Computer Architecture HW2: Simple ALU Unit

TA: Ting Wang (王珽)
Email: d10943014@ntu.edu.tw

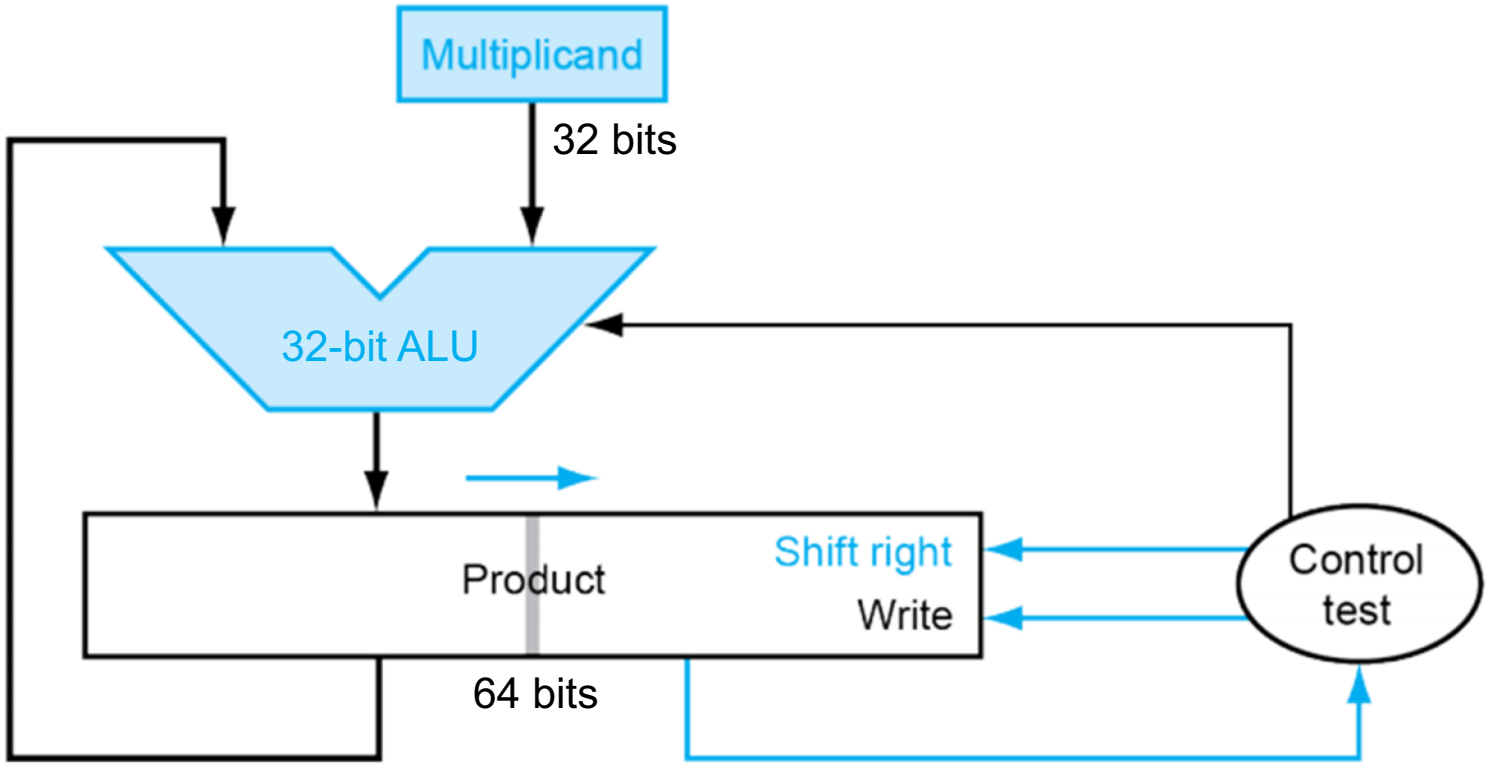
Goal

- Implement an unsigned multiplier
- Implement an unsigned divider
- Implement AND/OR operation
- Combine the 4 operations into single unit (ALU)

- In this homework, you will learn:
  - How to use assignments with `wire`
  - How to use always blocks with `reg`
  - How to describe combinational and sequential circuits
  - How to control state machine
  - How to develop a good coding style

