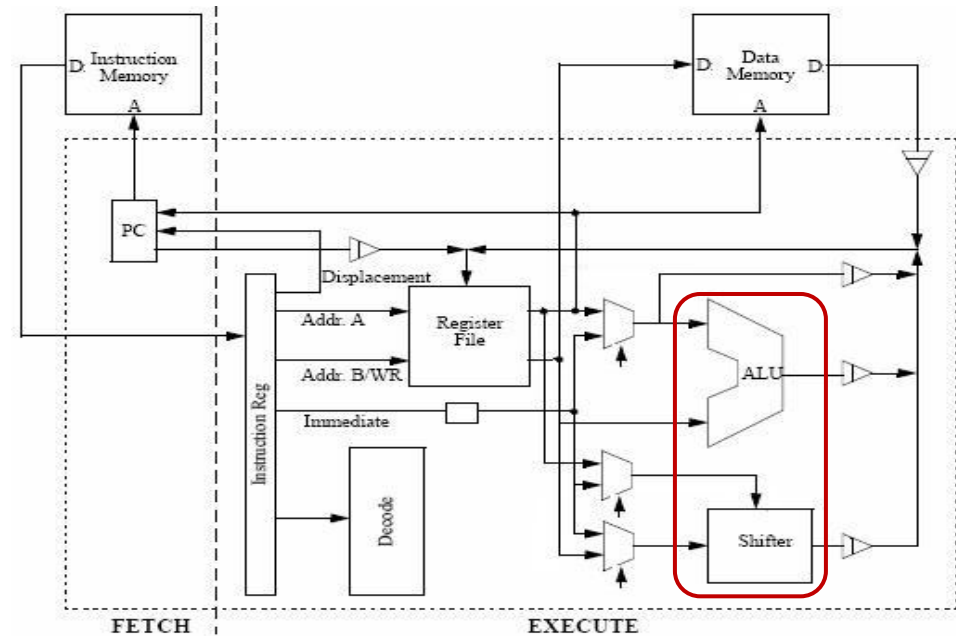
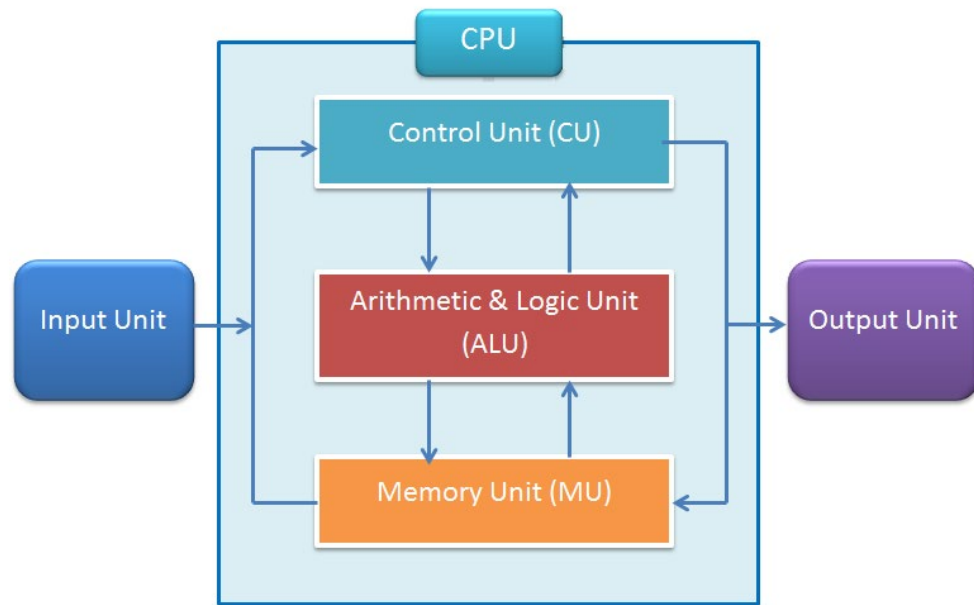


