Logic Design

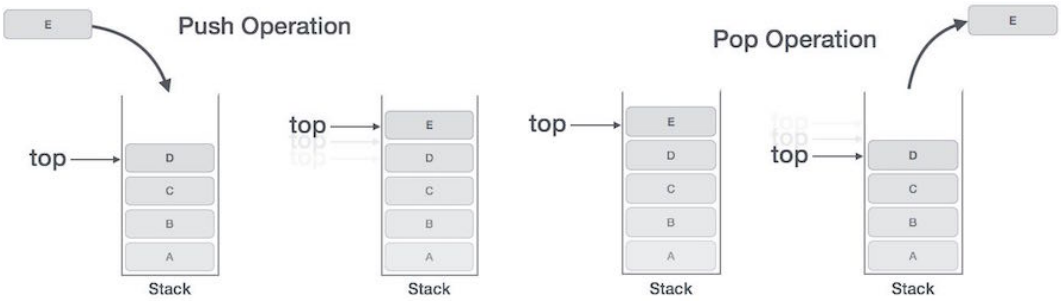
EECS101001

Lab 5: Stack Machine – Report

104021215 熊 磊

1. Foreword

This Lab is trying to design a Stack Machine, which is a **Last In First Out** structure. And also support basic operation: addition, subtraction and multiplication.



SM

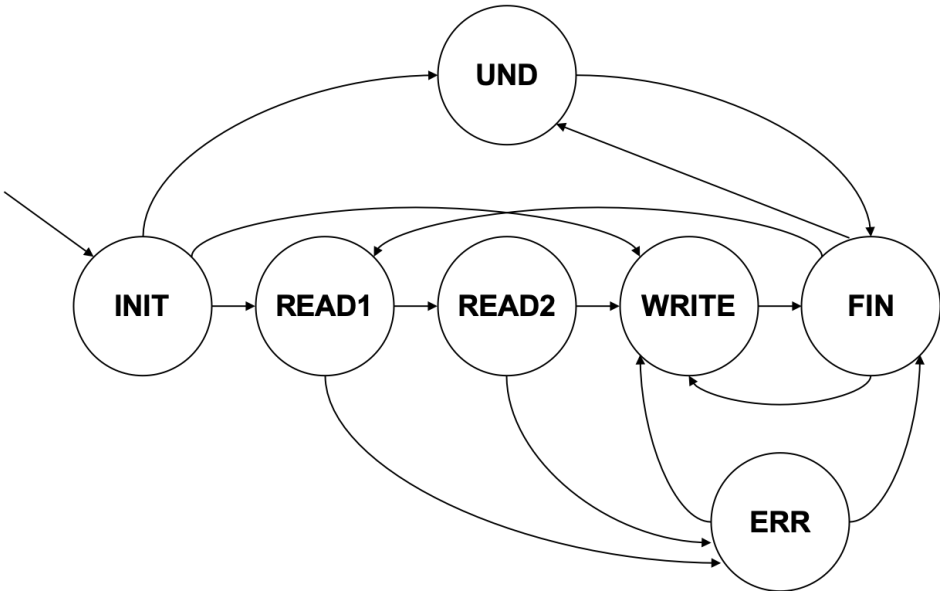
name	size	function	I/O
clk	1	時間訊號	input
rst_n	1	rst_n = 0 時進行 reset	input
instr	13	instruction	input
pc	10	instruction address，instruction 的總長度放在 pc = 1023 的位置	output
err_code	3	錯誤訊息	output
d_valid	1	當輸出的資料為 add, sub, mul 的最終結果時時把 d_valid 拉起	output
out_data	20	輸出的資料	output
fin	1	當處理完所有的 instruction 之後拉成 1	output

SM_Mem

name	size	function	I/O
r_data	20	從 stack memroy 讀出來的資料	input
full	1	當 stack memory 滿的時候拉為一	input
empty	1	當 stack memory 是空的時候拉為一	input
cntrl	2	控制 stack memory 的功能	output
w_data	20	要寫入 stack memory 的資料	output

2. Implementation Details and Designs

First, we draw a State Transition Graph as follow:



We need at least 7 states to handle different situations and operations. Hence, we define these states in 3-bits. Also, we define instruction's name.

```
`define PUSH 3'b000
`define ADD 3'b001
`define SUB 3'b010
`define MUL 3'b011
`define INIT 3'b000
`define READ1 3'b001
`define READ2 3'b010
`define WRITE 3'b011
`define FIN   3'b100
`define ERR   3'b101
`define UND   3'b110
```

State Name	Instruction
READ1	POP a number from stack.
READ2	POP a number from stack.
WRITE	Do operation and push result to stack, or just push a number to stack.
FIN	Get a new address (pc), and change to next state.
ERR	If it is an invalid operation (pop from empty stack or push to full stack), will be in **ERR** state. And if we need to restore number back to stack, then it turns to **WRITE** state.
UND	If it is an undefined instruction, then it turns to **FIN** state, waiting for next instruction.