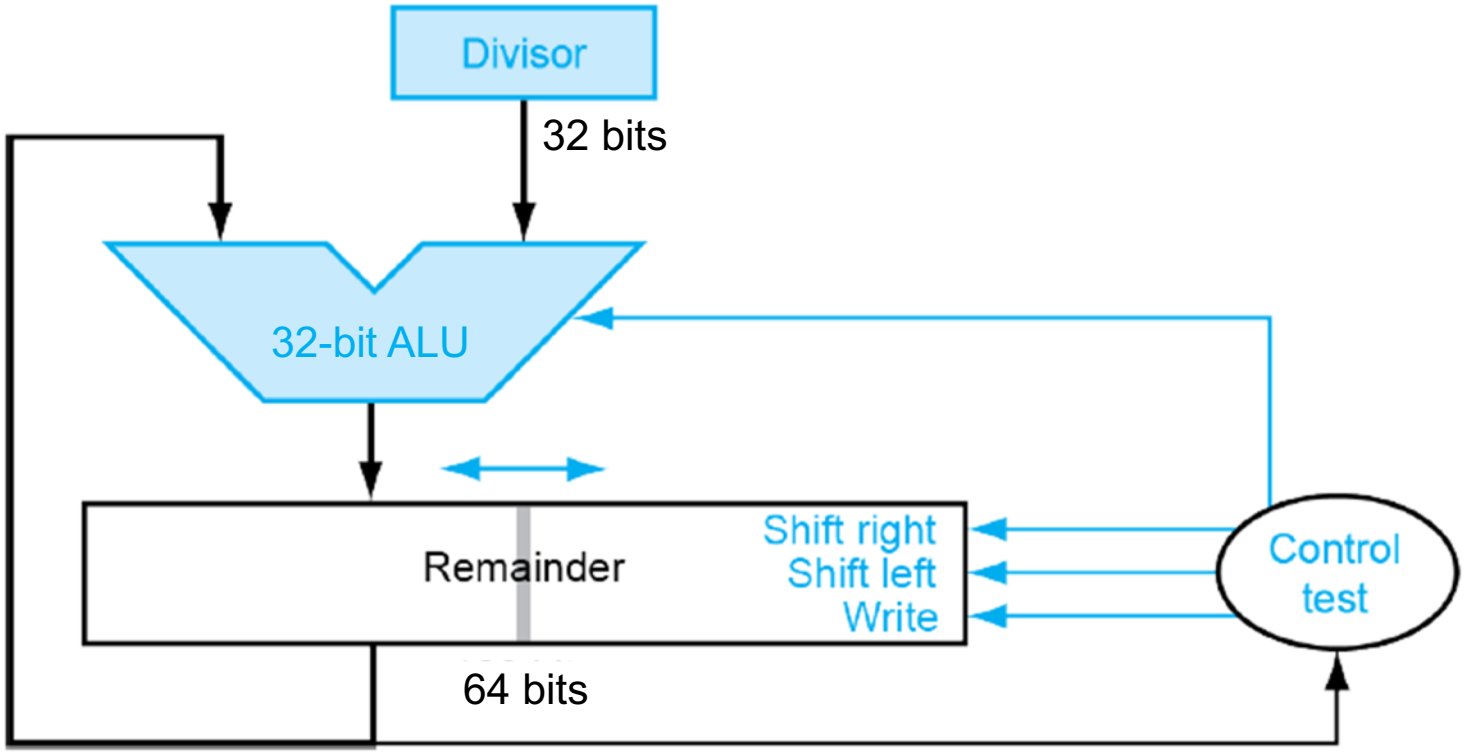
Quick Review: Multiplication Circuit

- Reduced register consumption
- Reduced size of ALU



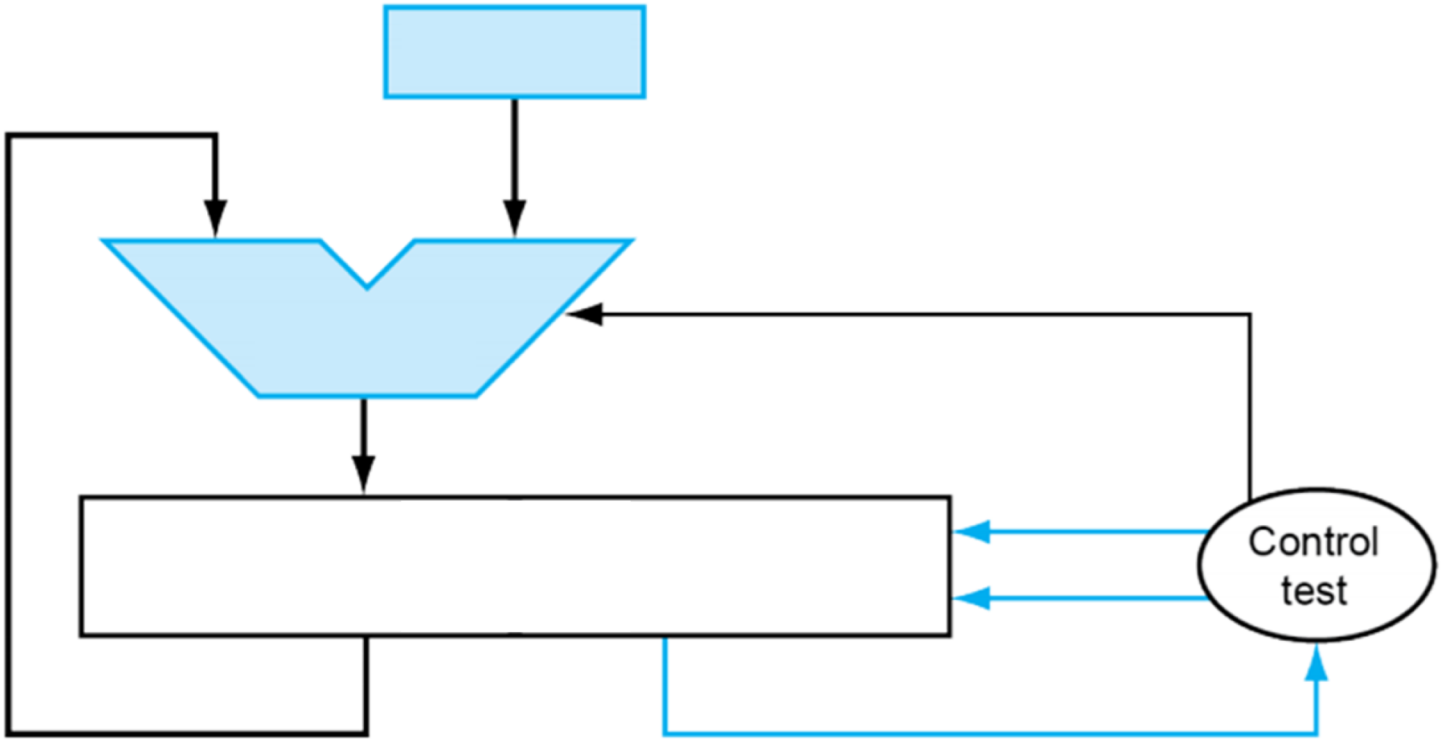
Quick Review: Division Circuit

- Reduced register consumption
