

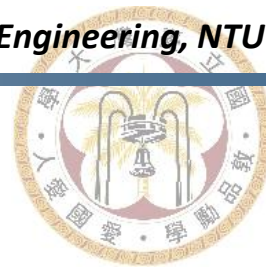
Computer-Aided VLSI System Design

Homework 2: Simple RISC-V CPU

Graduate Institute of Electronics Engineering, National Taiwan University

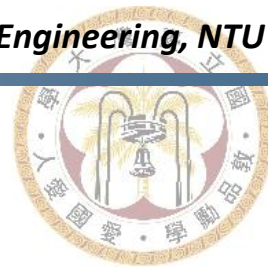


NTU GIEE



Goal

- In this homework, you will learn
 - How to write testbench
 - How to design FSM
 - How to use Memory IP
 - Generate patterns for testing
 - How to implement some RISC-V instruction operations, so **designware is not allowed in this assignment**

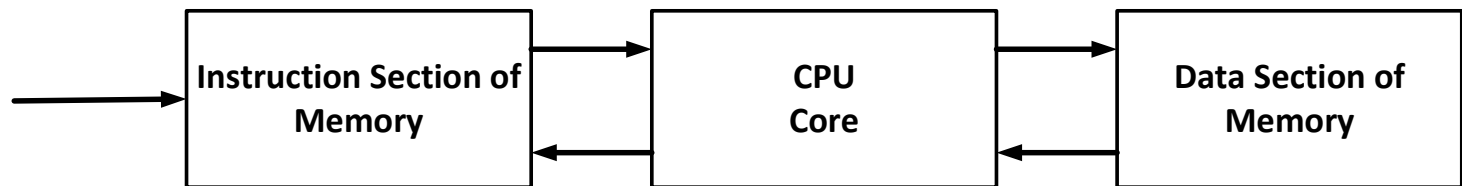


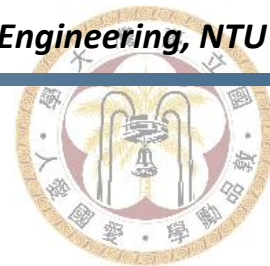
Introduction

- Central Processing Unit (CPU) is the important core in the computer system. In this homework, you are asked to design a simple RISC-V CPU [1], which contains the basic module of program counter, ALU and register files. The instruction set of the simple CPU is similar to RISC-V structure.

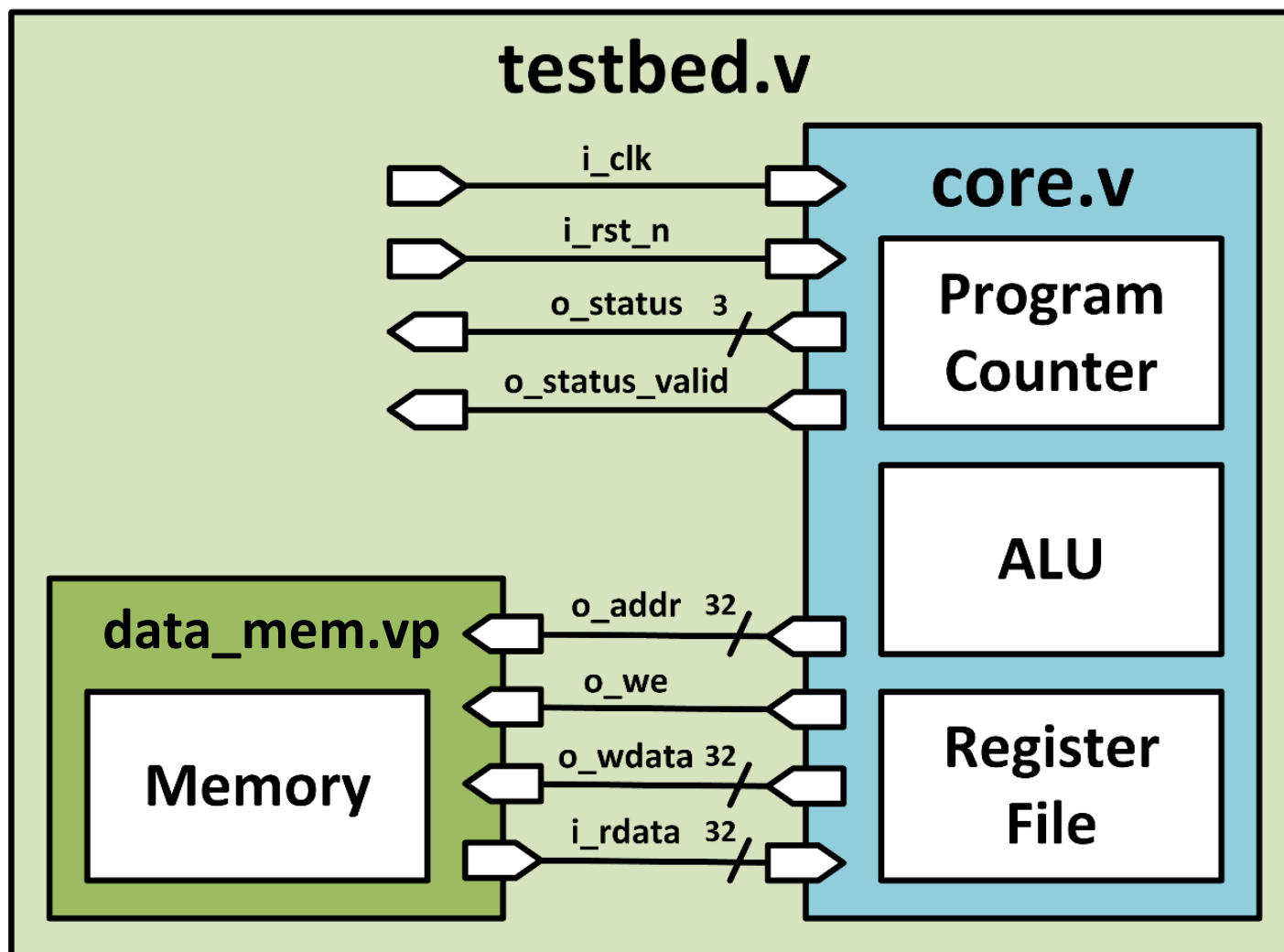
Instruction set

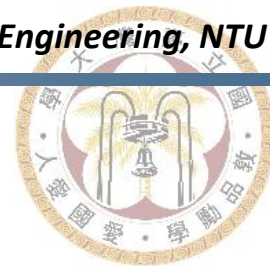
```
fsw    $2    $0    40
fsw    $3    $0    48
fclass $15    $2
flt    $16    $2    $3
flt    $17    $3    $2
blt    $16    $17    8
sw     $15    $0    56
eof
```