# Lab 2: Design Execution Part

~~10/3 — 10/11 (11:59:59 pm)~~

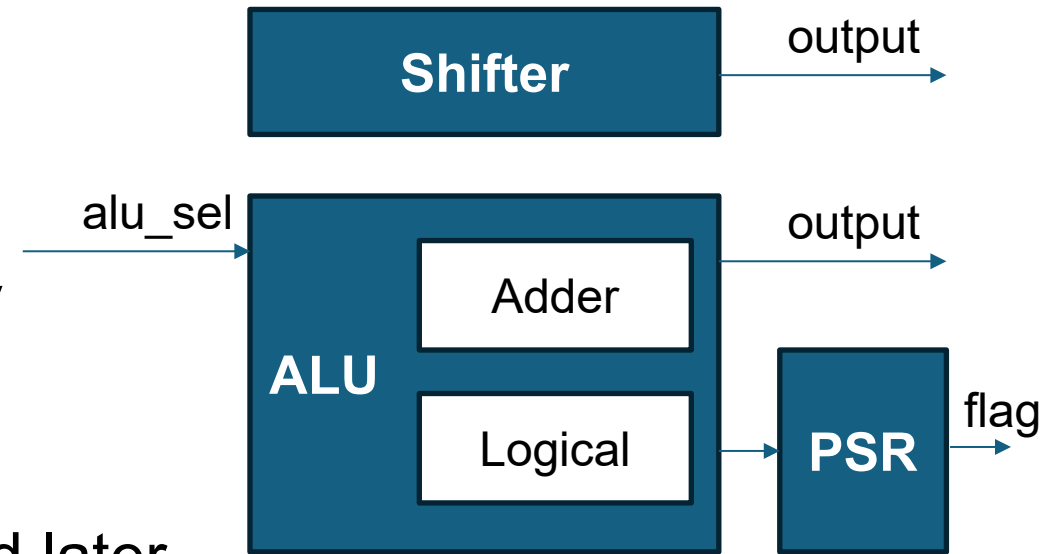
# Execution Part

- Use ALU or shifter to perform operations on data
- Might output execution flags used for branch/jump



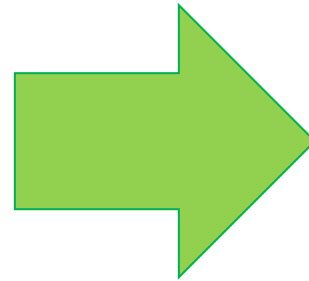
# Execution Consists of...

- ALU includes followings
  - Adder: for ADD, SUB, CMP
  - Logical operators: AND/OR/XOR
  - ALU modes are defined in ALU\_ops.v
- Next to ALU...
  - Simple bidirectional shifter
  - PSR: status register of ALU, explained later
- We implement and integrate components



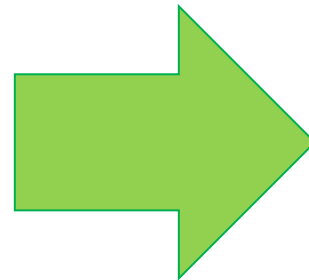
# ALU's Operation Modes in `ALU_ops.v`

- `alu_sel : 6'b100000`
  - `ADD`
- `alu_sel : 6'b010000`
  - `SUB`
- `alu_sel : 6'b001000`
  - `CMP`



**Result as CLA's output**

- `alu_sel : 6'b000100`
  - `AND`
- `alu_sel : 6'b000010`
  - `OR`
- `alu_sel : 6'b000001`
  - `XOR`



**Result as logic's output**

# Adder: Carry Lookahead Adder (CLA)

- Assume: adder for  $A_n \dots A_0$  and  $B_n \dots B_0$
- Able to get result only using input and input carry in without knowing carryout at each bit position
- Defined new signals “propagate” and “generate”

$$p_i = A_i + B_i$$

$$g_i = A_i \cdot B_i$$

- Reform carry-out at each bit position

$$c_{i+1} = A_i \cdot B_i + A_i \cdot c_i + B_i \cdot c_i \Rightarrow c_{i+1} = g_i + p_i \cdot c_i$$

