 1; dbus2ir = 1;
43       end
44     else if(cs == `EXECA)
45       case(irout[15:12])
46       `PUSHI:
47         begin
48           ir2dbus = 1; dbus2qtop = 1; push = 1;
49         end
50       `PUSH:
51         begin
52           ir2abus = 1; cont = 1;
53         end
54       `POP:
55         begin
56           ir2abus = 1; qtop2dbus = 1; dbus2ram = 1; pop =
                 1;
57         end
58       `JMP:
59         begin
60           ir2abus = 1; abus2pc = 1;
```

```
61          end
62        `JZ:
63          begin
64            if(qtop == 0)
65              begin
66                ir2abus = 1; abus2pc = 1;
67              end
68            pop = 1;
69          end
70        `JNZ:
71          begin
72            if(qtop != 0)
73              begin
74                ir2abus = 1; abus2pc = 1;
75              end
76            pop = 1;
77          end
78        `IN:
79          begin
80            in2dbus = 1; dbus2qtop = 1; push = 1;
81          end
82        `OUT:
83          begin
84            qtop2dbus = 1; dbus2obuf = 1; pop = 1;
85          end
86        `OP:
87          begin
88            alu2dbus = 1; dbus2qtop = 1;
89            if(irout[4] == 0) pop = 1;
90          end
91        default:
92          halt = 1;
93        endcase
94      else if(cs == `EXECB)
95        if(irout[15:12]==`PUSH)
96          begin
97            ram2dbus = 1; dbus2qtop = 1; push = 1;
98          end
99
100   end
101
102 endmodule
```

8 Machine program and assembly language program

Next program contains the assembly language program and the corresponding machine program for countdown. *Assembly language program* uses mnemonic (PUSHI, PUSH, etc), and labels (L1, n, etc). Machine program is a list of 4-digit hexadecimal numbers, which is obtained from the assembly language program. Each number is assigned to each memory. In this program, the memory with address 000-00C are used. The address 00C are used to store the value of n.

In this program, the value obtained from input port in is written in n. After that, the values of n, n-1, . . ., 0 are written in the output buffer.

```
000:D000            IN
001:300C            POP n
002:200C      L1:   PUSH n
003:E000            OUT
004:200C            PUSH n
005:500B            JZ L2
006:200C            PUSH n
007:1001            PUSHI 1
008:F001            SUB
009:300C            POP n
00A:4002            JMP L1
00B:0000      L2:   HALT
00C:0000      n: 0
```

List 3 shows the block RAM initialized this machine program.

List 3: Block RAM ram.v for countdown program

```
1  module ram(clk, load, addr, d, q);
2    parameter DWIDTH=16,AWIDTH=12,WORDS=4096;
3
4    input clk,load;
5    input [AWIDTH−1:0] addr;
6    input [DWIDTH−1:0] d;
7    output [DWIDTH−1:0] q;
8    reg [DWIDTH−1:0] q;
9    reg [DWIDTH−1:0] mem [WORDS−1:0];
10
11   always @(posedge clk)
12     begin
13       if(load) mem[addr] <= d;
14       q <= mem[addr];
15     end
16
17   integer i;
18   initial begin
19     for(i=0;i<WORDS;i=i+1)
20       mem[i]=12'h000;
21     mem[12'h000] = 16'hD000; // IN
22     mem[12'h001] = 16'h300C; // POP n
23     mem[12'h002] = 16'h200C; // L1: PUSH n
24     mem[12'h003] = 16'hE000; // OUT
25     mem[12'h004] = 16'h200C; // PUSH n
26     mem[12'h005] = 16'h500B; // JZ L2
27     mem[12'h006] = 16'h200C; // PUSH n
28     mem[12'h007] = 16'h1001; // PUSHI 1
29     mem[12'h008] = 16'hF001; // SUB
30     mem[12'h009] = 16'h300C; // POP n
31     mem[12'h00A] = 16'h4002; // JMP L1
32     mem[12'h00B] = 16'h0000; // L2: HALT
33     mem[12'h00C] = 16'h0000; // n: 0
34   end
35
36 endmodule
```

List 4: Test bench tinycpu_tb.v for tinycpu

