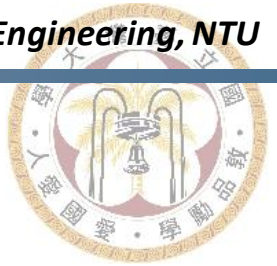
Computer-Aided VLSI System Design

Homework 2: Simple RISC-V CPU

Graduate Institute of Electronics Engineering, National Taiwan University

NTU GIEE



Goal

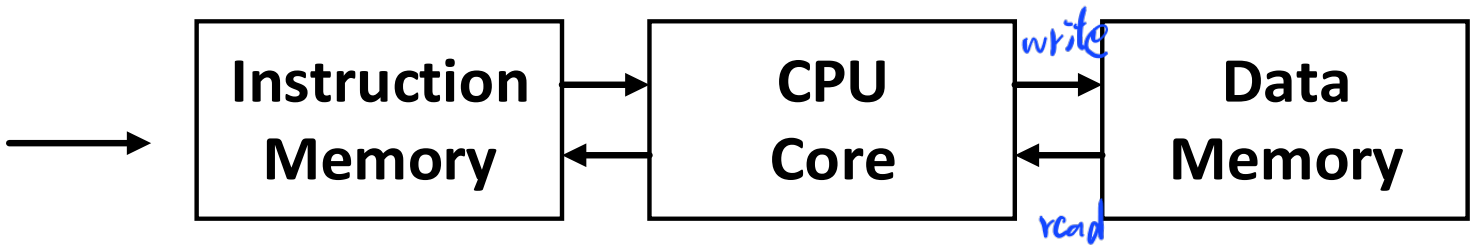
- In this homework, you will learn
  - How to write testbench
  - How to design FSM
  - How to use IP TA 提供的 SRAM
  - Generate patterns for testing

Introduction

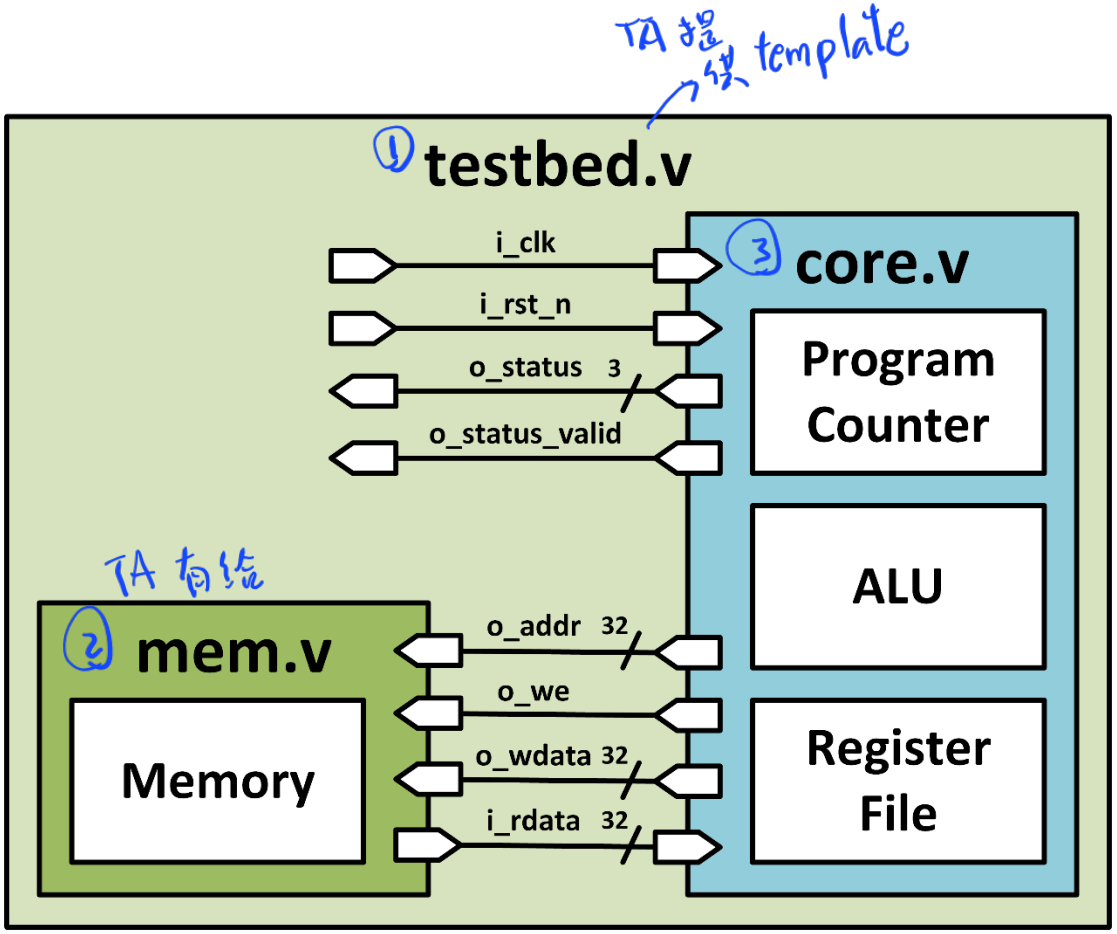
- Central Processing Unit (CPU) is the important core in the computer system. In this homework, you are asked to design a simple RISC-V CPU [1], which contains the basic module of program counter, ALU and register files. The instruction set of the simple CPU is similar to RISC-V structure.