# A 4b CLA

$$c_1 = g_0 + p_0 \cdot c_0$$

$$c_2 = g_1 + p_1 \cdot c_1$$

$$= g_1 + p_1 \cdot (g_0 + p_0 \cdot c_0)$$

$$= g_1 + p_1 \cdot g_0 + p_1 \cdot p_0 \cdot c_0$$

$$c_3 = g_2 + p_2 \cdot c_2$$

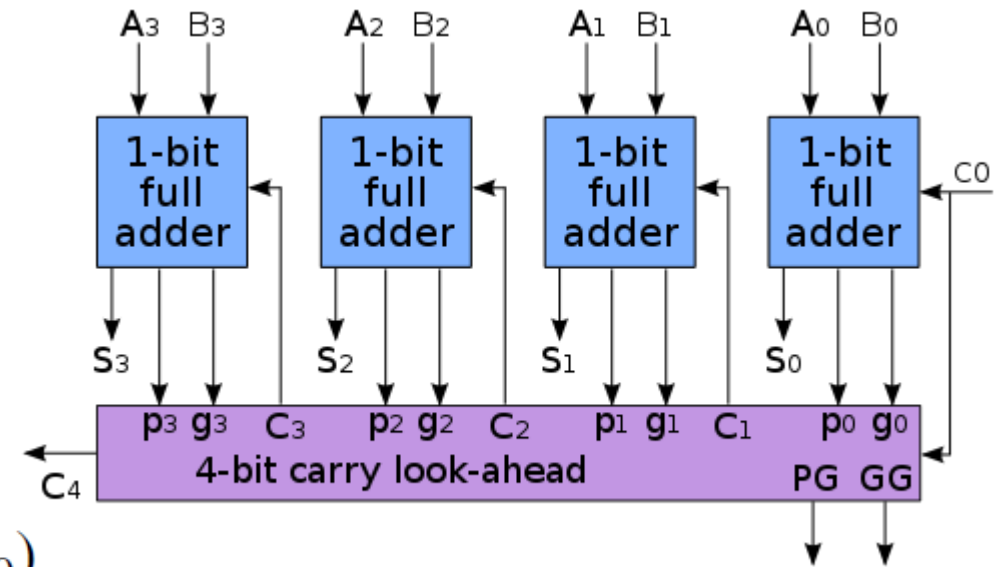
$$= g_2 + p_2 \cdot (g_1 + p_1 \cdot g_0 + p_1 \cdot p_0 \cdot c_0)$$

$$= g_2 + p_2 \cdot g_1 + p_2 \cdot p_1 \cdot g_0 + p_2 \cdot p_1 \cdot p_0 \cdot c_0$$

$$c_4 = g_3 + p_3 \cdot c_3$$

$$= g_3 + p_3 \cdot (g_2 + p_2 \cdot g_1 + p_2 \cdot p_1 \cdot g_0 + p_2 \cdot p_1 \cdot p_0 \cdot c_0)$$

$$= g_3 + p_3 \cdot g_2 + p_3 \cdot p_2 \cdot g_1 + p_3 \cdot p_2 \cdot p_1 \cdot g_0 + p_3 \cdot p_2 \cdot p_1 \cdot p_0 \cdot c_0$$



# Extending 4b CLA as 16b Group CLA

- Components
  - Four groups of 4b CLAs
  - One lookahead carry unit (LCU)
- LCU generates carry-in to each group
- Notes
  - Provided code is slightly different from figure
  - CLA does not know input is signed or unsigned

