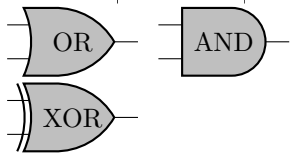


Bit	Decimal	Hex
0	1	1
1	2	16
2	4	256
3	8	4096
4	16	65.536
5	32	Hex
6	64	Hex
7	128	Hex
8	256	Hex
9	512	Hex
10	1024	Hex
11	2048	Hex
12	4096	Hex
13	8192	Hex
14	16.384	Hex
15	32.768	Hex
16	65.536	Hex

Binary	Decimal	Hex
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	a
1011	11	b
1100	12	c
1101	13	d
1110	14	e
1111	15	f



$50 \bmod 3 \rightarrow 50/3 = 16.6667$
 $\rightarrow 16.6667 - 16 = 0.6667$
 $\rightarrow 0.6667 * 3 = 2$
 lw \rightarrow longest insn.

```

module coffee_mealy(
    input clk,
    input reset,
    input insert,
    input [1:0] coins,
    output reg coffee,
    output [2:0] state_display
);

    localparam STAT0 = 0,
    STAT1 = 1,
    STAT2 = 2;

    localparam COIN0 = 2'b00,
    COIN1 = 2'b10,
    COIN2 = 2'b01;

    reg insert_prv;
    reg [2:0] state_next;
    reg [2:0] state_curr;
    reg coffee_next;

    assign state_display = state_curr;

    //reset active HIGH
    //transition on: insert HIGH && clock edge
    //Update output on: posedge clk

    //Initial state -> coins is 0
    always @(posedge clk) begin
        //Reset?
        if(reset) begin
            insert_prv <= 1'b0;
            coffee <= 1'b0;
            state_curr <= 1'b0;
        end
        else if (insert && ~insert_prv) begin
            insert_prv <= insert;
            state_curr <= state_next;
            coffee <= coffee_next;
        end
        else begin
            insert_prv <= insert;
            state_curr <= state_curr;
            coffee <= coffee;
        end
    end

    always @(*) begin
        case (state_curr)
            STAT1: begin
                case (coins)
                    COIN1: begin
                        state_next = STAT2;
                        coffee_next = 0;
                    end
                    COIN2: begin
                        state_next = STAT0;
                        coffee_next = 1;
                    end
                    default: begin
                        state_next = STAT1;
                        coffee_next = 0;
                    end
                endcase
            end
            endcase

            STAT2: begin
                case (coins)
                    COIN1: begin
                        state_next = STAT0;
                        coffee_next = 1;
                    end
                    COIN2: begin
                        state_next = STAT2;
                        coffee_next = 0;
                    end
                    default: begin
                        state_next = STAT2;
                        coffee_next = 0;
                    end
                endcase
            end
            endcase

            default: begin
                case (coins)
                    COIN1: begin
                        state_next = STAT1;
                        coffee_next = 0;
                    end
                    COIN2: begin
                        state_next = STAT2;
                        coffee_next = 0;
                    end
                    default: begin
                        state_next = STAT0;
                        coffee_next = 0;
                    end
                endcase
            end
            endcase
        endcase
    end

    assign out = out_next; // For fast changes
  
```

```

$display : print the immediate values
$strobe : print the values at the end of the current timestep
$monitor : print the values at the end of the current timestep if any values changed.
$monitor can only be called once; sequential call will override the previous.
$write : same as $display but doesn't terminate with a newline (\n)
$display("display a:%h b:%h @ %0t", a, b, $time);

%d -> decimal %t -> time %h -> hex %b -> binary %f -> real %c -> char %s -> string

$finish(1) -> Exit + Prints simulation time and location
$time -> Scaled Time Integer
$stime -> Scaled Time Real
  