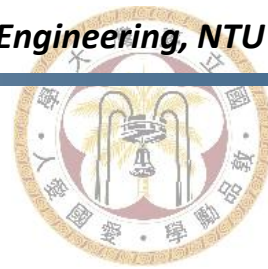
Block Diagram





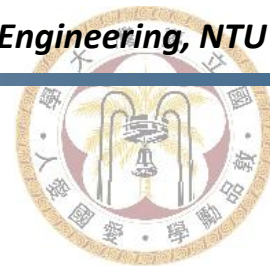
Input/Output

Signal Name	I/O	Width	Simple Description
i_clk	I	1	Clock signal in the system.
i_rst_n	I	1	Active low asynchronous reset.
o_we	O	1	Write enable of memory Set low for reading mode, and high for writing mode
o_addr	O	32	Address for memory
o_wdata	O	32	Data input to memory
i_rdata	I	32	Data or instruction output from memory
o_status	O	3	Status of core processing to each instruction
o_status_valid	O	1	Set high if ready to output status



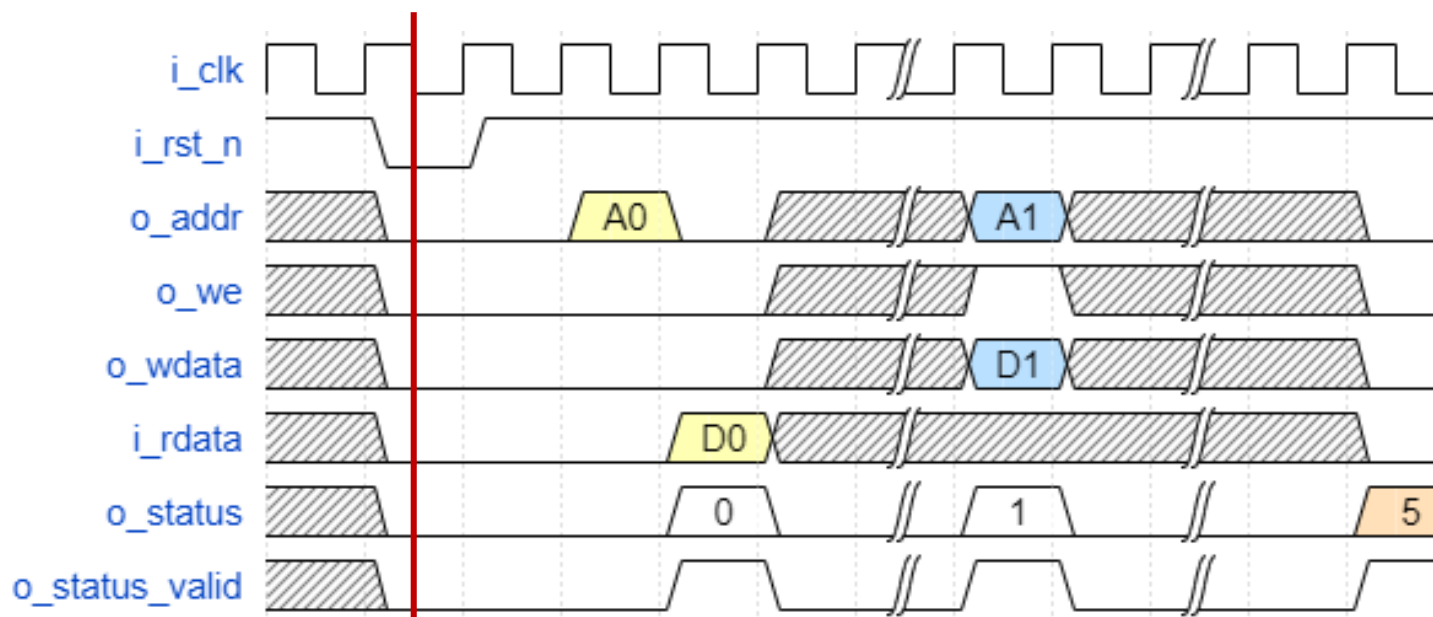
Specification

- All outputs should be synchronized at clock **rising** edge.
- Memory is provided. All values in memory are reset to be zero.
- You should create **32 signed 32-bit registers** and **32 single-precision floating-point registers** in register file.
- Less than **1024** instructions are provided for each pattern.
- The whole processing time can't exceed **120000** cycles for each pattern.
- **x0 register** in this homework is allowed to store values other than 0.

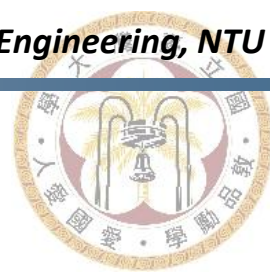


Specification

- You should set all your outputs and register file to be zero when `i_rst_n` is **low**. Active low asynchronous reset is used.