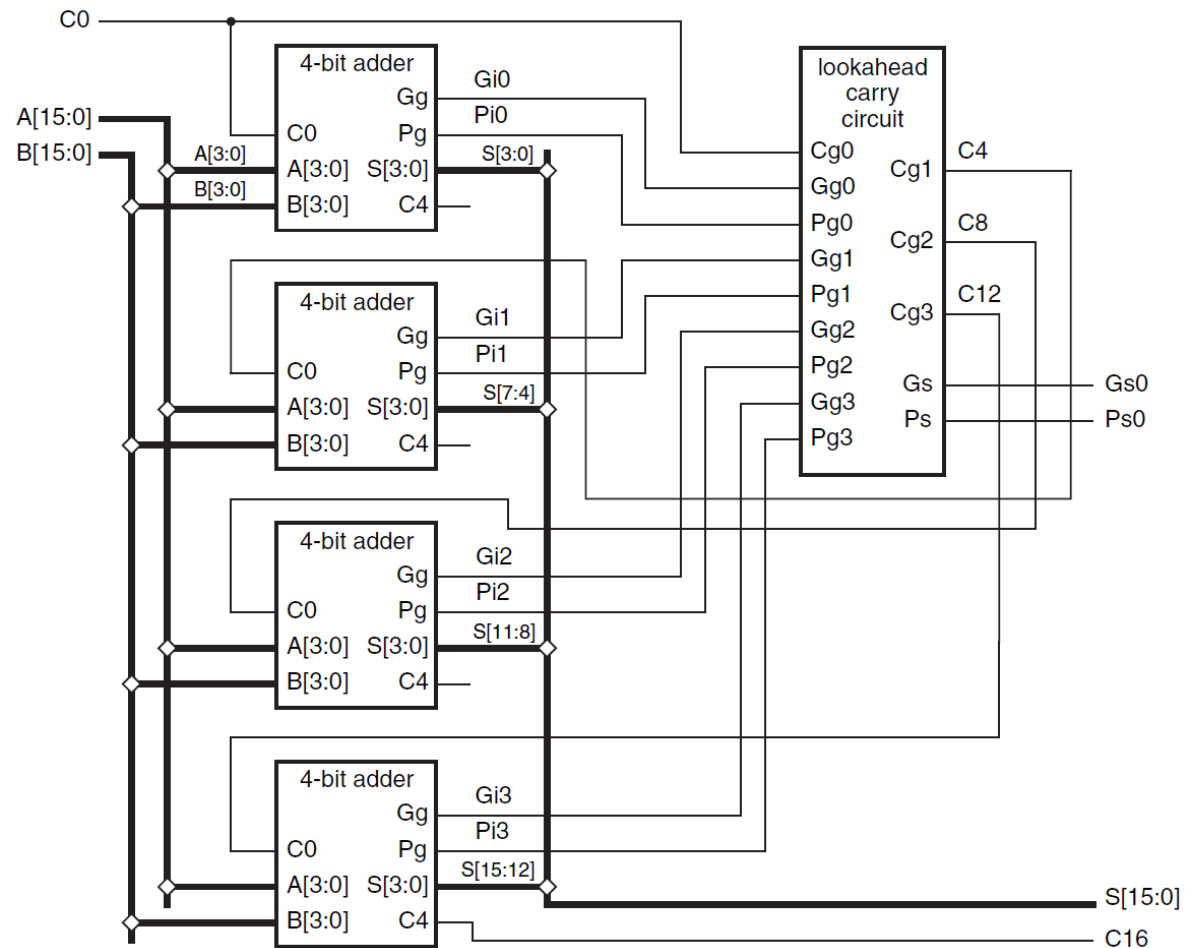


Figure 8-8 A 16-bit group-carry-lookahead adder with 4-bit groups.

# Signals of 16b CLAI in Our Code

- Input
  - $A_i$ : first 16b operand
  - $B_i$ : second 16b operand
  - $CARRYIN_i$ : Used to select operation mode.
    - 0:  $A_i + B_i$
    - 1:  $B_i - A_i$
- Output
  - $sum_o$ : calculation result of CLA in 16b
  - **flag\_overflow\_o**: overflow flags. One flag will be used at a time by the *instruction decoder* depending on the operation mode (or *opcode*)
    - [0] (C-flag): overflow that is meaningful for unsigned operations
    - [1] (F-flag): overflow that is meaningful for signed operations (2s complement)



