Computer Architecture HW2: Arithmetic Logic Unit (ALU)

TA: Yung-Chin Chen (陳永縉)

Email: b08901061@ntu.edu.tw

Due Monday, 11/20, 23:59

Data Preparation

- Decompress HW2.zip
  - `$ unzip HW2.zip`
- Direction hierarchy:
  - 00_TB
    - HW2_tb.v ➔ testbench file
    - Pattern ➔ test pattern
      - I0
      - I1
      - …
  - 01_RTL
    - 00_license.f ➔ EDA tool license source command
    - 01_run.f ➔ vcs/ncverilog command
    - 99_clean_up.f ➔ Command to clean temporary data
    - HW2.v ➔ Your design

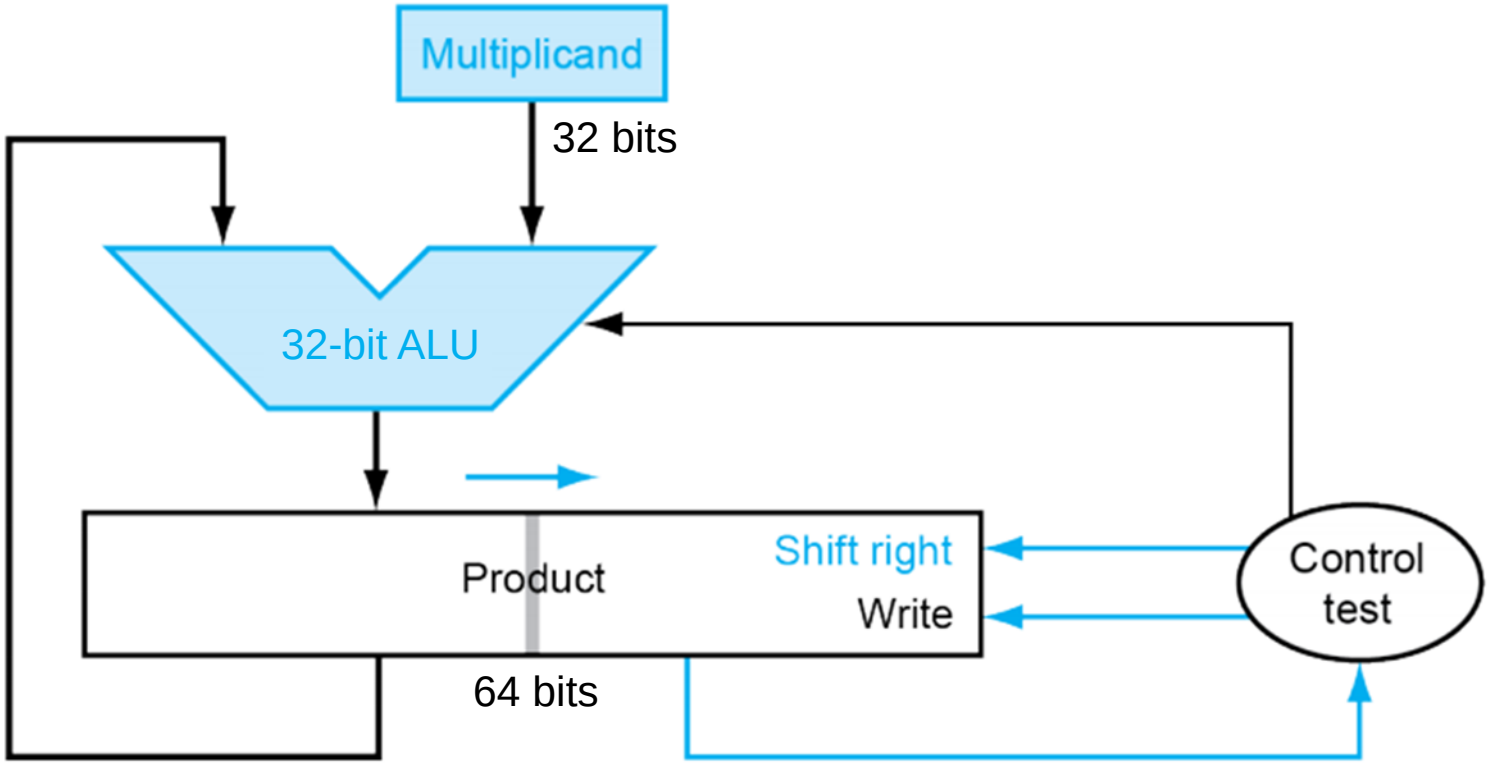
Goal

- To implement common arithmetic operations
  - Signed: **ADD**, **SUB**, **SLT**, **SRA**
  - Unsigned: **AND**, **OR**
  - Multicycle unsigned **MUL** and **DIV**
- Combine the 8 operations above into a single unit (ALU)

- In this homework, you will learn:
  - How to use assignments with `wire`
  - How to use always blocks with `reg`
  - How to implement combinational and sequential circuits
  - How to design finite state machines for state control
  - How to develop a good verilog coding style