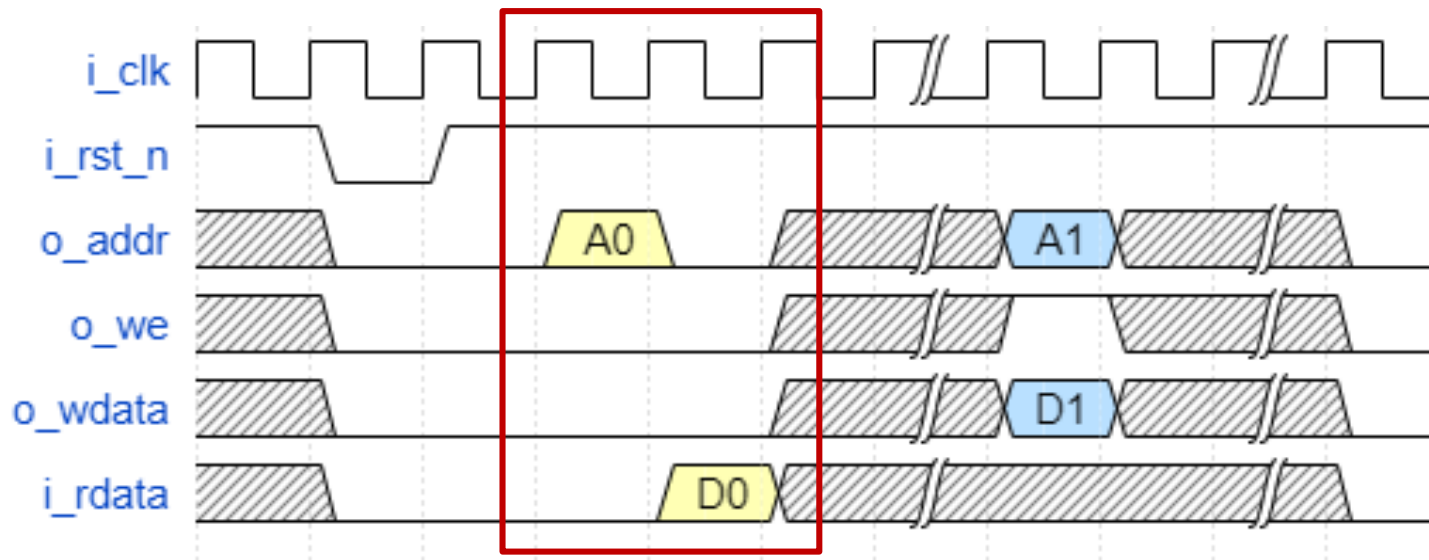


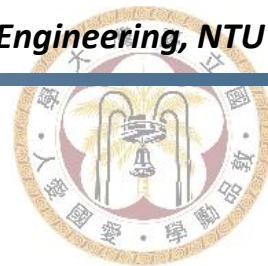
All output must be zero when reset



Specification

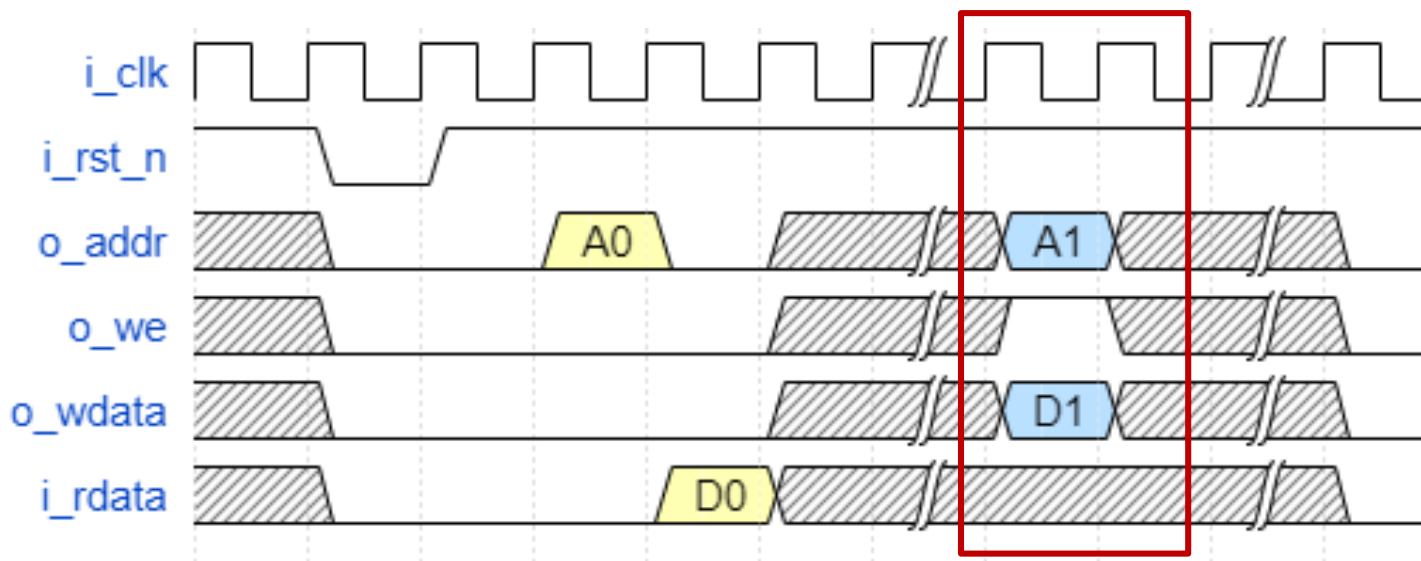
- To load data or instruction from the memory, set o_we to **0** and o_addr to relative address value. i_rdata can be received at the next rising edge of the clock.

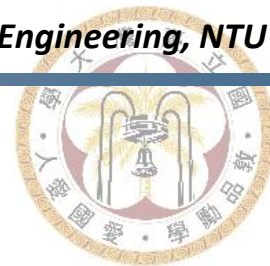




Specification

- To save data to the memory, set o_we to **1**, o_addr to relative address value, and o_wdata to the written data.





Instruction mapping

▪ R-type

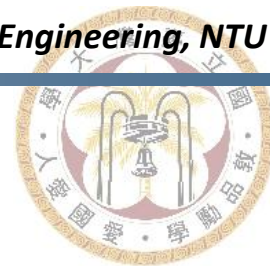
[31:25]	[24:20]	[19:15]	[14:12]	[11:7]	[6:0]
funct7	r2/f2	r1/f1	funct3	rd/fd	opcode

▪ I-type

[31:20]	[19:15]	[14:12]	[11:7]	[6:0]
imm[11:0]	r1/f1	funct3	rd/fd	opcode

▪ S-type

[31:25]	[24:20]	[19:15]	[14:12]	[11:7]	[6:0]
imm[11:5]	r2/f2	r1/f1	funct3	imm[4:0]	opcode



Instruction mapping (cont'd)

▪ B-type

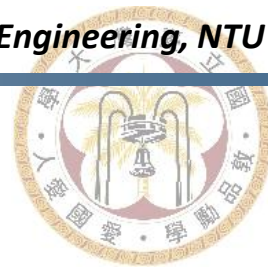
[31]	[30:25]	[24:20]	[19:15]	[14:12]	[11:8]	[7]	[6:0]
imm[12]	imm[10:5]	r2/f2	r1/f1	funct3	imm[4:1]	imm[11]	opcode