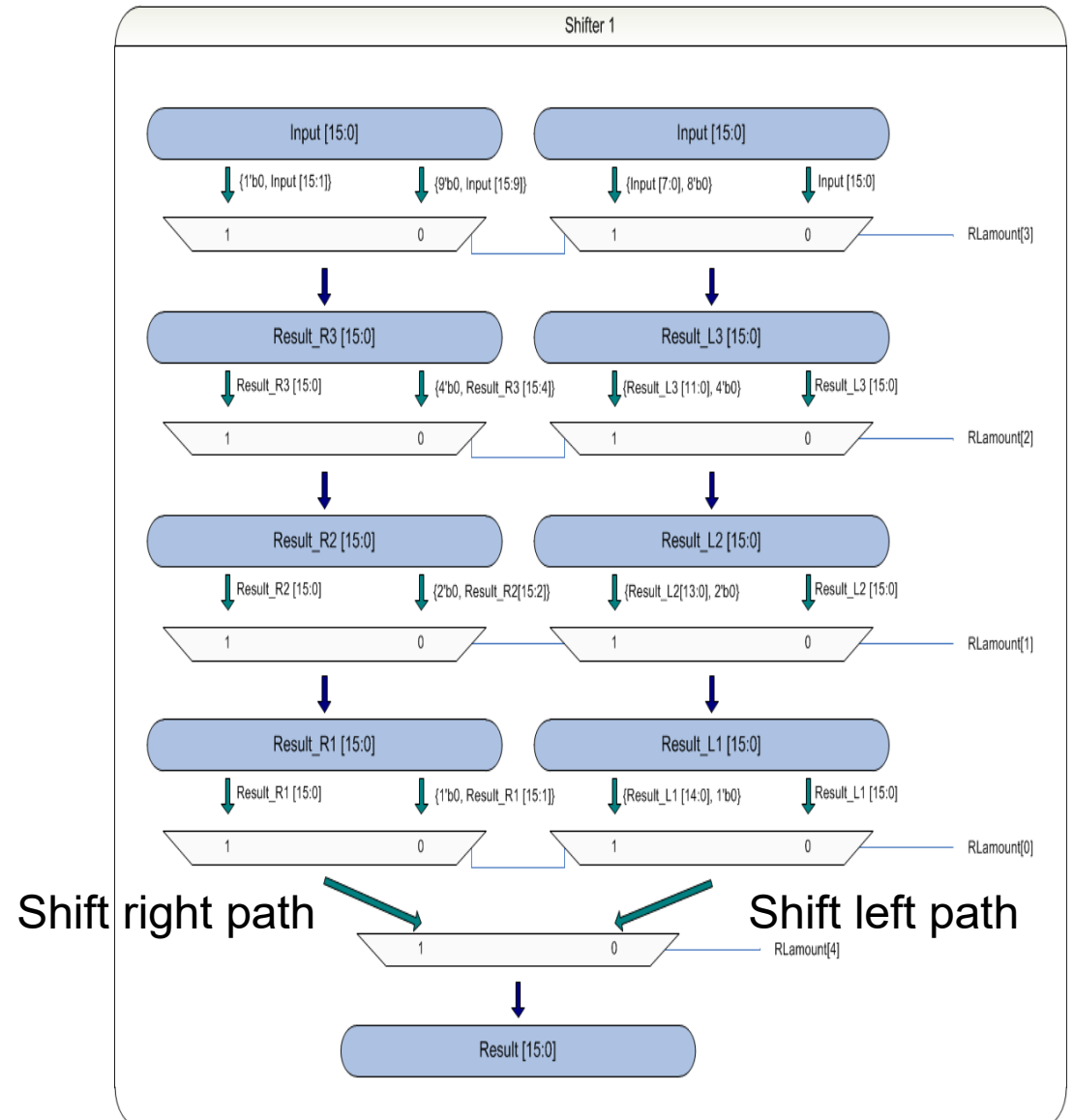
# TODO: Implement Overflow Flags

- Case studies for understanding (0x: Hexadecimal)

A [unsigned] [signed]	B [unsigned] [signed]	CLA16b.ADD (carryout, 16b)	CLA16b.SUB (carryout, 16b)	Unsigned		Signed		F		C	
				True.ADD	True.SUB	True.ADD	True.SUB	ADD	SUB	ADD	SUB
0xDEAD [57005] [-8531]	0xCAFE [51966] [-13570]	1, 0xA9AB	0, 0xEC51	0x1_A9AB (>65535)	-5039 (not unsigned!)	0xA9AB	0xEC51	0	0	1	1
0x10AF [4271] [4271]	0xBEEF [48879] [-16657]	0, 0xCF9E	1, 0xAE40	0xCF9E	0xAE40	0xCF9E	0xAE40	0	0	0	0
0xBAAD [47789] [-17747]	0xC0DE [49374] [-16162]	1, 0x7B8B	1, 0x0631	0x1_7B8B (>65535)	0x0631	-33909 (0x7B8B>0)	0x0631	1	0	1	0
0x0003	0x0004	0, 0x0007	1, 0x0001	0x0007	0x0001	0x0007	0x0001	0	0	0	0
0x4B1D [19229] [19229]	0x10AF [4271] [4271]	0, 0x5BCC	0, 0xC592	0x5BCC	-14958 (not unsigned!)	0x5BCC	0xC592	0	0	0	1
0x5440 [21568] [21568]	0xABBA [43962] [-21574]	0, 0xFFFFA	1, 0x577A	0xFFFFA	0x577A	0xFFFFA	-43142 (0x577A>0)	0	1	0	0