```
1  `timescale 1ns / 1ps
2  module tinycpu_tb;
3
4    reg clk,reset,run;
5    reg [15:0] in;
6    wire [2:0] cs;
7    wire [15:0] irout, qtop, dbus, out;
8    wire [11:0] pcout, abus;
9
10   tinycpu tinycpu0(.clk(clk), .reset(reset), .run(run), .in(in), .cs
         (cs), .pcout(pcout), .irout(irout), .qtop(qtop), .abus(
         abus), .dbus(dbus), .out(out));
11
12   initial begin
13     clk = 0;
14     forever #50 clk =~ clk;
15   end
16
17   initial begin
18     reset = 0; run = 0; in = 3;
19     #100 reset = 1; run = 1;
20     #100 run = 0;
21   end
22
23 endmodule
```

9 Implementing tinycpu in the FPGA

Table 5 shows the mapping between ports of tinycpu, top module cpu_top, lcdctrl, and IO devices of the FPGA board.

List 5: Top module cpu_top.v of tinycpu.v

```
1  module cpu_top(CLK50MHZ,ROT_A,ROT_CENTER,
       BTN_SOUTH,BTN_EAST,BTN_NORTH,BTN_WEST,SW
       ,LED,LCD_E,LCD_RS,LCD_RW,SF_D);
2
3    input CLK50MHZ,ROT_A,ROT_CENTER,BTN_SOUTH,
         BTN_EAST,BTN_NORTH,BTN_WEST;
4    input [3:0] SW;
5    output [4:0] LED;
6    output LCD_RS,LCD_E,LCD_RW;
7    output [11:8] SF_D;
8    wire clk, reset;
9    wire [2:0] cs;
10   wire [6:0] in;
11   wire [15:0] data0,data3,data1,data2,data4,data5;
12
13   chattering #(8) chattering0(.clk(CLK50MHZ),.reset(reset),.
         in({BTN_WEST,BTN_NORTH,BTN_EAST,SW,~
         ROT_A}),.out({in,clk}));
14   lcdctrl lcdctrl0(.clk(CLK50MHZ), .reset(reset), .lcd_e(LCD_E
         ), .lcd_rs(LCD_RS), .lcd_rw(LCD_RW), .sf_d(SF_D), .
         data0(data0), .data1(data1), .data2(data2), .data3(
         data3), .data4(data4), .data5(data5));
15   tinycpu tinycpu0(.clk(clk), .reset(reset), .run(ROT_CENTER)
         , .in({9'h000,in}), .cs(cs), .pcout(data0), .irout(data1),
         .qtop(data2), .abus(data3), .dbus(data4), .out(data5));
16
17   assign reset = ~BTN_SOUTH;
18   assign LED[4] = (cs==`IDLE);
19   assign LED[3] = (cs==`FETCHA);
20   assign LED[2] = (cs==`FETCHB);
21   assign LED[1] = (cs==`EXECA);
22   assign LED[0] = (cs==`EXECB);
23
24 endmodule
```

List 6: UCF cpu_top.ucf for cpu_top.v(Spartan-3A Starter kit)