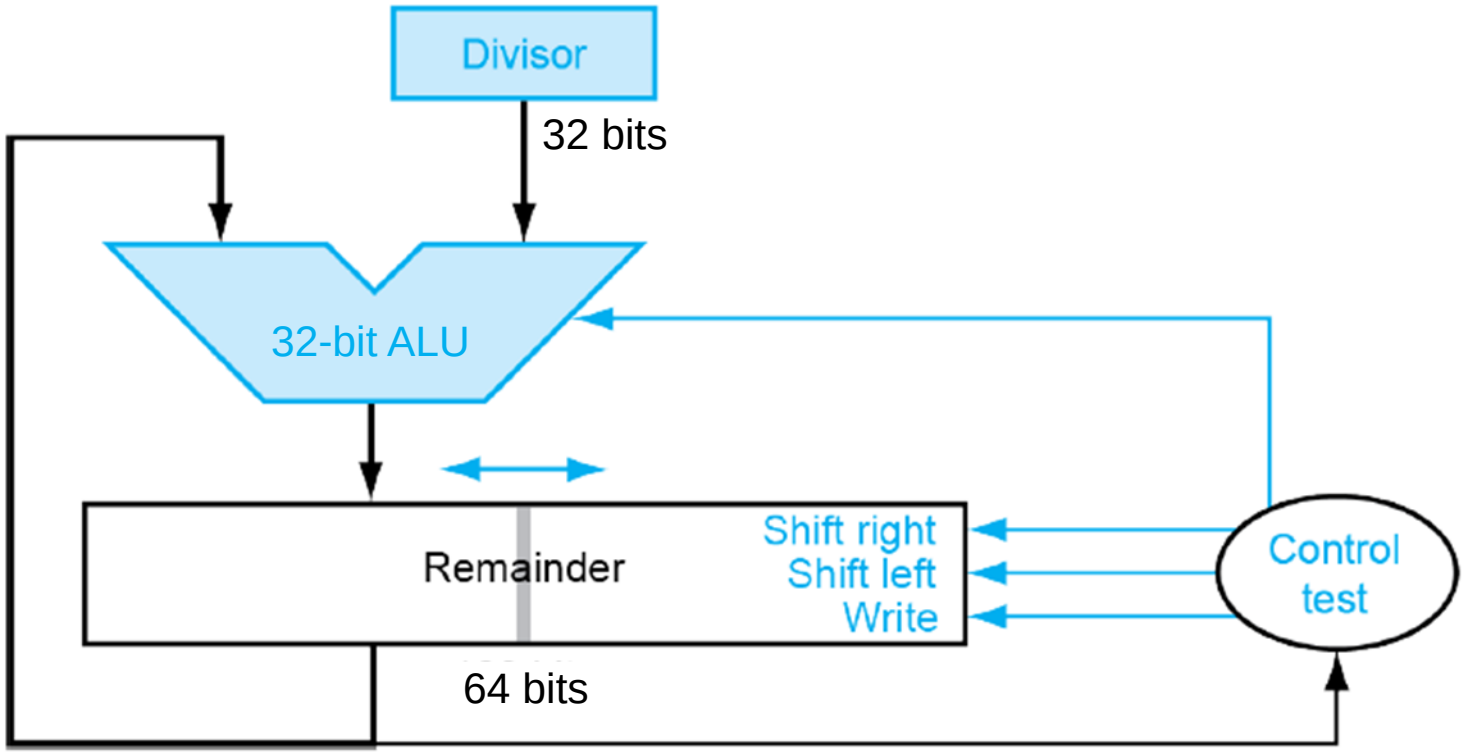
Quick Review: Multiplication Circuit

- Reduced register consumption
- Reduced size of ALU



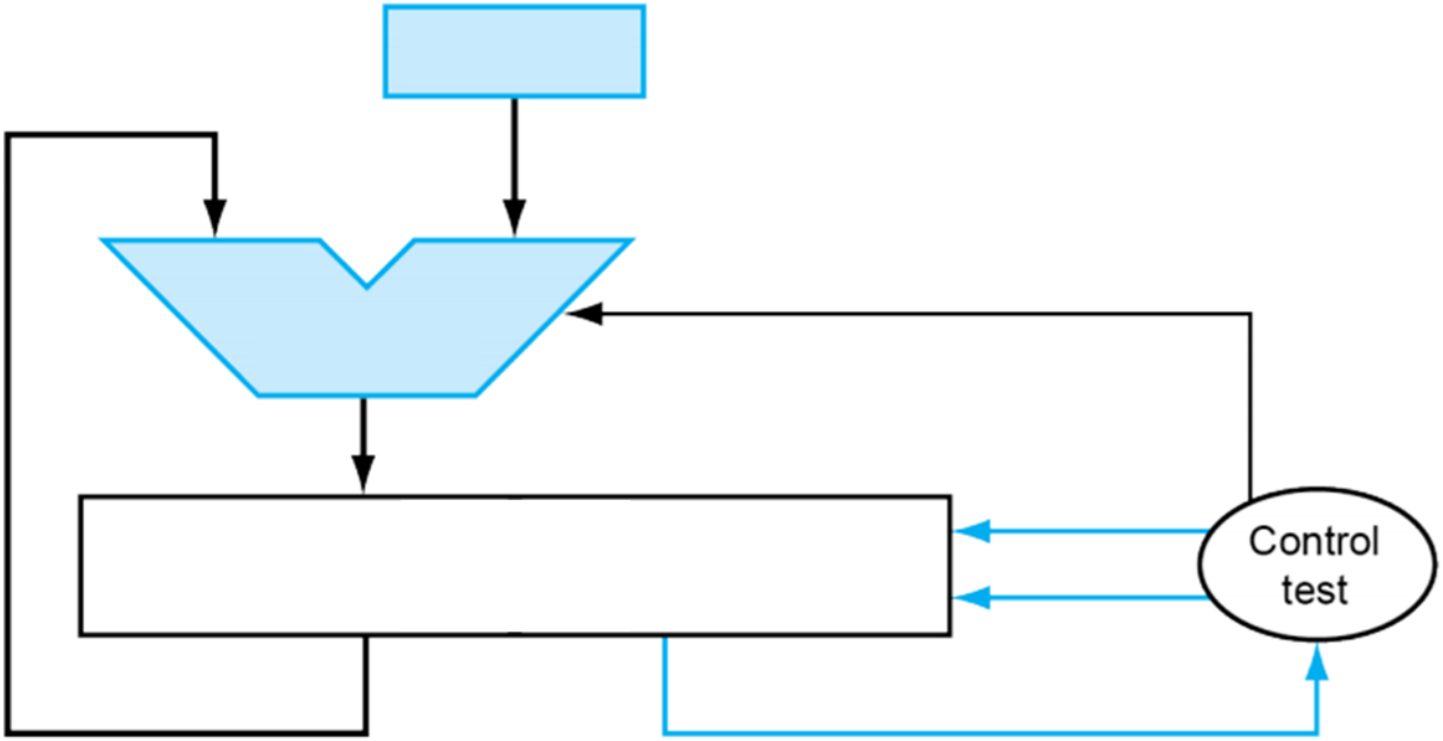
Quick Review: Division Circuit

- Reduced register consumption
- Reduced size of ALU

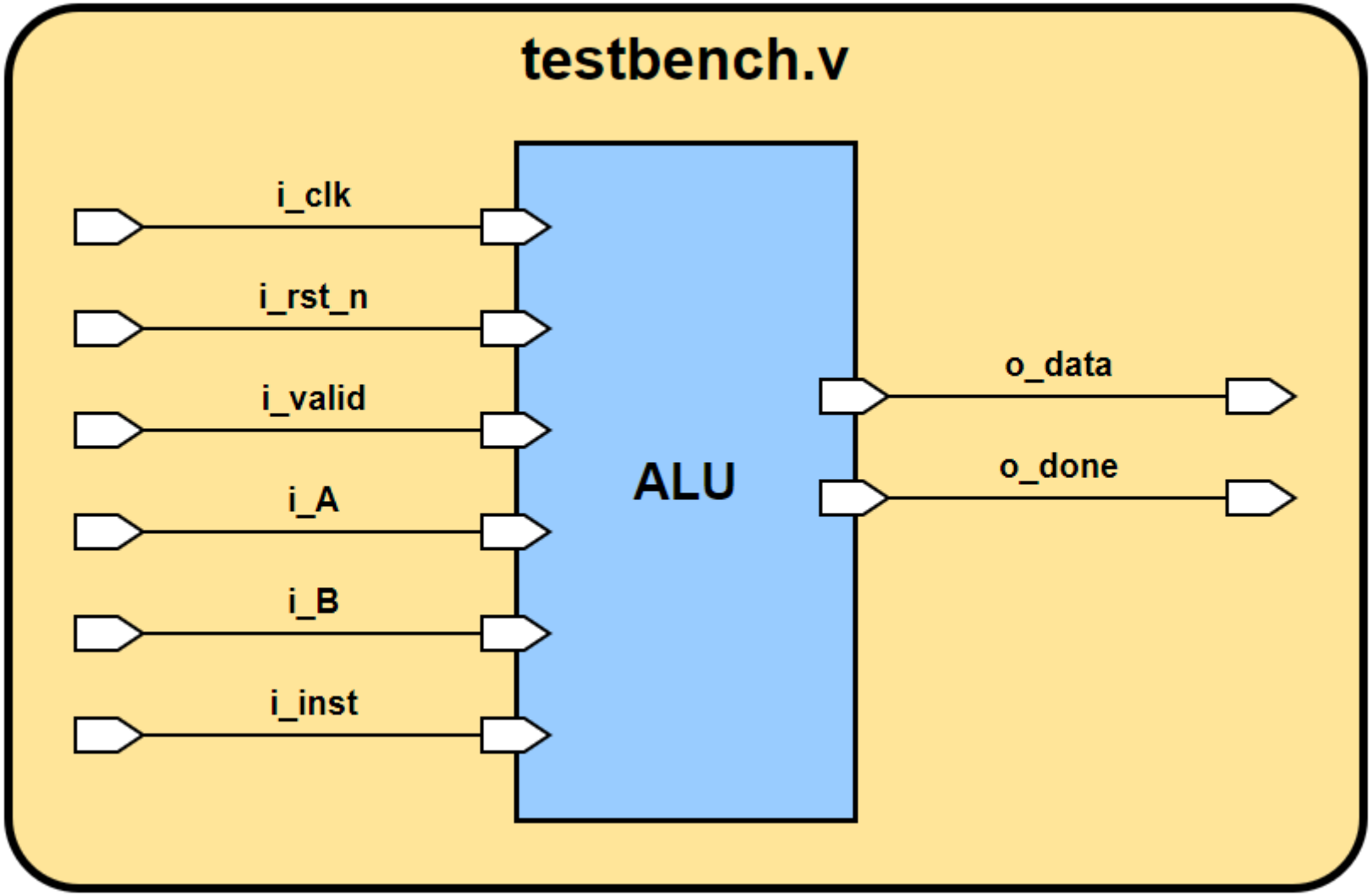


Observation

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



Block Diagram



Specification – I/O

Signal Name	I/O	Width	Description
i_clk	I	1	Clock signal
i_rst_n	I	1	Active **low** asynchronous reset
i_valid	I	1	Valid input data when valid = 1
i_A	I	32	32-bit input data A
i_B	I	32	32-bit input data B
i_inst	I	3	Operation instruction
o_data	O	64	64-bit output data
o_done	O	1	Set **high** with valid output data

Specification – I/O (cont.)

- **Do not Modify the I/O parts !!!**

```
module ALU #(
    parameter DATA_W = 32
)
(
    input                       i_clk,
    input                       i_rst_n,