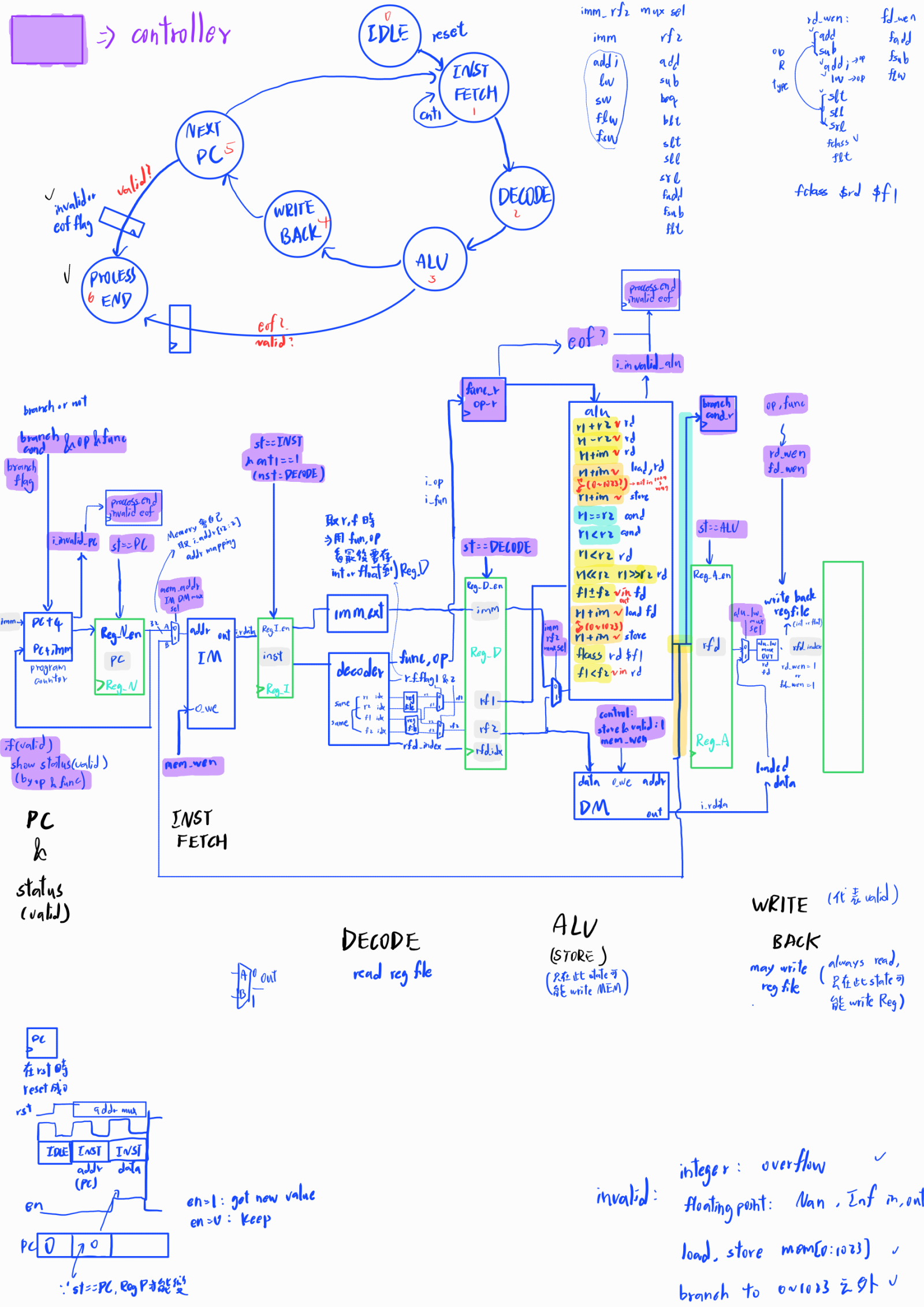
Instruction set

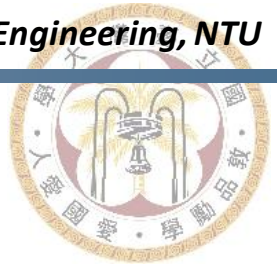
```
addi $7 $3 4
sub  $7 $7 $5
sw   $7 $4 8
bne  $3 $5 12
lw   $6 $0 8
add  $7 $6 $2
sw   $7 $4 8
eof
```