- Reduced size of ALU



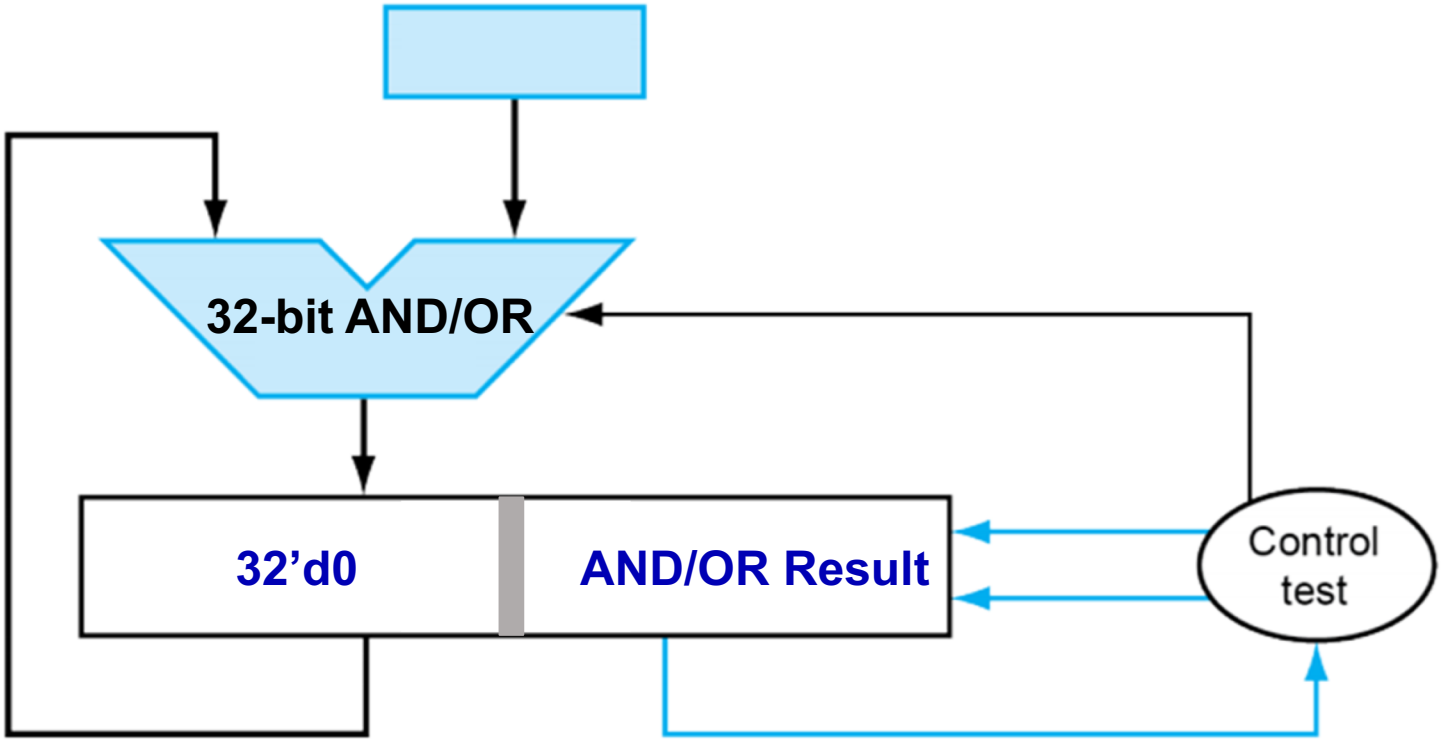
Observation

- The structures of both multiplier and divider are similar
  - Operand register
  - Shift register
  - ALU
- Reuse the hardware!