# Simple Barrel Shifter

- A 4-stage cascaded combinational logic
- Able to shift for any amounts towards bidirectionally



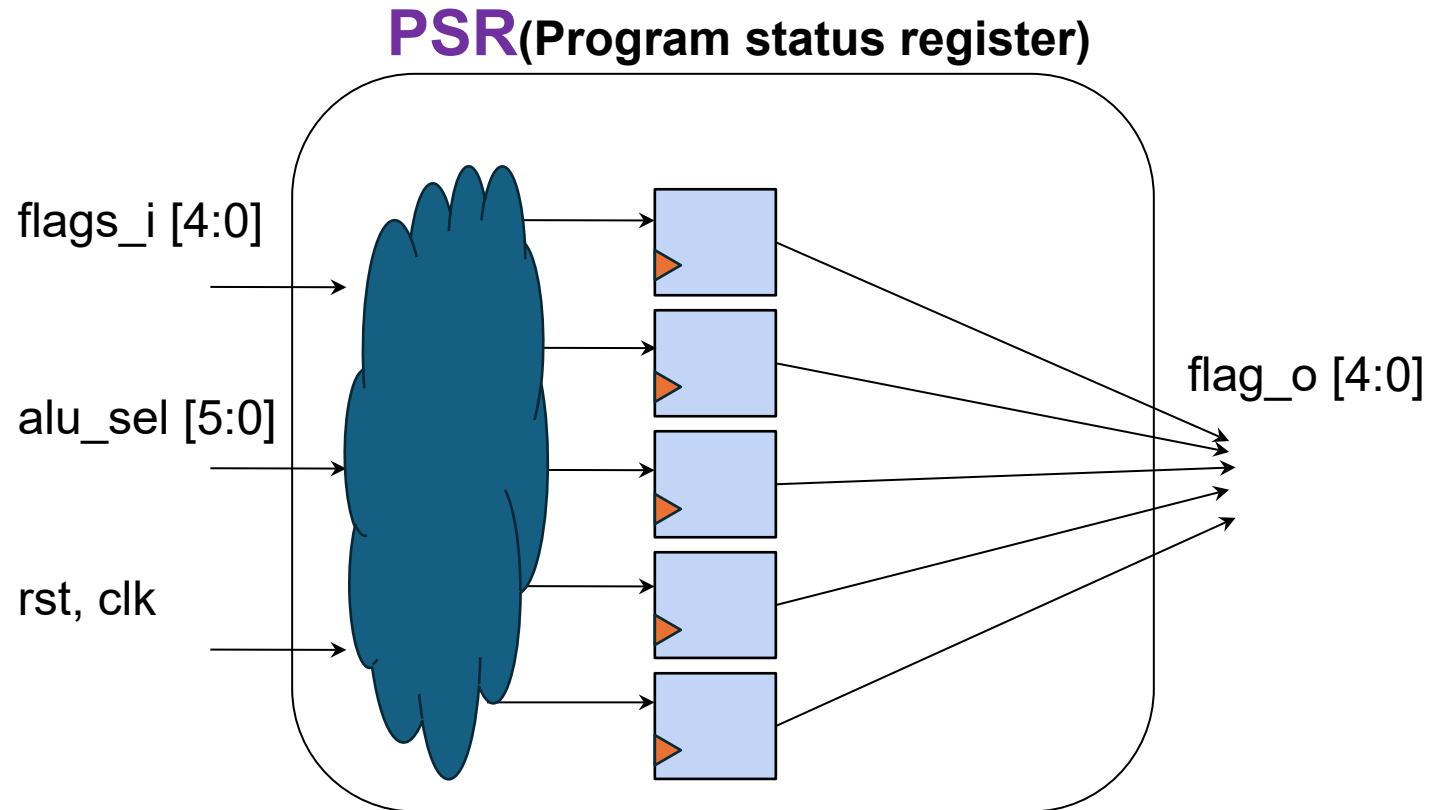
# TODO: Implement Shifter

- rl\_shift\_amt:
  - 2s complement signed number
  - Determines shift direction & amount
  - ***If it is negative, shift right***
- lui: treat rl\_shift\_amt as ***don't care*** and simply shift left for 8 bits

```
1 module shifter
2 (
3     input [15:0] data_i,
4     input [4:0]  rl_shift_amt_i,
5     input        lui_i,
6
7     output [15:0] data_o
8 );
9
```

# Program Status Register (PSR)

- Output program status (e.g., overflow) based ALU's mode



# TODO: Propagate Register Inputs Properly

- Flags\_in is fed from ALU's flags\_o
- Alu\_sel is fed with ALU's selection in parallel
- PSR's flags\_o (FLCNZ) conditions
  - If alu\_sel == ADD or SUB
    - Propagate ALU's flags\_o[4] and [2] only
  - If alu\_sel == CMP