    input                       i_valid,
    input [DATA_W - 1 : 0]      i_A,
    input [DATA_W - 1 : 0]      i_B,
    input [         2 : 0]      i_inst,

    output [2*DATA_W - 1 : 0]   o_data,
    output                      o_done
);
// Do not Modify the above part !!!
```

Specification – Other Description

- All inputs are synchronously fed within negative edge clock.
- All outputs should be synchronized at clock rising edge.
- All outputs should be reset when i_rst_n is **low**. Active low asynchronous reset is used only once.
- Instructions are given by i_inst when i_valid is **high**.
- i_valid stays high for only 1 cycle.
- The runtime of the design should be within 10000 cycles
- **The operators “ * ” and “ / “ are forbidden**

Design Description

- The followings are the operation modes you need to design for this homework:

Operations	i_inst	Meaning	Note
ADD	3'd0	A + B	Signed operation
SUB	3'd1	A – B	Signed operation
AND	3'd2	A & B	unsigned operation
OR	3'd3	A | B	unsigned operation
SLT	3'd4	If A < B then output 1; else output 0.	signed operation
SRA	3'd5	Shift A with B bits right	signed operation
MUL	3'd6	Multicycle A * B	unsigned operation
DIV	3'd7	Multicycle A / B	unsigned operation

Design Description (cont.)

- Overflow issue for **ADD**, **SUB**
  - Set o_data = $\begin{cases} 2^{31}-1, & if\ the\ result > 2^{31}-1 \\ -2^{31}, & if\ the\ result < -2^{31} \end{cases}$
- **MUL** and **DIV** operation
  - Multiplication
    - Unsigned input (32b)
    - out = i_A × i_B (32b × 32b → 64b)
  - Division
    - Unsigned input (32b)
    - quotient = i_A / i_B (quotient is 32b)
    - remainder = i_A % i_B (remainder is 32b)
    - out = {remainder, quotient}

Design Issues

- **Require 1 cycle to load data**
- Require 32 cycles to do MUL/DIV
- Require 1 cycle to do the rest operations
- 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 operations after 1 cycle