Now we determine when to change states and change to which state. If it detects an error occurred, it will turn to corresponding state, **ERR (Error operation)** or **UND (Undefined Operation)**.

```
118     always@(*) begin
119         case(state)
120             `INIT: next_state1 = (oper == `PUSH) ? `WRITE : (oper == `ADD || oper == `SUB || oper == `MUL) ? `READ1 : `UND;
121             `READ1: next_state1 = (empty == 1) ? `ERR : `READ2;
122             `READ2: next_state1 = (empty == 1) ? `ERR : `WRITE;
123             `WRITE: next_state1 = `FIN;
124             `FIN: next_state1 = (oper == `PUSH) ? ((full == 0) ? `WRITE : `ERR) : ((oper == `ADD || oper == `SUB || oper == `MUL) ? `READ1 : `UND);
125             `ERR: next_state1 = (restore == 1) ? `WRITE : `FIN;
126             `UND: next_state1 = `FIN;
127             default: next_state1 = `INIT;
128         endcase
129     end
```

Also, we need some wires to do restore (if error occurred), error code, read data, write data, control operation, and signal from the Stack Memory determine whether it is full or empty. Here is a list for each instruction of wire.

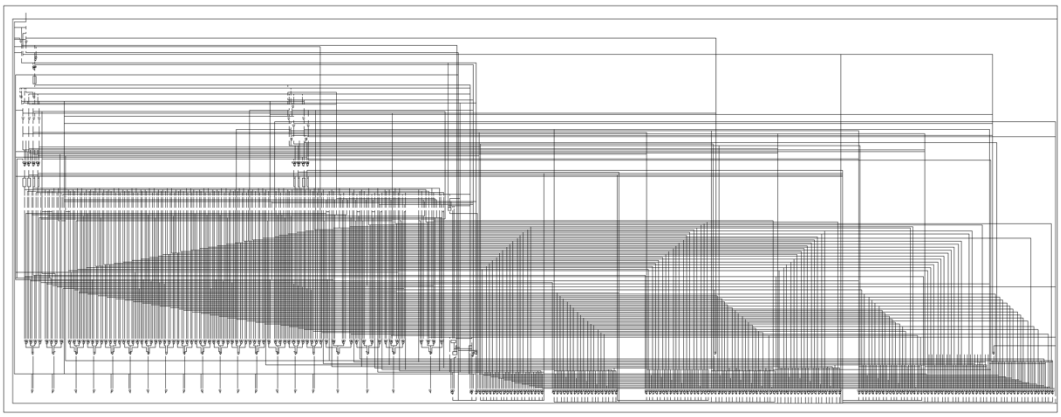
wire	function
state	Transmit the current state, initial will be **INIT**.
pc	Transmit the address to get the instruction from input, initial will be **1023**, to get the total numbers of instructions.
len	Store the numbers of instructions.
data	Store the first data pop from SM_Mem.
data2	Store the second data pop from SM_Mem.
restore	If it is an invalid operation, and we have to roll back to original situation.
cnt	cnt is a signal to transmit whether the writing data is the result of operation, so that d_valid can raise up.
r_data	r_data is a data poping from SM_Mem.
w_data	w_data is a data pushing to SM_Mem.
full	A signal to transmit if SM_Mem is full.
empty	A signal to transmit if SM_Mem is empty.

In **SM_Mem**, we use `num1`, `num2`, ⋯, `num8` to store each number in stack memory. And using `top` to indicate top element of stack.

```
156     assign next_top = (rst_n == 1'b0) ? 4'b0 :
157                       (cntrl == `DO_PUSH) ? ((top == `STK8) ? top : top + 1'b1) :
158                       (cntrl == `DO_POP) ? ((top == `STK0) ? top : top - 1'b1) : top;
159     assign full = (top == `STK8) ? 1'b1 : 1'b0;
160     assign empty = (top == `STK0) ? 1'b1 : 1'b0;
161     assign next_num1 = (top == `STK0 && cntrl == `DO_PUSH) ? w_data : num1;
162     assign next_num2 = (top == `STK1 && cntrl == `DO_PUSH) ? w_data : num2;
163     assign next_num3 = (top == `STK2 && cntrl == `DO_PUSH) ? w_data : num3;
164     assign next_num4 = (top == `STK3 && cntrl == `DO_PUSH) ? w_data : num4;
165     assign next_num5 = (top == `STK4 && cntrl == `DO_PUSH) ? w_data : num5;
166     assign next_num6 = (top == `STK5 && cntrl == `DO_PUSH) ? w_data : num6;
167     assign next_num7 = (top == `STK6 && cntrl == `DO_PUSH) ? w_data : num7;
168     assign next_num8 = (top == `STK7 && cntrl == `DO_PUSH) ? w_data : num8;
169     assign r_data = (cntrl == `DO_POP) ? ((top == `STK1) ? num1 : (top == `STK2) ? num2 : (top == `STK3) ? num3 : (top == `STK4) ? num4 :
170                     (top == `STK5) ? num5 : (top == `STK6) ? num6 : (top == `STK7) ? num7 : (top == `STK8) ? num8 : 20'b0):20'b0;
```