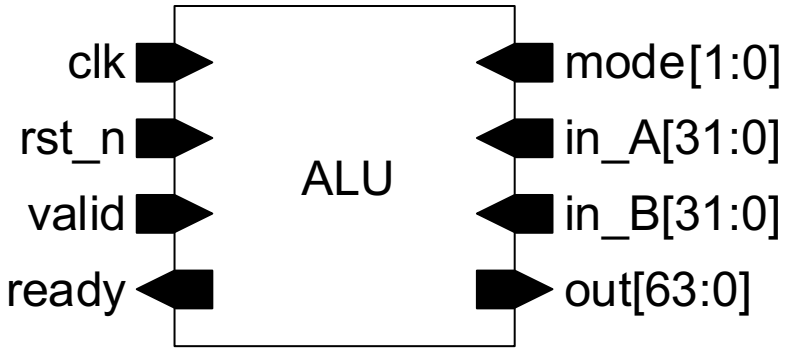


AND/OR Operation

- 32-bit ALU can also perform AND/OR Operation
- The most significant 32 bits of Remainder register should remain 32'd0
- Share the hardware with multiplication/division



Specification



Name	I/O	Width	Description
clk	I	1	Positive edge-triggered clock
rst_n	I	1	Asynchronous negative edge reset
valid	I	1	Valid input data when valid = 1
ready	O	1	Ready output data when ready = 1
mode[1:0]	I	2	0: multiplication, 1: division 2: AND, 3: OR
in_A[31:0]	I	32	Input A
in_B[31:0]	I	32	Input B
out[63:0]	O	64	Output

Specification

- Multiplication
  - Unsigned input (32b)
  - out = in_A × in_B (32b × 32b → 64b)
- Division
  - Unsigned input (32b)
  - quotient = in_A / in_B (quotient is 32b)
  - remainder = in_A % in_B (remainder is 32b)
  - out = {remainder, quotient}
- AND
  - out = {32'd0, in_A & in_B}
- OR
  - out = {32'd0, in_A | in_B}