Load Data

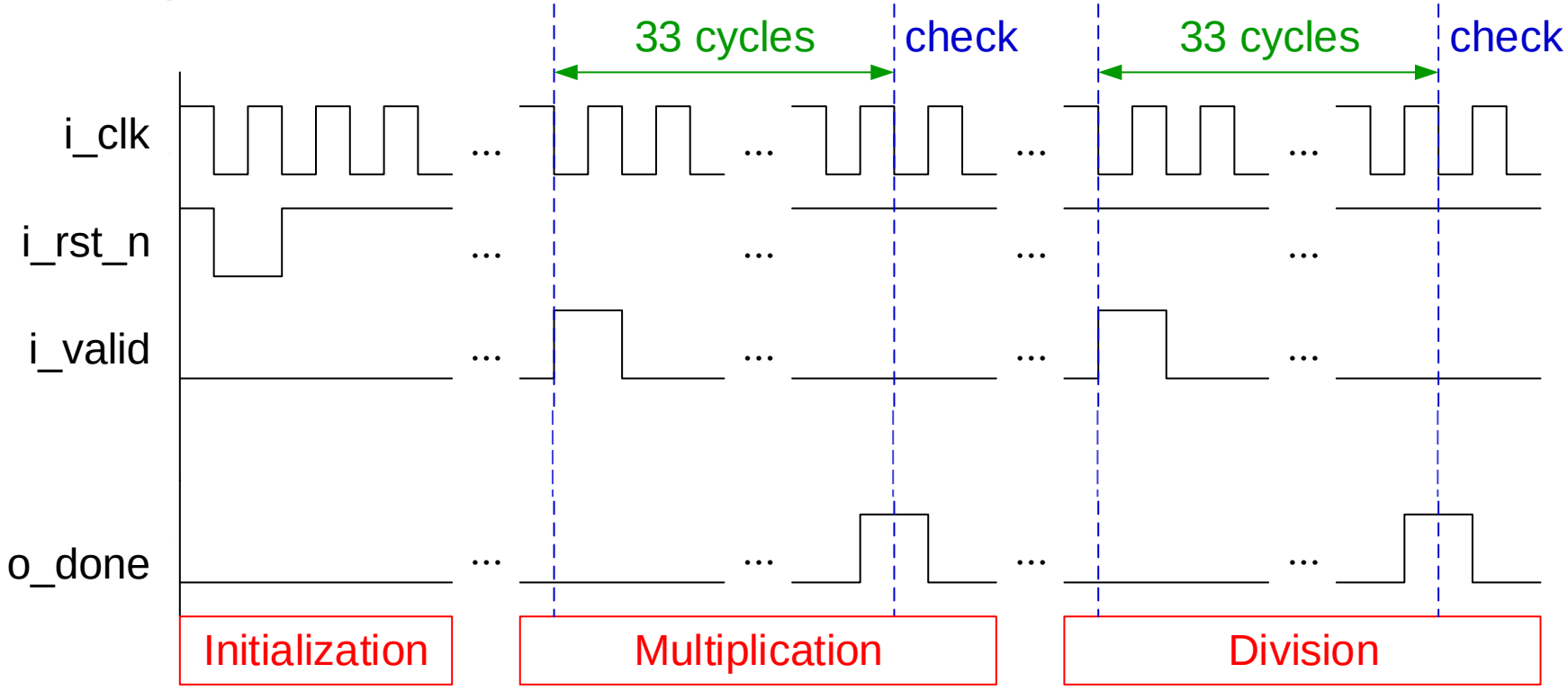
- Already implemented “Load Data” for you
- Please directly use the loaded data for computation

```
// load input
always @(*) begin
    if (i_valid) begin
        operand_a_nxt = i_A;
        operand_b_nxt = i_B;
        inst_nxt      = i_inst;
    end
    else begin
        operand_a_nxt = operand_a;
        operand_b_nxt = operand_b;
        inst_nxt      = inst;
    end
end
```

```
// Todo: Sequential always block
always @(posedge i_clk or negedge i_rst_n) begin
    if (!i_rst_n) begin
        state     <= S_IDLE;
        operand_a   <= 0;
        operand_b   <= 0;
        inst        <= 0;
    end
    else begin
        state     <= state_nxt;
        operand_a   <= operand_a_nxt;
        operand_b   <= operand_b_nxt;
        inst        <= inst_nxt;
    end
end
```

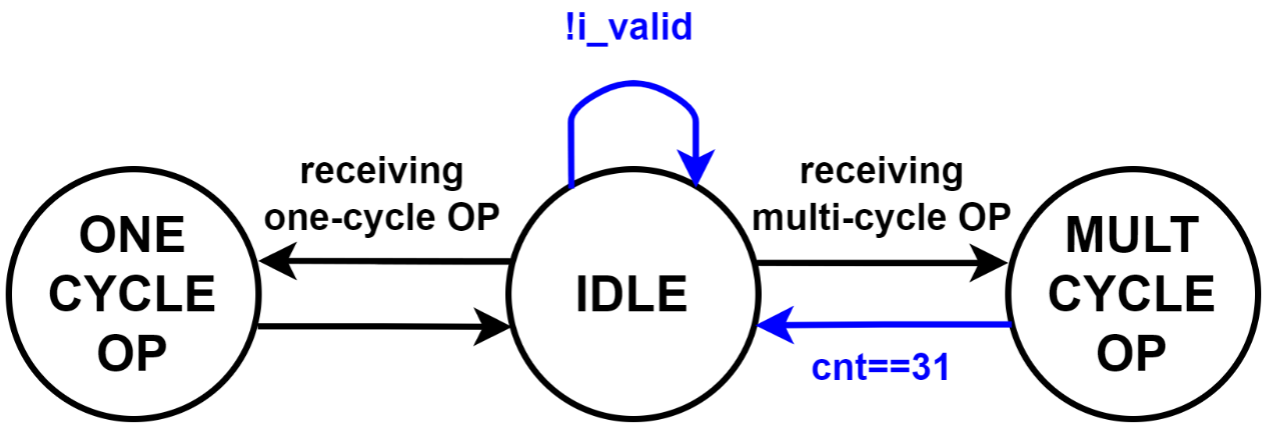
Waveform

- The testbench checks the result at the negedge of clock if o_done is **high**
- The testbench checks if the cycles used by each operation is correct