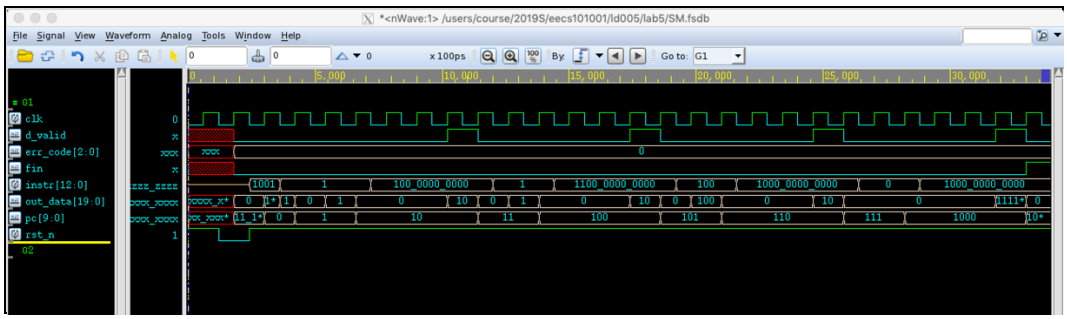
Finally, if we instantiate SM_Mem in SM, we can push number to or do operation from stack memory.

3. Block Diagram

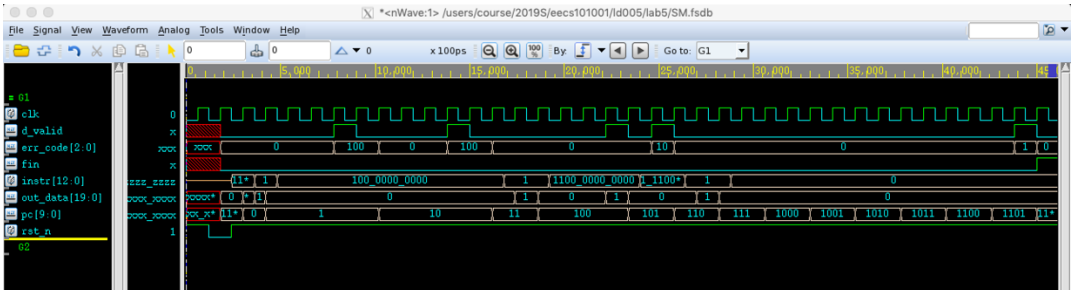


4. nWave Result

Without Error Detection



With Error Detection



5. Report Area

```
****************************************
Report : area
Design : SM
Version: K-2015.06-SP1
Date   : Sat Jun  8 05:15:57 2019
****************************************

Library(s) Used:

    slow (File: /theda21_2/CBDK_IC_Contest/cur/SynopsysDC/db/slow.db)

Number of ports:                          794
Number of nets:                          2344
Number of cells:                         1496
Number of combinational cells:           1240
Number of sequential cells:               235
Number of macros/black boxes:               0
Number of buf/inv:                        289
Number of references:                      39

Combinational area:              12584.523772
Buf/Inv area:                     1021.834784
Noncombinational area:            5898.465105
Macro/Black Box area:                0.000000
Net Interconnect area:      undefined  (No wire load specified)

Total cell area:                 18482.988877
Total area:                 undefined
```

6. Encountered Problems and Solving
   - To start this lab, I am confusing how to determine the states I need. Can I directly use instruction as state? I think it is not easy do define and might be ambiguous in changing state. Hence, I define another 7 states, and it make me more easily in later work.
   - If I assign value without using DFF, e.g. `counter`, the value will be wrong value.

7. Questions
   - I use 8 DFFs to store numbers in Stack Memory, is there any other to design a stack machine?

8. Impression and Experience

   Stack Machine is a simple idea, however, in implementation we have to consider many aspects. And this is our last Lab, after these labs and the course, I think that I have basic concept in Verilog! Thank TAs.