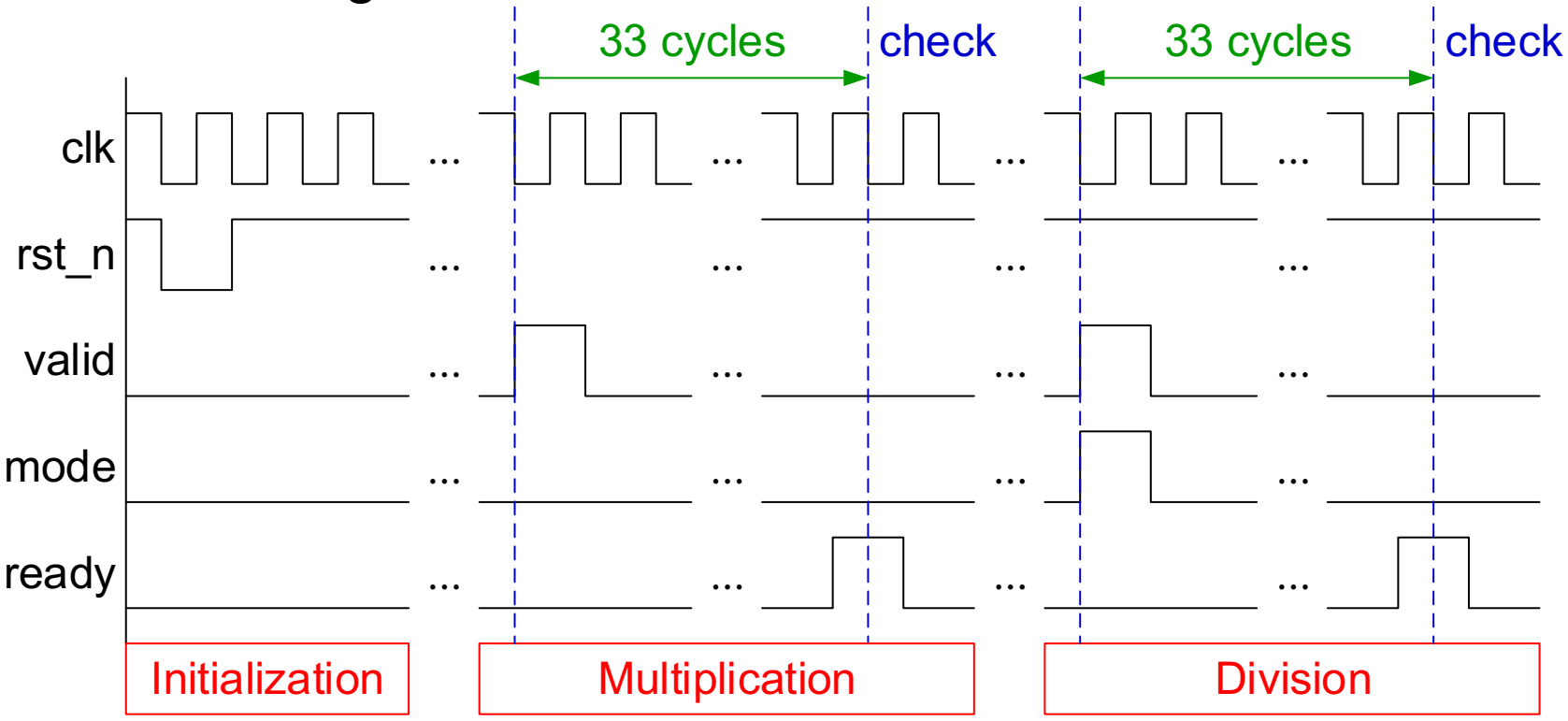
Design Issues

- Require 1 cycle to load data
- Require 32 cycles to do multiplication/division
- Require 1 cycle to do AND/OR
- Require 1 cycle to output data

- Consider CPU is issuing a task to ALU
  - Valid input to ALU for only 1 cycle
  - Expect to receive ready output from multiplication/division after 32 cycles
  - Expect to receive ready output from AND/OR after 1 cycle

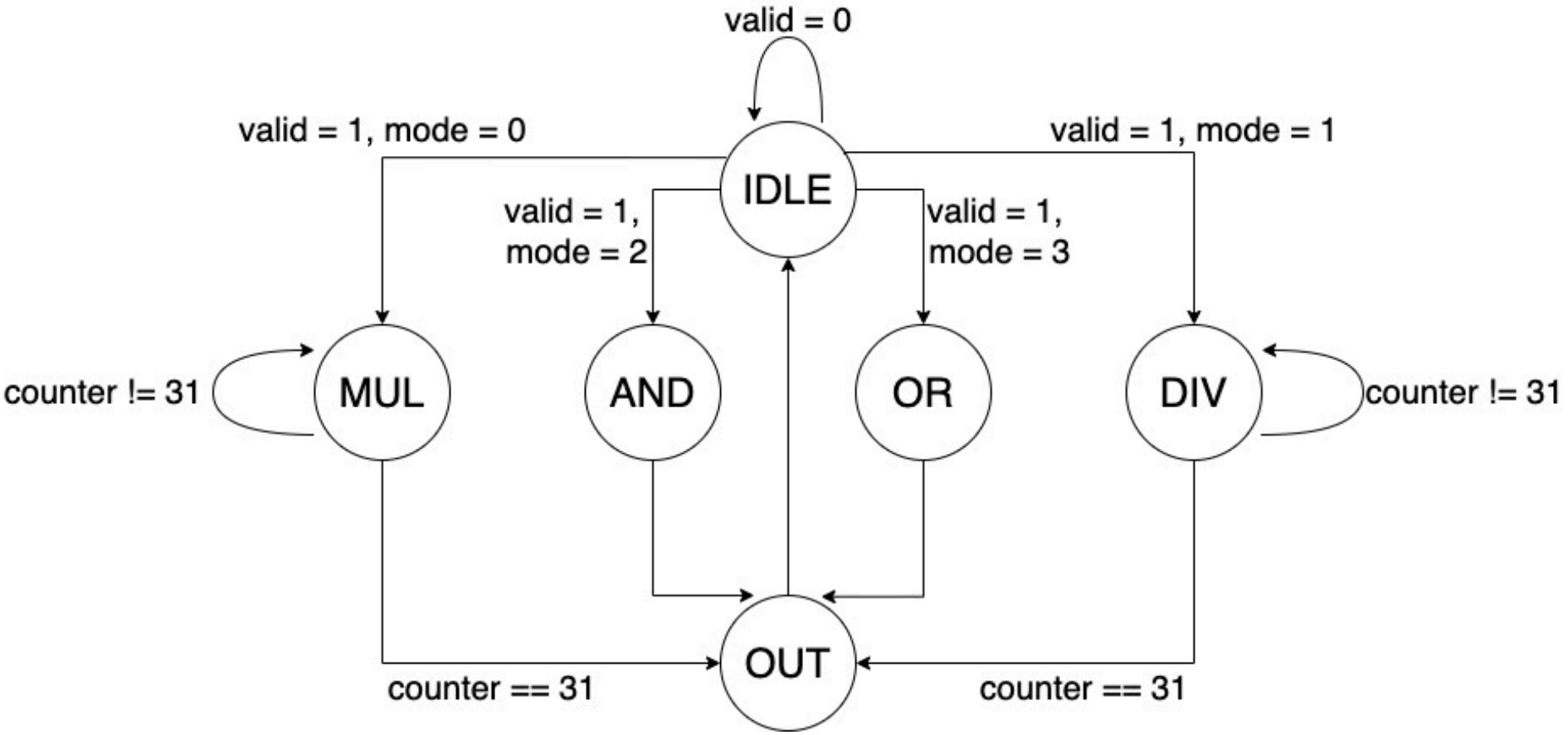
Waveform

- The testbench checks the result after 33 cycles once valid changes to 1 for multiplication/division
- The testbench checks the result after 2 cycles once valid changes to 1 for AND/OR



Finite State Machine (FSM)