Block Diagram




=> controller
IDLE
reset
INST FETCH
cnt1
DECODE
ALU
WRITE BACK
NEXT PC
valid?
invalid or eof flag
PROCESS & END
eof & valid?
imm_rf2 mux sel
imm
addi
lw
sw
flw
fsw
rf2
add
sub
beq
blt
slt
sll
srl
fadd
fsub
flt
rd_wen:
OP R type
fadd
fsub
flw
fclass
slt
sll
srl
fclass $rd $f1
branch or not
branch cond & op & func
branch flag
process_end invalid eof
i_invalid_PC
st==PC
Memory 會自己設 i_addr[12:2] addr mapping
mem_addr IM DM mux sel
PC+4
PC+imm
program counter
Reg_N_en
PC
Reg_N
addr out
IM
O_we
mem_wen
if(valid) show status(valid) (by op & func)
st==INST & cnt1==1 (next: DECODE)
Reg_I_en
inst
Reg_I
i_rdata
imm_ext
decoder
same
same
func, op
rf_flag1 & 2
rfd_index
fsw r,f 時 => 用 fun, op 看最後要存 int or float到 Reg_D
i_op
i_fun
st==DECODE
Reg_D_en
imm
Reg_D
rf1
rf2
rfd_idx
imm rf2 mux sel
control: store & valid: 1 mem_wen
data O_we addr
DM
out
i_rdata
loaded data
eof?
process_end invalid eof
i_invalid_alu
func_r op_r
alu
r1+r2 rd
r1-r2 rd
r1+imm rd
r1+imm load, rd
(0~1023)
r1+imm store
r1==r2 cond
r1<r2 cond
r1<r2 rd
r1<<r2 r1>>r2 rd
f1±f2 vin fd
r1+imm load fd
(0~1023)
r1+imm store
fclass rd $f1
f1<f2 vin rd
branch cond_r
st==ALU
Reg_A_en
rfd
Reg_A
alu_lw mux sel
alu_lw mux OUT
rd fd
rfd_index
rd_wen=1 or fd_wen=1
op, func
rd_wen fd_wen
write back regfile (int or float)
PC & status (valid)
INST FETCH
DECODE
read reg file
ALU
(STORE)
(只在此 state 可能 write MEM)
WRITE BACK
(if 是 valid)
may write reg file
(always read, 只在此 state 可能 write Reg)
A 0 out
B 1
PC
在 rst 時 reset 成 0
rst
addr mux
IDLE
INST addr (PC)
INST data
en
PC
0
0
∵ st==PC, Reg_P 才能改變
en=1: get new value
en=0: keep
invalid:
integer: overflow ✓
floating point: Nan, Inf in, out
load, store mem[0:1023] ✓
branch to 0or1023 之外 ✓



