Last lecture – a simple microprocessor
“Putting it all together” – A microprocessor in SystemVerilog

- An even simpler CPU
• PC - Program Counter
• ACC - Accumulator (Register)
• ALU - Arithmetic and Logic Unit
• IR - Instruction Register
• MDR - Memory Data Register
• MAR - Memory Address Register
• Sequencer - State Machine
• System Bus
• Control Signals - from sequencer to other units
• Flags - from units to sequencer
Instructions

- **C:**

  \[ a = b + c; \]

- **Assembler:**

  ```
  LD b
  ADD c
  ST a
  ```

- **Machine Code (not AVR nor ARM)**

  ```
  LD b        000 00001
  ADD c       010 00010
  ST a        001 00011
  ```
Machine code

- Simple example:
  - 8 bit instructions
    - 3 bit opcode, 5 bit data
    
    | Instruction | Opcode | Data |
    |-------------|--------|------|
    | LD          | 000    |      |
    | ADD         | 010    |      |
    | ST          | 001    |      |
    | b           | 00001  |      |
    | c           | 00010  |      |
    | a           | 00011  |      |
Unit Structure
Control Signals

ACC_bus  Drive bus with contents of ACC. (Enable three state output.)
load_ACC  Load ACC from bus.
PC_bus  Drive bus with contents of PC.
load_IR  Load IR from bus.
load_MAR  Load MAR from bus.
MDR_bus  Drive bus with contents of MDR.
load_MDR  Load MDR from bus.
ALU_ACC  Load ACC with result from ALU.
INC_PC  Increment PC, and save the result in PC.
Addr_bus  Drive bus with operand part of instruction held in IR.
CS  Chip Select. Use contents of MAR to set up memory address.
R_NW  Read, Not Write. When false, contents of MDR are stored
ALU_add  Perform an add operation in the ALU.
ALU_sub  Perform a subtract operation in the ALU.
module PC #(parameter WORD_W = 8, OP_W = 3)
    (input logic clock, n_reset, PC_bus, load_PC, INC_PC,
     inout wire [WORD_W-1:0] sysbus);

logic [WORD_W-OP_W-1:0] count;

assign sysbus = PC_bus ? {{OP_W{1'b0}},count} : {WORD_W{1'bZ}};

always_ff @(posedge clock, negedge n_reset)
begin
    if (~n_reset)
        count <= 0;
    else
        if (load_PC)
            if (INC_PC)
                count <= count + 1;
            else
                count <= sysbus;
end
endmodule
Sequencer

1. **Fetch**
   - MAR $\leftarrow$ PC
   - PC $\leftarrow$ PC + 1

2. **CS**
   - R_NW

3. **IR**
   - MDR

4. **MAR**
   - Addr

5. **op = store**
   - MAR $\leftarrow$ MAR
   - MDR $\leftarrow$ ACC

6. **CS**
   - R_NW

7. **op = load**
   - IR $\leftarrow$ MDR
   - ACC $\leftarrow$ MDR

8. **op = add**
   - ACC $\leftarrow$ MDR + ACC
   - ACC $\leftarrow$ MDR - ACC
Branching

1. CS R_NW

2. op = load

3. 1: ACC ← MDR

4. 0: z_flag

5. 0: PC ← MDR

6. 1: ACC ← MDR + ACC

7. 0: ACC ← MDR - ACC

8. 1: op = add

9. 0: op = bne

10. 1: ACC ← MDR

11. 0: ACC ← MDR
module sequencer #(parameter OP_W = 3) 
  (input logic clock, n_reset, z_flag, 
   input logic [OP_W-1:0] op, 
   output logic ACC_bus, load_ACC, 
   PC_bus, load_PC, load_IR, load_MAR, 
   MDR_bus, load_MDR, ALU_ACC, ALU_add, 
   ALU_sub, INC_PC, Addr_bus, CS, R_NW);

`include "opcodes.v"

typedef enum {s0, s1, s2, s3, s4, s5, s6, s7, 
   s8, s9, s10} state_type; 
(* syn_encoding="sequential" *)
state_type Present_State, Next_State;
`define LOAD 3'b000
`define STORE 3'b001
`define ADD 3'b010
`define SUB 3'b011
`define BNE 3'b100
always_ff @(posedge clock, negedge n_reset) 
begin: seq
    if (~n_reset)
        Present_State <= s0;
    else
        Present_State <= Next_State;
end

always_comb 
begin: com 
    // reset all the control signals to default
    ACC_bus = 1'b0;
    load_ACC = 1'b0;
    PC_bus = 1'b0;
    load_PC = 1'b0;
    load_IR = 1'b0;
    load_MAR = 1'b0;
    MDR_bus = 1'b0;
    load_MDR = 1'b0;
    ALU_ACC = 1'b0;
    ALU_add = 1'b0;
    ALU_sub = 1'b0;
    INC_PC = 1'b0;
    Addr_bus = 1'b0;
    CS = 1'b0;
    R_NW = 1'b0;
    Next_State = Present_State;
case (Present_State)
  s0: begin
    PC_bus = 1'b1;
    load_MAR = 1'b1;
    INC_PC = 1'b1;
    load_PC = 1'b1;
    Next_State = s1;
  end
  s1: begin
    CS = 1'b1;
    R_NW = 1'b1;
    Next_State = s2;
  end
  s2: begin
    MDR_bus = 1'b1;
    load_IR = 1'b1;
    Next_State = s3;
  end
  s3: begin
    Addr_bus = 1'b1;
    load_MAR = 1'b1;
    if (op == `STORE)
      Next_State = s4;
    else
      Next_State = s6;
  end
  s4: begin
    ACC_bus = 1'b1;
    load_MDR = 1'b1;
    Next_State = s5;
  end
  s5: begin
    CS = 1'b1;
    Next_State = s0;
  end
  s6: begin
    CS = 1'b1;
    R_NW = 1'b1;
    if (op == `LOAD)
      Next_State = s7;
    else if (op == `BNE)
      begin
        if (z_flag == 1'b0)
          Next_State = s9;
        else
          Next_State = s10;
      end
    else
      Next_State = s8;
  end
  s7: begin
    MDR_bus = 1'b1;
    load_ACC = 1'b1;
    Next_State = s0;
  end
  s8: begin
    MDR_bus = 1'b1;
    ALU_ACC = 1'b1;
    load_ACC = 1'b1;
    if (op == `ADD)
      ALU_add = 1'b1;
    else if (op == `SUB)
      ALU_sub = 1'b1;
      Next_State = s0;
  end
  s9: begin
    MDR_bus = 1'b1;
    load_PC = 1'b1;
    Next_State = s0;
  end
  s10: Next_State = s0;
endcase
end
endmodule