- Requires a counter to record the process



Todo 1: FSM

- Define your finite state machine
- Notice the syntax

```
// Definition of states
parameter IDLE = 3'd0;
parameter MUL  = 3'd1;
parameter DIV  = 3'd2;
parameter AND = 3'd3;
parameter OR = 3'd4;
parameter OUT  = 3'd5;
```

```
always @(*) begin
    case(state)
        IDLE: begin

        end
        MUL :
        DIV :
        AND :
        OR  :
        OUT : state_nxt = IDLE;
        default :
    endcase
end
```

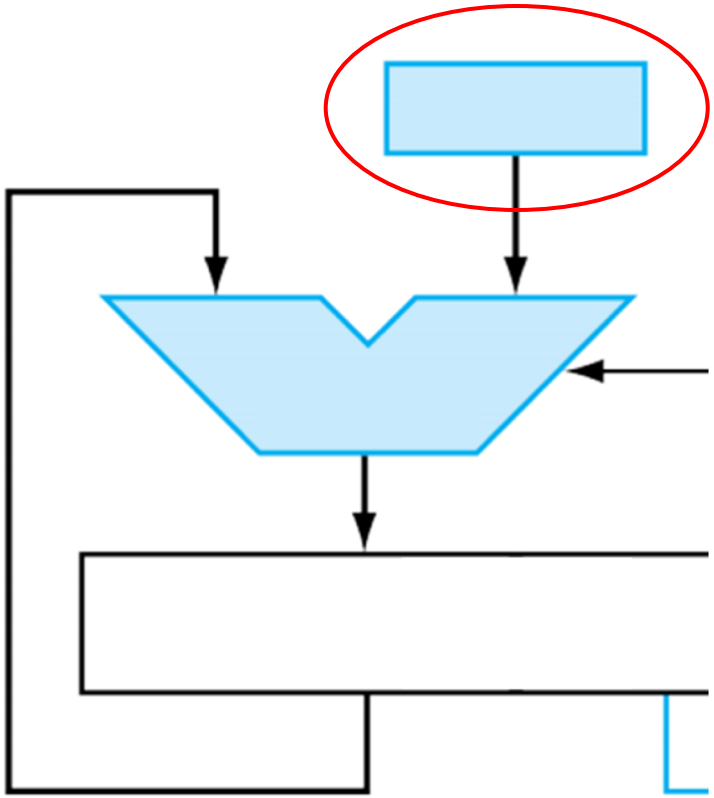
```
// Todo: Sequential always block
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        state <= IDLE;
    end
    else begin
        state <= state_nxt;
    end
end
```

Todo 2: Counter

- Hint
  - Counter counts from 0 to 31 when the state is MUL or DIV
  - Otherwise, keep it zero
- Please describe it with Verilog

Operand Register (alu_in)

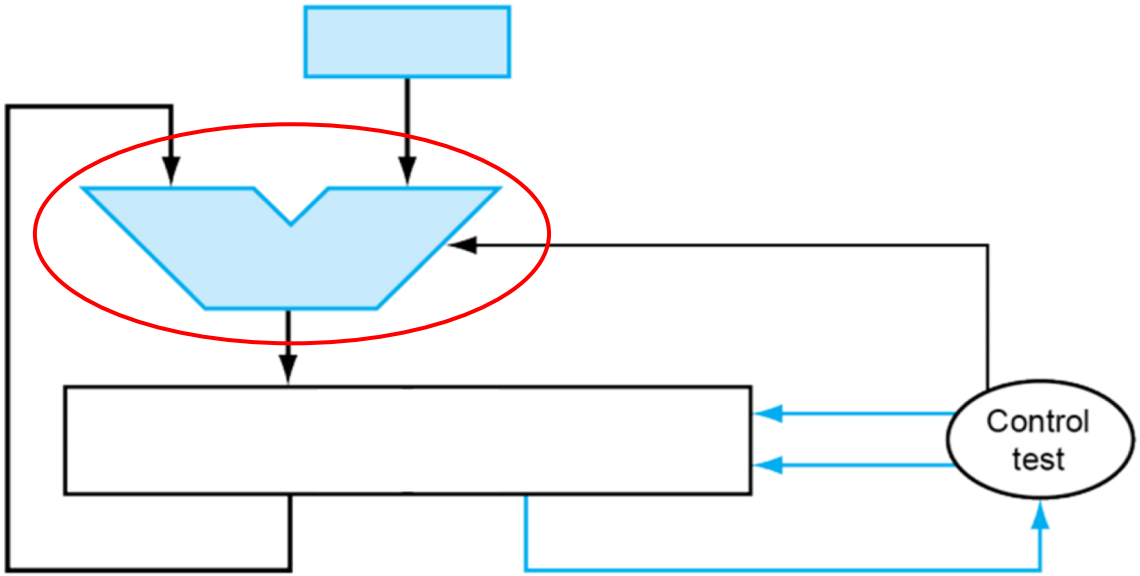
- Load data when the state is IDLE and valid = 1
- When the state is back to IDLE, clear it



```
// ALU input
always @(*) begin
    case(state)
        IDLE: begin
            if (valid) alu_in_nxt = in_B;
            else       alu_in_nxt = 0;
        end
        OUT : alu_in_nxt = 0;
        default: alu_in_nxt = alu_in;
    endcase
end
```