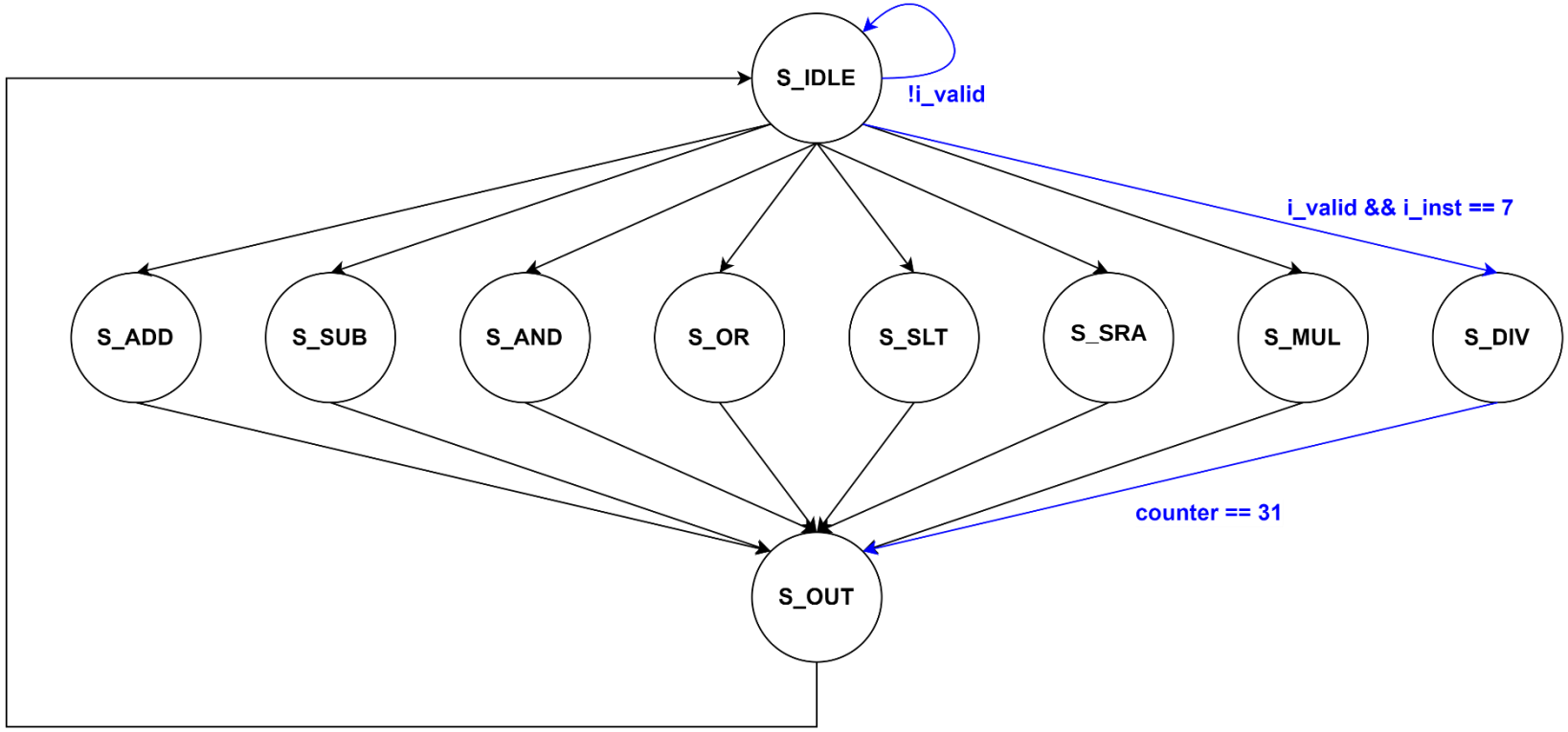
Finite State Machine (FSM)

- Categorizing operation modes based on operation cycles



Finite State Machine (FSM)

- Independent state for each operation mode



- You can design your own FSM

Todo: FSM

- Define your finite state machine

```
// 1. FSM based on operation cycles
parameter S_IDLE           = 4'd0;
parameter S_ONE_CYCLE_OP   = 4'd1;
parameter S_MULTI_CYCLE_OP = 4'd2;
```

```
// 2. FSM based on operation modes
// parameter S_IDLE = 4'd0;
// parameter S_ADD  = 4'd1;
// parameter S_SUB  = 4'd2;
// parameter S_AND  = 4'd3;
// parameter S_OR   = 4'd4;
// parameter S_SLT  = 4'd5;
// parameter S_SRA  = 4'd6;
// parameter S_MUL  = 4'd7;
// parameter S_DIV  = 4'd8;
// parameter S_OUT  = 4'd9;
```

- Sequential and Combinational Blocks

```
// Todo: Sequential always block
always @(posedge i_clk or negedge i_rst_n) begin
    if (!i_rst_n) begin
        state <= S_IDLE;
    end
    else begin
        state <= next_state;
    end
end
```