▪ U-type

[31:12]	[11:7]	[6:0]
imm[31:12]	rd/fd	opcode

▪ EOF

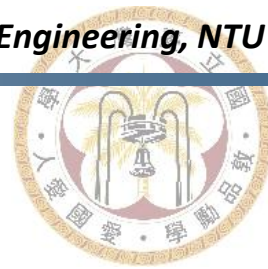
[31:7]	[6:0]
Not used	opcode



Status

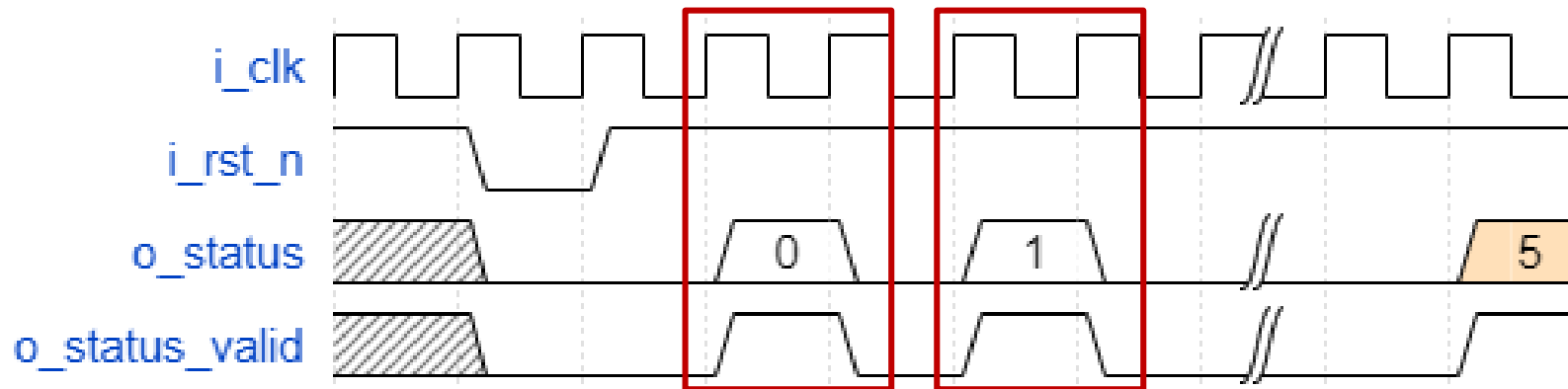
- 7 statuses of o_status

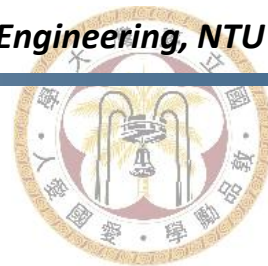
o_status[2:0]	Definition
3'd0	R_TYPE_SUCCESS
3'd1	I_TYPE_SUCCESS
3'd2	S_TYPE_SUCCESS
3'd3	B_TYPE_SUCCESS
3'd4	U_TYPE_SUCCESS
3'd5	INVALID_TYPE
3'd6	EOF_TYPE



Specification

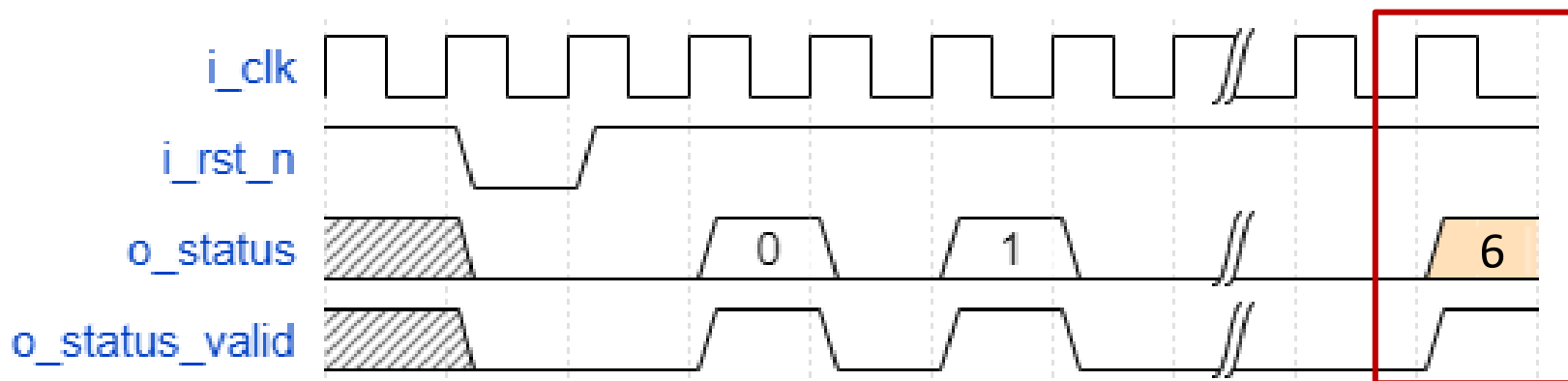
- Your `o_status_valid` should be turned to **high** for only **one cycle** for every `o_status`.
- The testbench will get your output at negative clock edge to check the `o_status` if your `o_status_valid` is **high**.

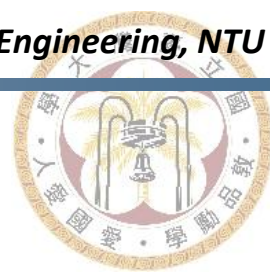




Specification

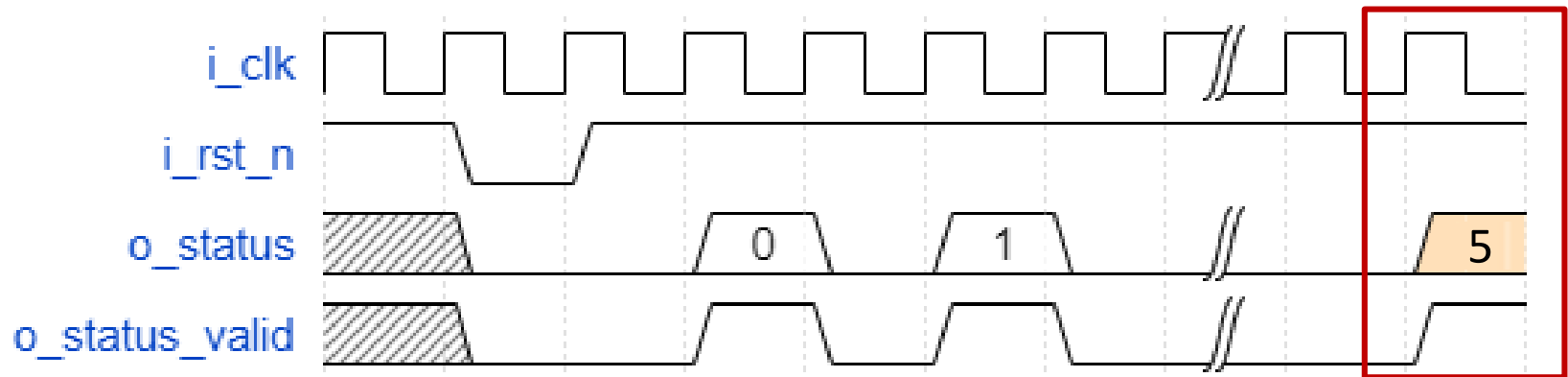
- When you set o_status_valid to **high** and o_status to **6**, stop processing. The testbench will check your memory value with golden data.

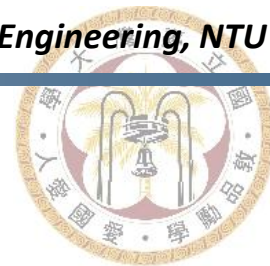




Specification

- If **invalid operation happened** (see p.24~26), stop processing and raise `o_status_valid` to **high** and set `o_status` to **5**. The testbench will check your memory value with golden data.