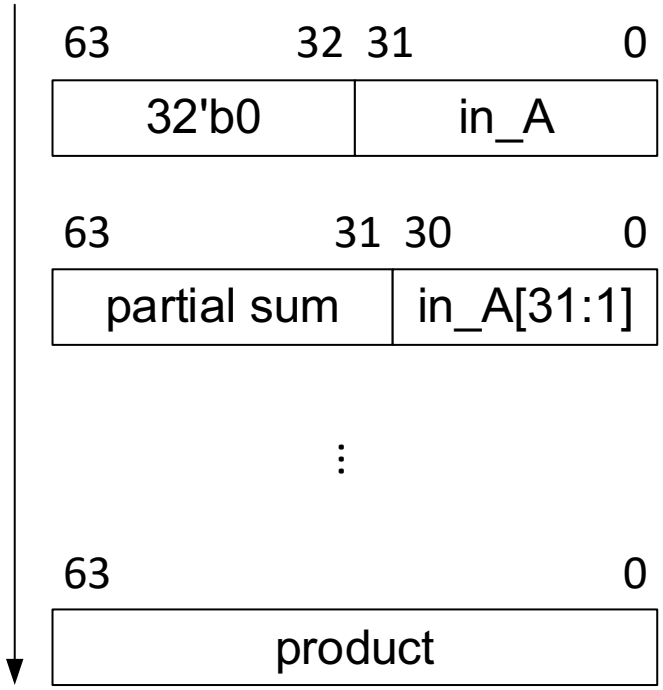
Todo 3: ALU

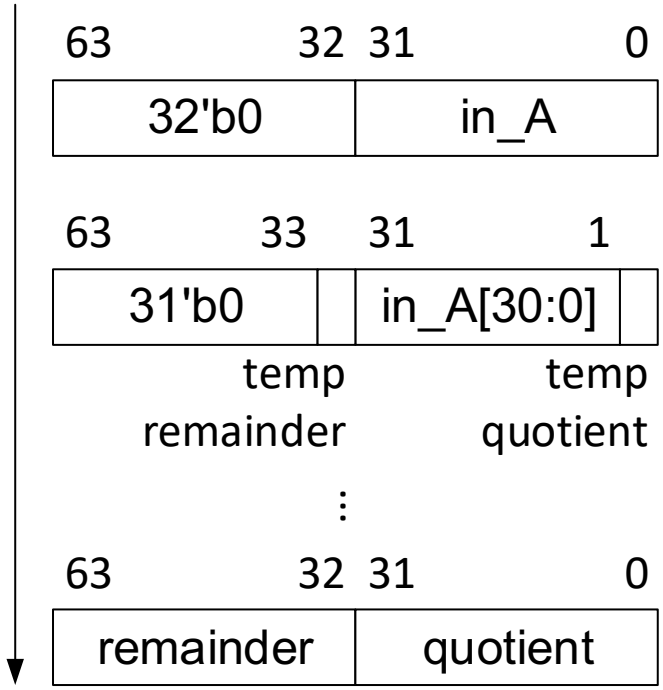
- Multiplication: addition / Division: subtraction
- AND/OR operation
- You should consider
  - Bit width of ALU (to handle overflow)
  - Which bit range of shift register should be transmitted to ALU
  - The operation in ALU given current state



Todo 4: Shift Register

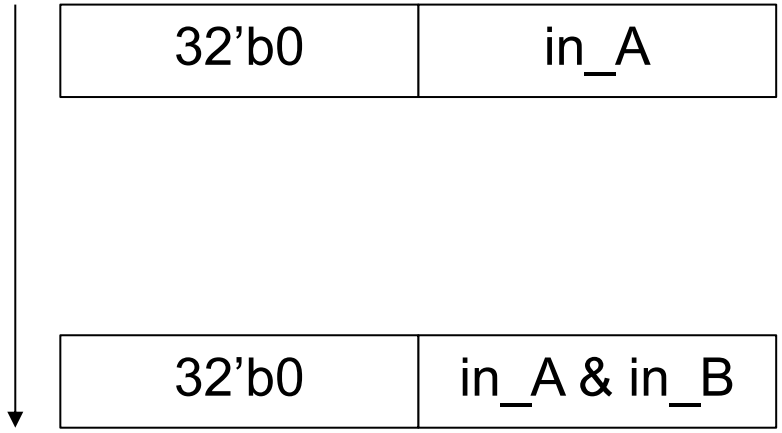
- Load data to the low 32 bits when the state is IDLE and valid = 1
- Multiplication: update data and right shift
- Division: update data and left shift