    - Propagate ALU's flags\_o[3], [1], [0]
  - Else → maintain the all registers' states

```
if clk.posedge {  
    if reset {  
        reset registers  
    } else if alu_sel = add or sub {  
        reg(F,C) = flag_i(F,C)  
    } else if alu_sel = cmp {  
        reg(L,N,G) = flag_i(L,N,Z)  
    }  
}  
flags_o = reg{F,L,C,N,Z}
```

# FLCNZ?

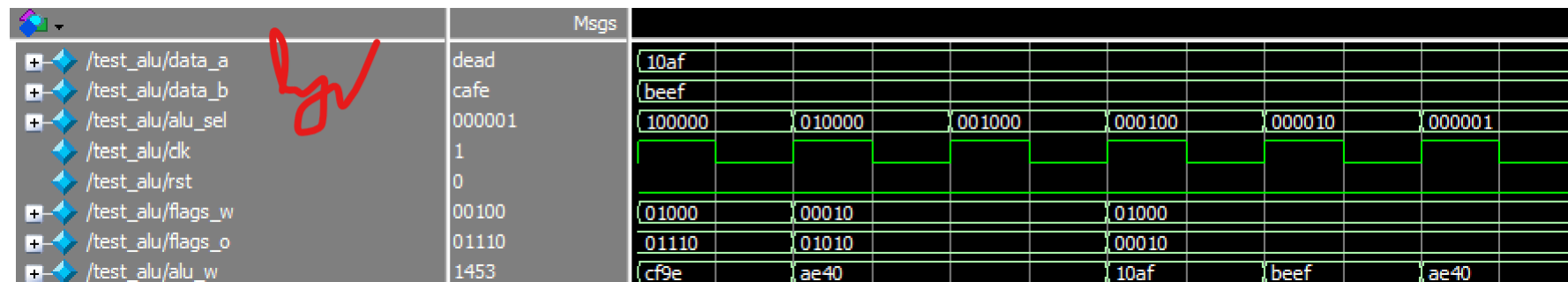
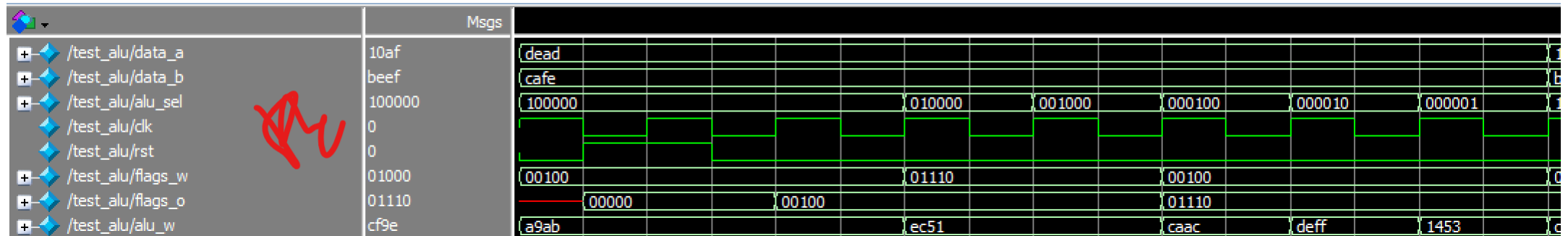
- Flags will be finally used by instruction decoder in the later lab
- Need to selectively propagate correct flags according to operation types
  - Signed? Unsigned?
  - [4:0] → FLCNZ in order
- Flags descriptions
  - C flag : overflow / underflow @ unsigned ADD/SUB
  - F flag : overflow / underflow @ signed ADD/ SUB
  - Z flag : A==B @ CMP
  - L flag : B<A @ unsigned CMP
  - N flag : B<A @ signed CMP
- Signed ADD/SUB or signed CMP will be inferred by instruction decoder