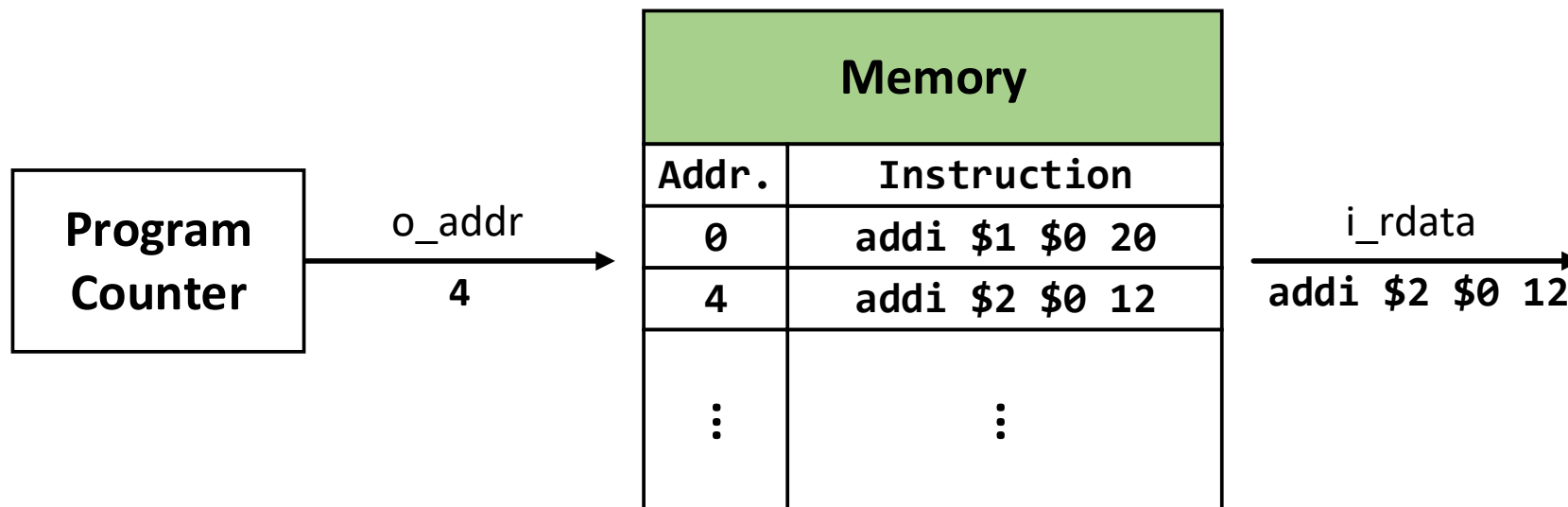


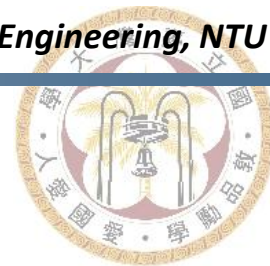


Program Counter

- Program counter is used to control the address of memory for instruction.

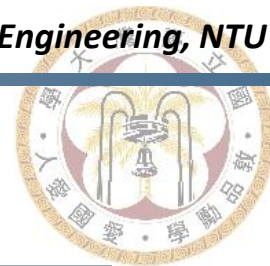
$\$pc = \$pc + 4$ for every instruction (except **beq, **blt**, **jalr**)**





Instruction

Operation	Assemble	Type	Meaning	Note
Subtract	sub	R	$\$rd = \$r1 - \$r2$	Signed Operation
Add immediate	addi	I	$\$rd = \$r1 + im$	Signed Operation
Load word	lw	I	$\$rd = Mem[\$r1 + im]$	Signed Operation
Store word	sw	S	$Mem[\$r1 + im] = \$r2$	Signed Operation
Branch on equal	beq	B	if($\$r1 == \$r2$), $\$pc = \$pc + im$; else, $\$pc = \$pc + 4$	PC-relative Signed Operation
Branch less than	blt	B	if($\$r1 < \$r2$), $\$pc = \$pc + im$; else, $\$pc = \$pc + 4$	PC-relative Signed Operation
Jump and link register	jalr	I	$\$rd = \$pc + 4$; $\$pc = (\$r1 + im) \& (\sim 0x1)$	PC-relative Signed Operation
Add upper immediate to PC	auipc	U	$\$rd = \$pc + (im \ll 12)$	PC-relative Signed Operation
Set on less than	slt	R	if($\$r1 < \$r2$), $\$rd = 1$; else, $\$rd = 0$	Signed Operation
Shift right logical	srl	R	$\$rd = \$r1 \gg \$r2$	Unsigned Operation



Instruction (cont'd)

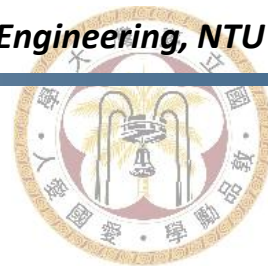
Operation	Assemble	Type	Meaning	Note
Floating-point subtract	fsub	R	$\$fd = \$f1 - \$f2$	Floating-point Operation
Floating-point multiply	fmul	R	$\$fd = \$f1 * \$f2$	Floating-point Operation
Floating-point to signed integer conversion	fcvt.w.s	R	$\$rd = s32f32(\$f1)$	Floating-point Operation
Load floating-point	flw	I	$\$fd = \text{Mem}[\$r1 + im]$	Signed Operation
Store floating-point	fsw	S	$\text{Mem}[\$r1 + im] = \$f2$	Signed Operation
Floating-point classify	fclass	R	$\$rd = \text{fclass}(\$f1)$	Classify floating-point format
End of File	eof	EOF	Stop processing	Last instruction in the pattern

Note: The notation of **im** in instruction is **2's complement**.

Note: The \$r notes that the data is read/written to **integer register file**; the \$f notes that the data is read/written to **floating-point register file**.

Note: Set the result of fsub and fmul to +0 if the arithmetic result is 0.

Note: **x0 register** in this homework is allowed to store values other than 0.



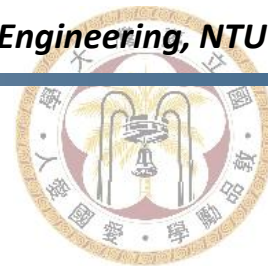
Floating Point

- For instructions **fsub**, **fmul**, **fcvt.w.s**, **fclass**, you will have to implement operations with **floating point** format
- IEEE-754 single precision format [2]
 - 1 signed bit
 - 8 exponent bit
 - 23 mantissa bit