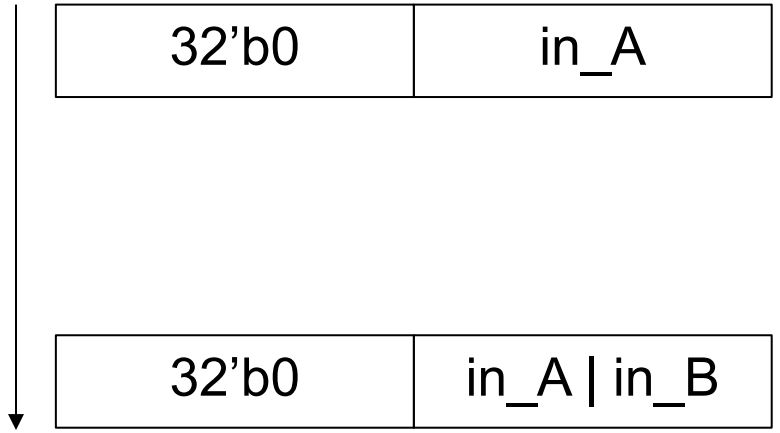




Todo 4: Shift Register (Cont.)

- Load data to the low 32 bits when the state is IDLE and valid = 1
- AND: update data
- Division: update data





Todo 5: Output Control

- Relate output port “out” to shift register
- When is your circuit “ready” to output?

- For such simple description, one can use wire assignments
  - Hint: output is default wire-typed

(Optional) How to Instantiate a Module

- See how testbench is written
- If needed, you can also create a sub-module and instantiate in your top design

```
module test_HW2();
    reg         clk, rst_n;
    reg  [31:0] A, B;
    wire [63:0] Z;
    reg         valid;
    reg  [1:0]  mode;
    wire        ready;
    wire [63:0] product;
    reg  [63:0] product_ans;
    wire [31:0] quotient, remainder;
    reg  [31:0] quotient_ans, remainder_ans;
    wire [63:0] and_temp;
    reg  [63:0] and_ans;
    wire [63:0] or_temp;
    reg  [63:0] or_ans;

    parameter num_data = `NUM_DATA;
    integer i, delay_num, err_mul, err_div, err_and, err_or;
```

```
ALU U0(
    .clk(clk),
    .rst_n(rst_n),
    .valid(valid),
    .ready(ready),
    .mode(mode),
    .in_A(A),
    .in_B(B),
    .out(Z)
);
```

Simulation

- There are two files in folder named “code”
  - HW2.v (homework)
  - HW2_tb.v (testbench)
- To run simulation, you should run source command in advance
  - `$ source license.cshrc` (use given file)
- Verilog simulation
  - `$ ncverilog HW2_tb.v HW2.v +access+r`

Modify the Testbench

- The testbench will run `NUM_DATA multiplications and `NUM_DATA divisions
- Between the time checking the result and the new valid input, there will be at most `MAX_DELAY cycles

```
// You can modify NUM_DATA and MAX_DELAY
`define NUM_DATA 10
`define MAX_DELAY 3
```

- Change input to what you want
  - Example: multiplication

```
// Multiplication
$display("-----------------------------------------");
$display("Test function of multiplication...");
for (i=1; i<=`NUM_DATA; i=i+1) begin
    delay_num = $abs($random()%`MAX_DELAY)+1;
    #(`CLCYE_TIME*delay_num)
    valid = 1;
    A = $random()%32'hFFFF_FFFF; // change your pattern here
    B = $random()%32'hFFFF_FFFF; // change your pattern here
```

Report (Snapshot, at Least One Table)

Register Name	Type	Width	Bus	MB	AR	AS	SR	SS	ST
alu_in_reg	Flip-flop	32	Y	N	Y	N	N	N	N
counter_reg	Flip-flop	5	Y	N	Y	N	N	N	N
shreg_reg	Flip-flop	64	Y	N	Y	N	N	N	N
state_reg	Flip-flop	2	Y	N	Y	N	N	N	N

- All sequential elements must be flip-flops
- Check by Design Compiler
- Command:
  - `$ dv -no_gui`
  - `design_vision> read_verilog HW2.v`
- Exit:
  - `design_vision> exit`

Submission

- Deadline: 1 pm on Dec. 20 (Mon.), 2021
  - No late submission allowed
- Upload HW2_<student_id>.zip to ceiba
  - HW2_<student_id>.zip
    - HW2_<student_id>/
    - HW2_<student_id>/HW2.v
    - HW2_<student_id>/report.pdf
- Example

```
[r08003@cad29 HW2]$ unzip HW2_r08943003.zip
Archive:  HW2_r08943003.zip
   creating: HW2_r08943003/
  inflating: HW2_r08943003/HW2.v
 extracting: HW2_r08943003/report.pdf
[r08003@cad29 HW2]$
```

Score

- TA will check if your design is reasonable (50 %)
  - Fits the architecture
  - Does NOT exist initial block
  - Does NOT exist delay declaration
  - Does NOT exist operators including “*” and “/”
  - Does NOT exist latches
  - **Functionality would only be checked if the demands above are satisfied**
- `NUM_DATA and `MAX_DELAY will be randomly chosen to check your functionality (50 %)
  - Default:
    `define NUM_DATA 10
    `define MAX_DELAY 3