# Assignment

- In your report...
  - Implement and explain overflow flags in CLA16b (cla16.v) – 20%
  - Implement and explain psr input signals in ALU (alu.v) – 20%
  - Implement and explain shifter (shifter.v) – 20%
  - Implement and explain PSR (psr.v) – 20%
  - Validate waveforms in next slides (test\_alu.v, test\_shifter.v) – 20%
    - Please attach your capture of waveforms
- Use of “+/-” is NOT allowed
- Use of “^/&/|” is ALLOWED









# Desired Waveforms of test\_alu









- flags\_w is output of combinational logic
- flags\_o is selectively registered value being used as final flags
- You only need to match **flags\_o** in the waveform














# Desired Waveforms of test\_alu

	Msgs	
 /test_alu/data_a	dead	(baad
 /test_alu/data_b	cafe	(c0de
 /test_alu/alu_sel	000001	(100000
 /test_alu/clk	1	010000
 /test_alu/rst	0	001000
 /test_alu/flags_w	00100	000100
 /test_alu/flags_o	01110	000010
 /test_alu/alu_w	1453	000001
		(7b8b
		0631
		808c
		faff
		7a73


	Msgs	
 /test_alu/data_a	dead	(0003
 /test_alu/data_b	cafe	(0004
 /test_alu/alu_sel	000001	(100000
 /test_alu/clk	1	010000
 /test_alu/rst	0	001000
 /test_alu/flags_w	00100	000100
 /test_alu/flags_o	01110	000010
 /test_alu/alu_w	1453	000001
		(01010
		000000
		01010
		000000
		0007
		0001
		0000
		0007

	Msgs	
 /test_alu/data_a	dead	(4b1d
 /test_alu/data_b	cafe	(10af
 /test_alu/alu_sel	000001	(100000
 /test_alu/clk	1	010000
 /test_alu/rst	0	001000
 /test_alu/flags_w	00100	000100
 /test_alu/flags_o	01110	000010
 /test_alu/alu_w	1453	000001
		(100000
		010000
		001000
		000100
		000010
		000001
		5440
		abba
		100000
		010000
		001000
		000100
		000010
		000001
		01000
		10010
		01000
		01010
		11010
		10010
		5bcc
		c592
		000d
		5bbf
		5bb2
		fffa
		577a
		0000
		fffa

# Waveform of test\_shifter



	Msgs	
/test_shifter/data_i	1100101011111110	1100101011111110
/test_shifter/rl_shift_amt	0	0
/test_shifter/loi	0	
/test_shifter/data_o	cafe	cafe 95fc 2bf8 57f0 afe0 5fc0 bf80 7f00 fe00 fc00 f800 f000 e000 c000 8000 0000



	Msgs	
/test_shifter/data_i	1100101011111110	1100101011111110
/test_shifter/rl_shift_amt	-2	-16 -15 -14 -13 -12 -11 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1
/test_shifter/loi	0	
/test_shifter/data_o	32bf	0000 0001 0003 0006 000c 0019 0032 0065 00ca 0195 032b 0657 0caf 195f 32bf 657f fe00