31

0



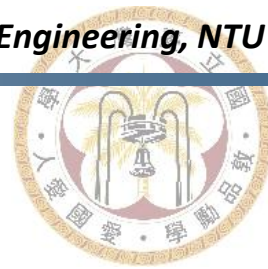
IEEE-754 Single Precision Format

[31]	[30:23]	[22:0]
sign	exponent	mantissa

31

0

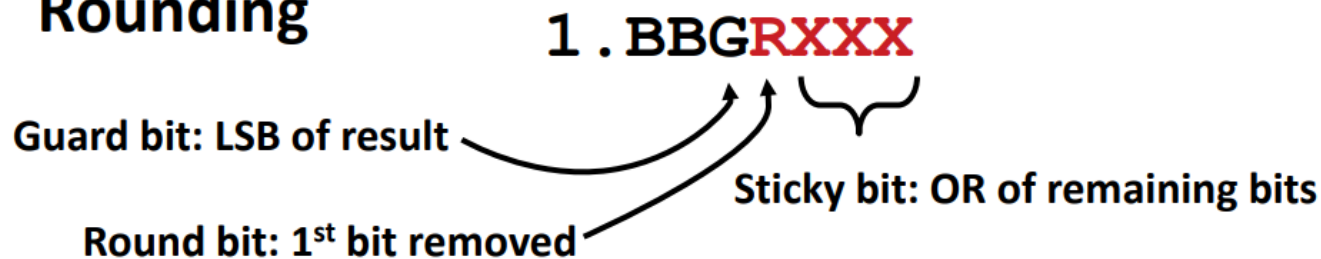
Single-Format Bit Pattern	Value
$0 < e < 255$	$(-1)^s \times 2^{e-127} \times 1.m$ (normal numbers)
$e = 0; m \neq 0$ (at least one bit in f is nonzero)	$(-1)^s \times 2^{-126} \times 0.m$ (subnormal numbers)
$e = 0; m = 0$ (all bits in f are zero)	$(-1)^s \times 0.0$ (signed zero)
$s = 0; e = 255; m = 0$ (all bits in f are zero)	+INF (positive infinity)
$s = 1; e = 255; m = 0$ (all bits in f are zero)	-INF (negative infinity)
$e = 255; m \neq 0$ (at least one bit in f is nonzero)	NaN (Not-a-Number)



Round to Nearest Even

- For instructions **fsub**, **fmul**, **fcvt.w.s**, you will have to round the mantissa or decimal with **round to nearest even** [3]

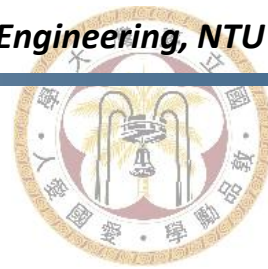
Rounding



Round up conditions

- Round = 1, Sticky = 1 → > 0.5
- Guard = 1, Round = 1, Sticky = 0 → Round to even

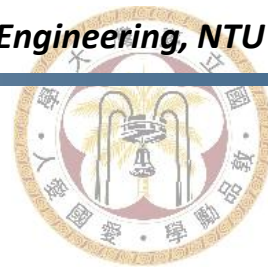
<i>Value</i>	<i>Fraction</i>	<i>GRS</i>	<i>Incr?</i>	<i>Rounded</i>
128	1.000 000	000	N	1.000
15	1.101 000	100	N	1.101
17	1.000 100	010	N	1.000
19	1.001 100	110	Y	1.010
138	1.000 1010	011	Y	1.001
63	1.111 1100	111	Y	10.000



Floating Point Classification

- For instruction **fc_{lass}**, you will have to classify the floating-point number stored in registers and a bitmask with the corresponding bit set should be written to the destination register

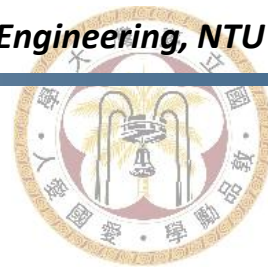
rd bit	Meaning
0	Negative infinite
1	Negative normal number
2	Negative subnormal number
3	Negative zero
4	Positive zero
5	Positive subnormal number
6	Positive normal number
7	Positive infinite
8	Signaling NaN
9	Quiet NaN



Memory IP

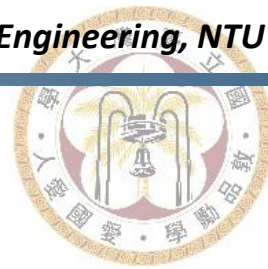
- Size: 2048×32 bit
- `i_addr[12:2]` for address mapping in memory
- Instructions are stored in address 0 - address 4095 of memory
- Data should be read from and written to address 4096 - address 8191 of memory

```
module data_mem (  
    input          i_clk,  
    input          i_rst_n,  
    input          i_we,  
    input [ 31 : 0 ] i_addr,  
    input [ 31 : 0 ] i_wdata,  
    output [ 31 : 0 ] o_rdata  
);
```



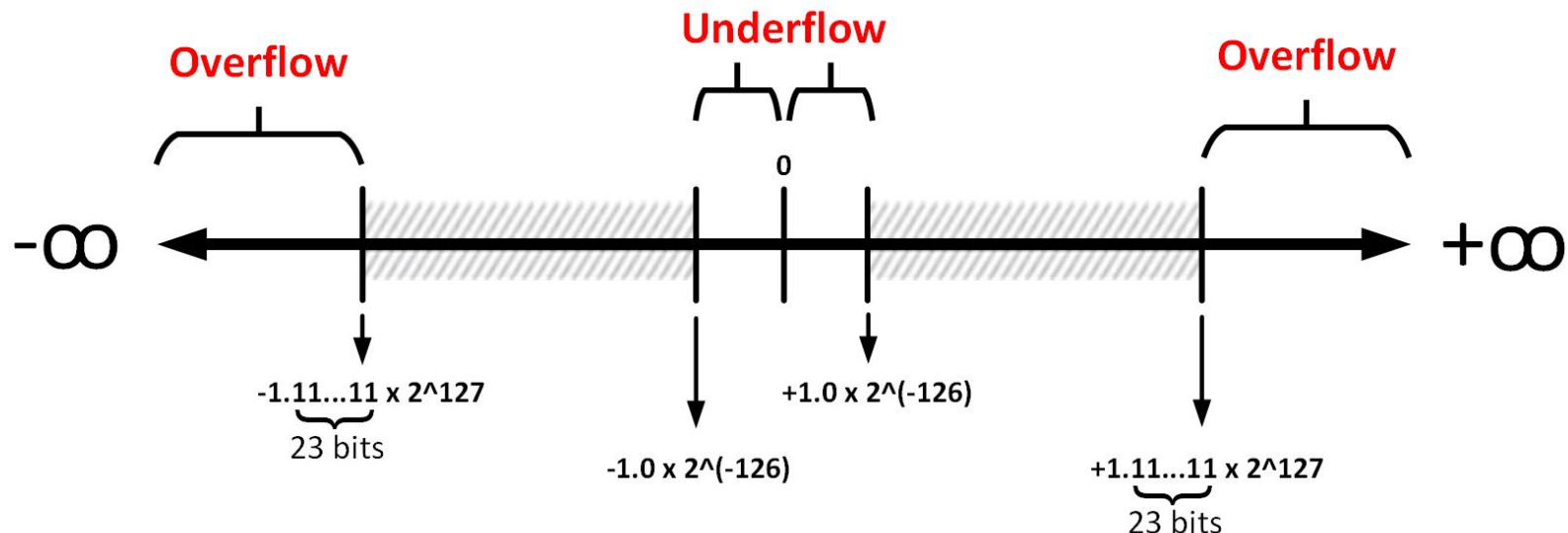
Invalid operation

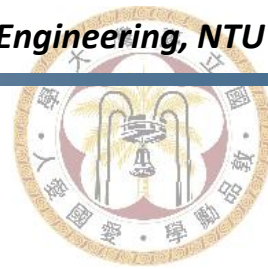
- Invalid operation may happened.
 - **Situation1**: Overflow happened at integer arithmetic instructions (**sub**, **addi**)
 - **Situation2**: Infinite, NaN happened at floating-point arithmetic instructions (**fsub**, **fmul**, **fcvt.w.s**)
 - For fcvt.w.s instruction, additional consideration is required when the floating-point value being converted to a 32-bit signed integer exceeds the representable range of a 32-bit signed integer
 - Do not consider when loading/storing infinite or NaN numbers from memory
 - Do not consider when executing **fclass** on infinite or NaN numbers