```

Isns Reg Order

add rd, rs1, rs2

bne rs1, rs2, IMM

ns	MHz
1	1000
10	100
100	10
1000	1

Twos compliment

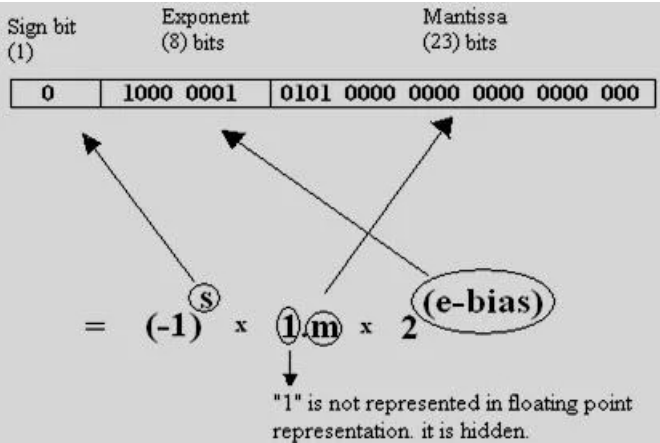
1. Make it bin 5 = 0101
2. NOT it 1010
3. Add 1 1011
4. fill out with 1s 11111111011

Cache Timing

CPI = average number of cycles per instruction;
 $T_{exec} = N_{instr} \times CPI \times T_{cycle};$
 $CPI = CPI_{ideal} + CPI_{stall};$
For pipelined processor: $CPI_{ideal} = 1;$
 $CPI_{stall} = \%reads \times miss_rate_read \times mis_spenalty_read + \%writes \times miss_rate_write \times miss_penalty_write;$
Cache misses increase CPI by adding stall cycles;
Averge Memory access time = Time for a hit + Miss rate x Miss penalty

Cache Addressing

Address_size = N_tag_bits + N_index_bits + N_word_offset_bits + Byte_offset;
 $N_blocks_in_cache = 2^{N_index_bits};$
 $N_word_in_block = log_2(words);$
Direct_memory → associativity = 1;
Cache_size = N_blocks_in_cache * associativity * (N_bits_in_word * N_words_in_block + N_tag_bits + Valid_bit + dirty_bit);



```
`timescale ins / ips
module custom_counter_v1_0_tb();
//stuff you need
reg arstn = 1'b0;
reg [2:0] write_prot = 3'd0; //type of write(leave at 0)
reg write_addr_valid = 1'b0; //master indicating address is valid
custom_counter_v1_0 #(
    .number_of_clk_cycles(10) //custom wires
) custom_counter_v1_0_inst (
    // Define wires
    .leds(leds),
);
always
#5 aclk <= ~aclk;
initial
begin
arstn = 0;
#20 arstn = 1;
axi_write(32'd4,1'b1);
axi_read(32'd8, output_data);
#200
$finish;
end
```

- We can re-design 8 block direct mapped cache

Index	V bit	Tag	Data
000	Y	00	Mem (#00000)
001	N		
010	N		
011	N		
100	Y	00	Mem (#00100)
101	N		
110	N		
111	N		

Memory blocks with address ending with 000 are strictly placed to single cache block 000

Memory blocks with address ending with 00 are freely placed to one of the available cache block in set 00

- Into 2-way associative cache
 - Containing 4 sets of 2 blocks each

Index	V bit	Tag	Data	V bit	Tag	Data
00	Y	000	Mem (#00000)	Y	001	Mem (#00100)
01	N			N		
10	N			N		
11	N			N		

a set of two blocks

```
// TASKS
task axi_read;
input [31:0] addr; //Defines here
output reg [31:0] data;
begin
#3 read_addr <= addr; //puts adress on the bus
read_addr_valid <= 1'b1; //indicating address is valid
read_data_ready <= 1'b1; //indicating ready for a response
wait(read_addr_ready); //wait till slave is ready to read the address
@(posedge aclk); //handshake occurs
read_addr_valid <= 1'b0;
wait(read_data_valid);
@(posedge aclk);
data <= read_data;
@(posedge aclk);
read_data_ready <= 1'b0;
end
endtask
endmodule
```

Exam exercise for bne, beq. Be careful

PC →

```
12: Lb11: add t0, t2, t3
16:      add s1, s2, s3
20:      sub s0, s2, s3
24:      bne s0, s1, Lb11
```

Two ways to generate immediate value for binary instruction:

imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode
---------	-----------	-----	-----	--------	----------	---------	--------

Imm = Target addr - PC

Computing: 12 - 24 = -12_{dec}

Two's complement: -12_{dec} is 111111111010_{bin}