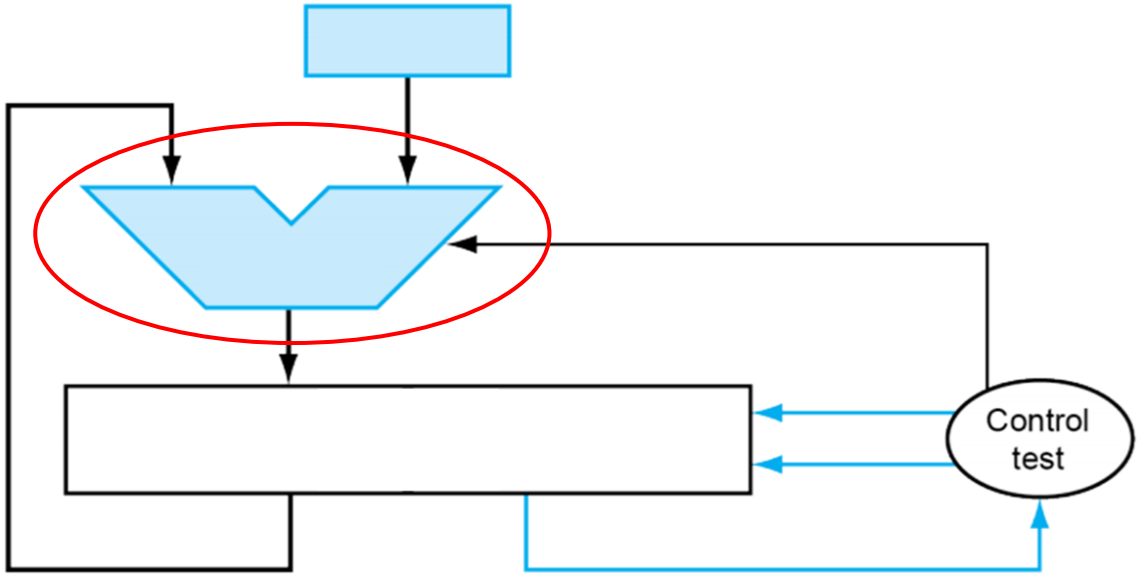
```
// Todo: FSM
    always @(*) begin
        case(state)
            S_IDLE           :
            S_ONE_CYCLE_OP   :
            S_MULTI_CYCLE_OP :
            default : state_nxt = state;
        endcase
    end
```

Todo: Counter

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

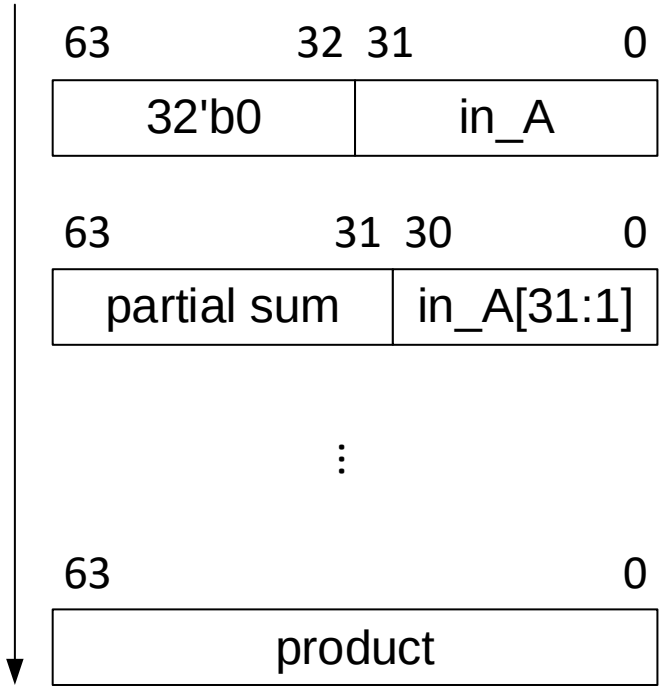
Todo: ALU output

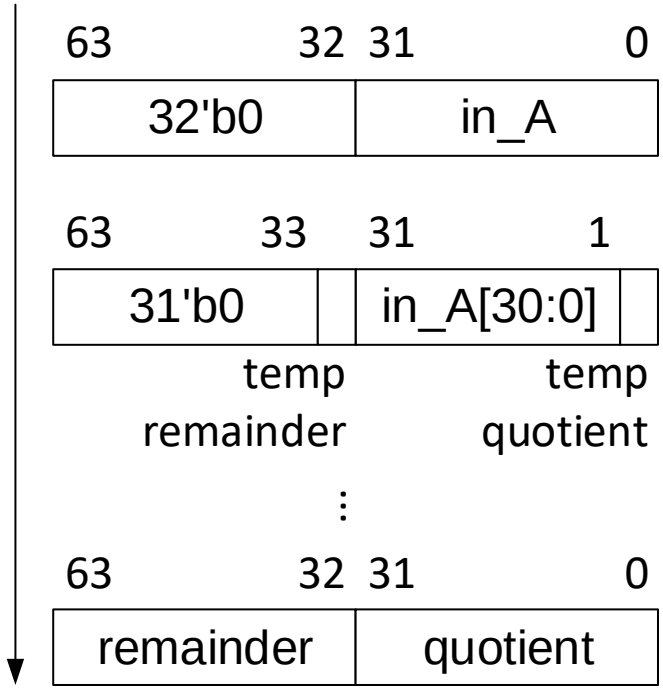
- Multiplication: addition / Division: subtraction
- Other operations
- You should consider
  - Bit width of ALU (to handle overflow)
  - Which bit range of shift register should be transmitted to ALU
  - The operation in ALU given current state