```
1  # ROT PUSH BUTTON
2  NET "ROT_A" LOC = "T13" | IOSTANDARD = LVTTL |
       PULLUP ;
3  NET "ROT_CENTER" LOC = "R13" | IOSTANDARD =
       LVTTL | PULLDOWN ;
4
5  # PUSH SWITCH
6  NET "BTN_SOUTH" LOC = "T15" | IOSTANDARD =
       LVTTL | PULLDOWN ;
7  NET "BTN_EAST" LOC = "T16" | IOSTANDARD = LVTTL
       | PULLDOWN ;
8  NET "BTN_NORTH" LOC = "T14" | IOSTANDARD =
       LVTTL | PULLDOWN ;
9  NET "BTN_WEST" LOC = "U15" | IOSTANDARD = LVTTL
       | PULLDOWN ;
10
11 # Slide SWITH
12 NET "SW<0>" LOC = "V8" | IOSTANDARD = LVTTL |
       PULLUP ;
13 NET "SW<1>" LOC = "U10"| IOSTANDARD = LVTTL |
       PULLUP ;
14 NET "SW<2>" LOC = "U8" | IOSTANDARD = LVTTL |
       PULLUP ;
15 NET "SW<3>" LOC = "T9" | IOSTANDARD = LVTTL |
       PULLUP ;
16
17 # LED
18 NET "LED<4>" LOC = "V19" | IOSTANDARD = LVTTL |
       SLEW = QUIETIO | DRIVE = 4 ;
19 NET "LED<3>" LOC = "U19" | IOSTANDARD = LVTTL |
       SLEW = QUIETIO | DRIVE = 4 ;
20 NET "LED<2>" LOC = "U20" | IOSTANDARD = LVTTL |
       SLEW = QUIETIO | DRIVE = 4 ;
21 NET "LED<1>" LOC = "T19" | IOSTANDARD = LVTTL |
       SLEW = QUIETIO | DRIVE = 4 ;
22 NET "LED<0>" LOC = "R20" | IOSTANDARD = LVTTL |
       SLEW = QUIETIO | DRIVE = 4 ;
23
24 # CLOCK
25 NET "CLK50MHZ" LOC = "E12"| IOSTANDARD =
       LVCMOS33 ;
26 NET "CLK50MHZ" PERIOD = 20.0ns HIGH 40%;
27
28 # LCD
29 NET "LCD_E" LOC = "AB4" | IOSTANDARD = LVTTL |
       DRIVE = 4 | SLEW = QUIETIO ;
```

Table 5: tinycpu and output module

ports of tinycpu (* indicates chattering removal)	i/o	ports of cpu_top or lcdstrl	IO devices of FPGA bord
clk*	i	ROT_A	Rotary encoder of rotary push switch
reset	i	~BTN_SOUTH	South push switch
run	i	ROT_CENTER	Push switch of rotary push switch
in[3:0]*	i	SW[3:0]	Slide switches
in[4]*	i	BTN_EAST	East push switch
in[5]*	i	BTN_NORTH	North push switch
in[6]*	i	BTN_WEST	West push switch
cs == 'IDLE	o	LED[4]	4th LED
cs == 'FETCHA	o	LED[3]	3rd LED
cs == 'FETCHB	o	LED[2]	2nd LED
cs == 'EXECA	o	LED[1]	1st LED
cs == 'EXECB	o	LED[0]	0th LED
pcout(program counter)	o	data0	top left of LCD
irout(instruction register)	o	data1	top center of LCD
qtop(stack top)	o	data2	top right of LCD
abus(address bus)	o	data3	bottom left of LCD
dbus(data bus)	o	data4	bottom center of LCD
out(output buffer)	o	data5	bottom right of LCD

```
30 NET "LCD_RS" LOC = "Y14" | IOSTANDARD = LVTTL |
       DRIVE = 4 | SLEW = QUIETIO ;
31 NET "LCD_RW" LOC = "W13" | IOSTANDARD = LVTTL |
       DRIVE = 4 | SLEW = QUIETIO ;
32 NET "SF_D<8>" LOC = "AA12" | IOSTANDARD = LVTTL
       | DRIVE = 4 | SLEW = QUIETIO ;
33 NET "SF_D<9>" LOC = "Y16" | IOSTANDARD = LVTTL |
       DRIVE = 4 | SLEW = QUIETIO ;
34 NET "SF_D<10>" LOC = "AB16" | IOSTANDARD =
       LVTTL | DRIVE = 4 | SLEW = QUIETIO ;
35 NET "SF_D<11>" LOC = "Y15" | IOSTANDARD = LVTTL
       | DRIVE = 4 | SLEW = QUIETIO ;
```

10 Homework

Homework 1 Illustrate the flow of data in Figure 1 for each of the instructions, PHSHI, PUSH, POP, JUMP, IN, OUT, and OP.

Homework 2 Let $f_1, f_2, \ldots$ be a Fibonacci numbers such that $f_1 = f_2 = 1$ and $f_{i+2} = f_{i+1} + f_i$ $(i \geq 3)$. Write the assembly program as follows:

- n is given from in.
- $f_1, f_2, \ldots, f_n$ are written in the output buffer in turn.

Convert the assembly program to the machine program. Perform the simulation to confirm that it works properly.