Invalid operation

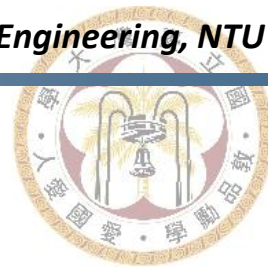
- Invalid operation may happen.
 - Situation3:** Overflow and underflow result happened at floating-point arithmetic instructions (**fsub**, **fmul**)
 - Consider the overflow and underflow before rounding arithmetic result
 - Underflow does not include zero





Invalid operation

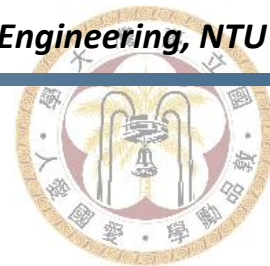
- Invalid operation may happened.
 - **Situation4**: If output address are mapped to unknown address in memory.
 - Consider the case when trying to load/store the address of memory for instruction
 - Consider the case when program counter is fetching instruction from the address of memory for data
 - Do not consider the case if instruction address is beyond eof, but the address mapping is in the size of memory for instruction



rtl.f

- Filelist

```
// -----  
// Simulation: HW2 simple RISC-V CPU  
// -----  
  
// define files: Do not modify  
// -----  
../00_TB/define.v  
  
// testbench: Do not modify  
// -----  
../00_TB/testbed.v  
../00_TB/data_mem.vp  
  
// design files: Be free to add your design files  
// -----  
./core.v
```



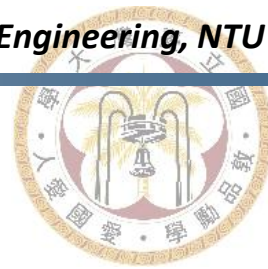
core.v

- Do not modify interface

```
module core #( // DO NOT MODIFY INTERFACE!!!
    parameter DATA_WIDTH = 32,
    parameter ADDR_WIDTH = 32
) (
    input i_clk,
    input i_rst_n,

    // Testbench IOs
    output [2:0] o_status,
    output      o_status_valid,

    // Memory IOs
    output [ADDR_WIDTH-1:0] o_addr,
    output [DATA_WIDTH-1:0] o_wdata,
    output                  o_we,
    input  [DATA_WIDTH-1:0] i_rdata
);
```



define.v

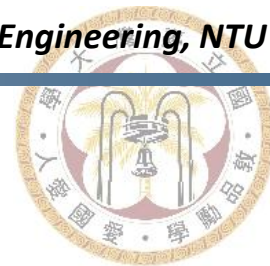
- Do not modify
- Define all the 17 instruction patterns

```
// DO NOT MODIFY THIS FILE
// status definition
`define R_TYPE 0
`define I_TYPE 1
`define S_TYPE 2
`define B_TYPE 3
`define U_TYPE 4
`define INVALID_TYPE 5
`define EOF_TYPE 6

// opcode definition
`define OP_SUB 7'b0110011
`define OP_ADDI 7'b0010011
`define OP_LW 7'b0000011
`define OP_SW 7'b0100011
`define OP_BEQ 7'b1100011
`define OP_BLT 7'b1100011
`define OP_JALR 7'b1100111
`define OP_AUIPC 7'b0010111
`define OP_SLT 7'b0110011
`define OP_SRL 7'b0110011
`define OP_FSUB 7'b1010011
`define OP_FMUL 7'b1010011
`define OP_FCVTWS 7'b1010011
`define OP_FLW 7'b0000111
`define OP_FSW 7'b0100111
`define OP_FCLASS 7'b1010011
`define OP_EOF 7'b1110011
```

```
// funct7 definition
`define FUNCT7_SUB 7'b0100000
`define FUNCT7_SLT 7'b0000000
`define FUNCT7_SRL 7'b0000000
`define FUNCT7_FSUB 7'b0000100
`define FUNCT7_FMUL 7'b0001000
`define FUNCT7_FCVTWS 7'b1100000
`define FUNCT7_FCLASS 7'b1110000

// funct3 definition
`define FUNCT3_SUB 3'b000
`define FUNCT3_ADDI 3'b000
`define FUNCT3_LW 3'b010
`define FUNCT3_SW 3'b010
`define FUNCT3_BEQ 3'b000
`define FUNCT3_BLT 3'b100
`define FUNCT3_JALR 3'b000
`define FUNCT3_SLT 3'b010
`define FUNCT3_SRL 3'b101
`define FUNCT3_FSUB 3'b000
`define FUNCT3_FMUL 3'b000
`define FUNCT3_FCVTWS 3'b000
`define FUNCT3_FLW 3'b010
`define FUNCT3_FSW 3'b010
`define FUNCT3_FCLASS 3'b000
```

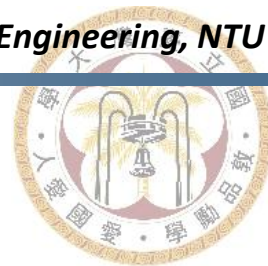


testbed_temp.v

- Things to add in your testbench
 - Clock
 - Reset
 - Waveform file
 - Function test
 - ...

```
module testbed;  
  
    wire clk, rst_n;  
    wire          dmem_we;  
    wire [ 31 : 0 ] dmem_addr;  
    wire [ 31 : 0 ] dmem_wdata;  
    wire [ 31 : 0 ] dmem_rdata;  
    wire [ 1 : 0 ] mips_status;  
    wire          mips_status_valid;
```

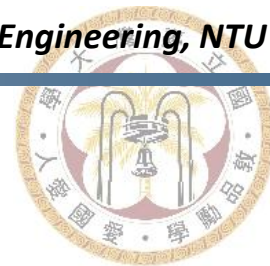
```
core u_core (  
    .i_clk(),  
    .i_rst_n(),  
    .o_status(),  
    .o_status_valid(),  
    .o_we(),  
    .o_addr(),  
    .o_wdata(),  
    .i_rdata()  
);  
  
data_mem u_data_mem (  
    .i_clk(),  
    .i_rst_n(),  
    .i_we(),  
    .i_addr(),  
    .i_wdata(),  
    .o_rdata()  
);
```