Todo: Shift Register

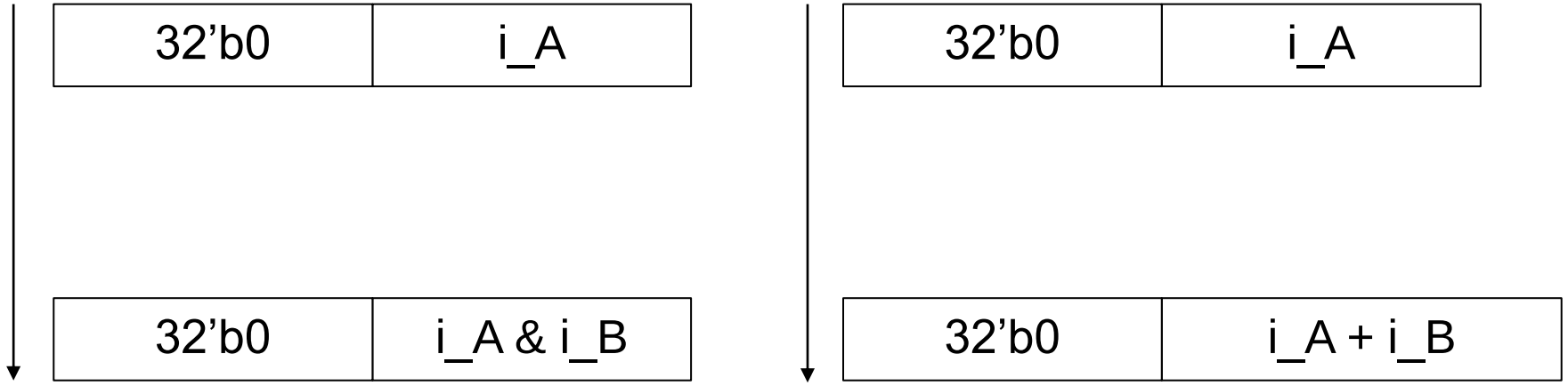
- Load data to the low 32 bits when the state is IDLE and valid = 1
- Multiplication: update data and right shift
- Division: update data and left shift





Todo: Shift Register (Cont.)

- Load data to the low 32 bits when the state is IDLE and valid = 1
- Other operations: (ex.)
  - AND: update data
  - ADD: update data



Todo: 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

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 00_license.f` (use given file)

Simulation (cont.)

- Verilog simulation
  - `$ source 01_run.f`
  - TA will run your code with following format of command in 01_run.f :

```
vcs ../00_TB/HW2_tb.v HW2.v -full64 -R -\
debug_access+all +v2k +notimingcheck +define+I0
```

```
vcs ../00_TB/HW2_tb.v HW2.v -full64 -R -\
debug_access+all +v2k +notimingcheck +define+I1
```

⋮

  - The word in the block "I0" is the instruction set of test pattern, please change it from I0 to I7 to test your design.
  - Make sure to pass every given testcase without any error messages.

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
- **Make sure there is no latches in your design**
- Check by Design Compiler
- Command:
  - `$ dv -no_gui`
  - `design_vision> read_verilog HW2.v`
- Exit:
  - `design_vision> exit`

Submission

- **Deadline: 23.59 pm on Nov. 20 (Mon.), 2023**
  - No late submissions are allowed
- Upload HW2_<student_id>_vk.zip to NTUCOOL
  - (k is the number of version, k =1,2,…)
  - Use lower case for the letter in your student ID. (Ex. HW2_b08901061_v1.zip)
  - Wrong file/folder format or additional files would result in **5-point** penalty each
  - HW2_<student_id>_vk.zip
    - HW2_<student_id>/
      - HW2_<student_id>/HW2.v
      - HW2_<student_id>/report.pdf
  - TA will only check the last version of your homework.

Grading Policy

- TA will check if your design is reasonable
  - Does NOT exist initial block and delay declaration
  - **Does NOT exist operators including " * " and " / "**
  - **Does NOT exist latches**
  - **Functionality would only be checked if the demands above are satisfied**
- Pass the patterns to get full score
  - Provided pattern: **80%** (I0 ~ I7)
    - **10%** for each set
    - **Don't implement your design directly based on the given testcases!**
  - Hidden pattern: **20%**
- Other rules:
  - Lose **5-point** for any wrong naming rule. Don't compress all homework folder.