Protected Files

- The following files are protected
 - data_mem.vp

```
module data_mem (
    input          i_clk,
    input          i_rst_n,
    input          i_we,
    input [ 31 : 0 ] i_addr,
    input [ 31 : 0 ] i_wdata,
    output [ 31 : 0 ] o_rdata
);
`protected
&6JU@A,>B[ZKNH#f\dwJ5ZgKY/4LTZcTK[9H@IT99E_YU\L\&A8-)gLM#\H80&9
CAINT2\;]80c#b5-A;1-?4M?C77#/U@_1&DWDI#/gT[Vd?L&5U#I6: :&,-e822f.
dPcB[;AOLA8FQd+Td+L2#YEY+#D1JX1Q#6TF0N^_2@aJc(RIWe8:AN=DV.0XBTP-
B,<E/\4X\GAJbwfYF)g07^)83,802)?K+>I,9M(UX0Sg2?g4RW:^,Y^?JH28>J=8
2FK>6\HU(3?LIBQQK9(:WZ+e/KCQgI/<T8FPN0KCIcU/1.=L;VQCBO3PPV+G_:1\
8N,g9>],5^](9f(g?^R[DW>/[/OTa>S).K4-C=85)5S>FC6La0\2g9Q+,Ad7fBF?
b6XA=:M7[_3COF+_59;H6E-Dfc7#U+&/A/A]WDWU>QUW.124=b>LE5EE04f6J:W)
44Za5?:](CHHVagBN[2/dBwMJ?2NgZ6,WN^P[W@YaI+,0]=Yb_W+?5AK/\a>SBF-
Z6M;_KM/.e05RCFK+_M?^IJI8)@,@J1N^DOE033(<Rg3df<=W#b]EB3dc0g[TOB
09CRJ3G3+DbS=;VI?_&/1f-VHY/5:WE,U<3g;#d]0eRaUU4-BDZ9P-@U\Q_4&W[B
IEB(fLJM45&JGf.&MX@=N#QdV1@;gc#d0ZR/Kc@6+PfE17d.+SOf6L(+ (QON-KUM
4FHe<QSVE;JNgd1U(Z0D1B57Z]RZWU^L>;>ZITDL1T?-) \E=KEF<8]5IO19@fZA-
f4;NUL/a9(7</dS#+;:_9aX4P&UC^8:=1g-,b&F4I5=P_e[6+99gHL+a]W/R8C()
P40gM;E>@Y1V1d9/fIP7PN1:#ffG-FUS=@?bU9SE(>=^dL,;]DOOX0RU00ZKaX,\
@,GLKWM,gX:DcdF2W@8M92XHHdcN>Q?M03I,C9HLE(@3=G/bb[J;TB=gLTSBB>f2
0S0;V?<6,FW=NI@^H#<aM]@)29VETb]B1Cg7gN(9CC@-2TR/;NDFdF=gM$
`endprotected
```



Command

- 01_run
 - Usage: ./01_run p0

```
vcs -full64 -R -f rtl.f +v2k -sverilog -debug_access+all +define+$1 | tee sim.log
```

- 99_clean_up

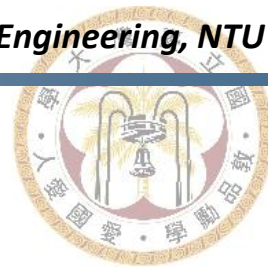
```
rm -rf *.history *.key *.log
```

```
rm -rf novas.rc novas.fsdb novas.conf
```

```
rm -rf INCA_libs nWaveLog BSSLib.lib++
```




- p. 34

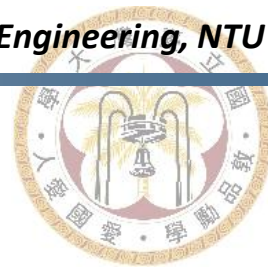


Grading Policy

- TA will run your code with following command (p0 is example)

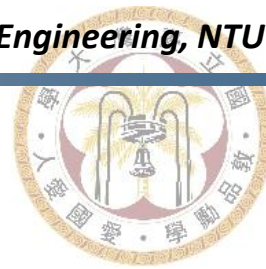
```
vcs -f rtl.f -full64 -sverilog -R -debug_access+all  
+define+p0 -v2k
```

- Pass the patterns to get full score
 - Provided pattern: **70%** (4 patterns in total)
 - **15%** for each test
 - **10%** for spyglass check (lint_rtl and lint_rtl_enhanced)
 - Hidden pattern: **30%**
 - **20 patterns** in total



Grading Policy

- **Deadline:** 2025/10/14 13:59:59 (UTC+8)
- Late submissions are not accepted
 - Any submission after the deadline will receive 0 points
- File corrections after the deadline should be avoided
 - Corrections for the folder name, file name, file hierarchy cause 5-point deduction
- The TA will grade your submissions by using scripts
- **No plagiarism**
 - Plagiarism in any form, including copying from online sources, is strictly prohibited

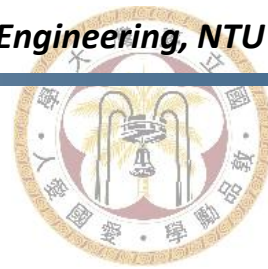


Submission

- Create a folder named **studentID_hw2**, and put all below files into the folder

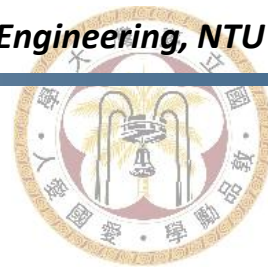
```
r13943119_hw2/  
├── 01_RTL  
│   ├── core.v  
│   ├── rtl.f  
│   └── (other design files)
```

- Compress the folder **studentID_hw2** in a tar file named **studentID_hw2_vk.tar** (k is the number of version, $k = 1, 2, \dots$)
 - Use lower case for student ID. (Ex. r13943119_hw2_v1.tar)
- Submit to NTU Cool



Hint

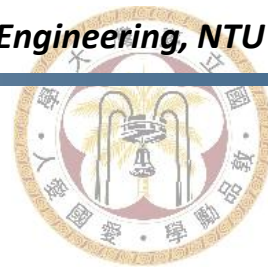
- Design your FSM with following states
 1. Idle
 2. Instruction Fetching
 3. Instruction decoding
 4. ALU computing/ Load data
 5. Data write-back
 6. Next PC generation
 7. Process end



Discussion

- **NTU Cool Discussion Forum**
 - For any questions not related to assignment answers or privacy concerns, please use the NTU Cool discussion forum.
 - **TAs will prioritize answering questions on the NTU Cool discussion forum**

- **Email: r13943119@ntu.edu.tw**
 - Title should start with **[CVSD 2025 Fall HW2]**
 - Email with wrong title will be moved to trash automatically



Reference

- [1] RISC-V User Manual
 - <https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf>
- [2] IEEE 754 Single Precision Format
 - https://zh.wikipedia.org/zh-tw/IEEE_754
- [3] Round to Nearest Even
 - <https://www.cs.cmu.edu/afs/cs/academic/class/15213-s16/www/lectures/04-float.pdf>
- [4] F Standard Extension for Single-Precision Floating-Point
 - <https://five-embeddev.com/riscv-user-isa-manual/Priv-v1.12/f.html>