Input/Output

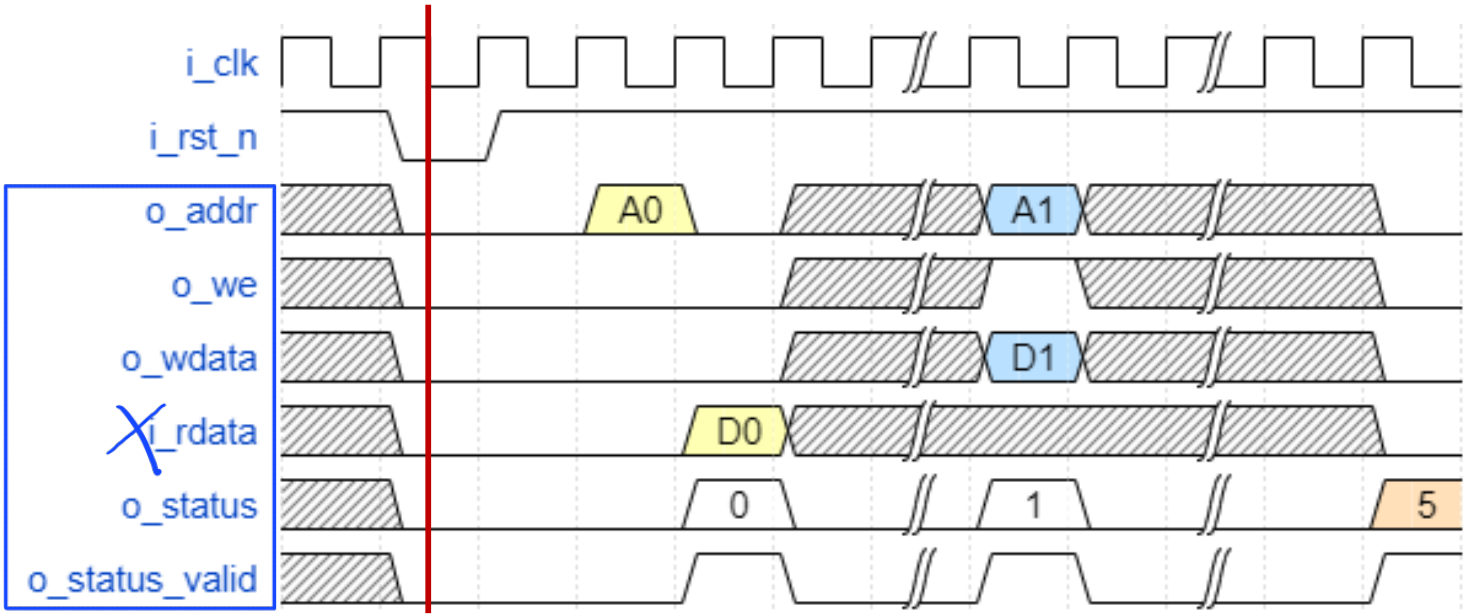
Signal Name	I/O	Width	Simple Description
i_clk	I	1	Clock signal in the system.
i_rst_n	I	1	Active low asynchronous reset.
o_we	O	1	Write enable of memory Set low for reading mode, and high for writing mode
o_addr	O	32	Address for memory
o_wdata	O	32	Unsigned data input to memory
i_rdata	I	32	Unsigned data 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.

Specification

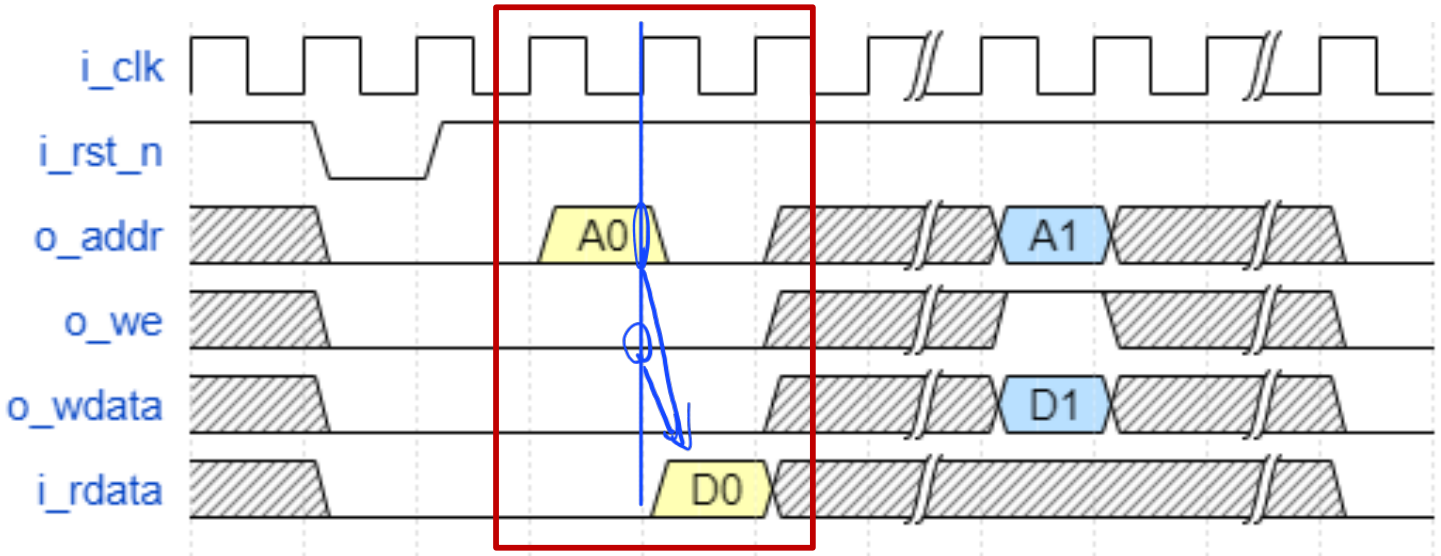
- You should set all your outputs and register file to be zero when i_rst_n is **low**. Active low asynchronous reset is used.



All output must be zero when reset

Specification

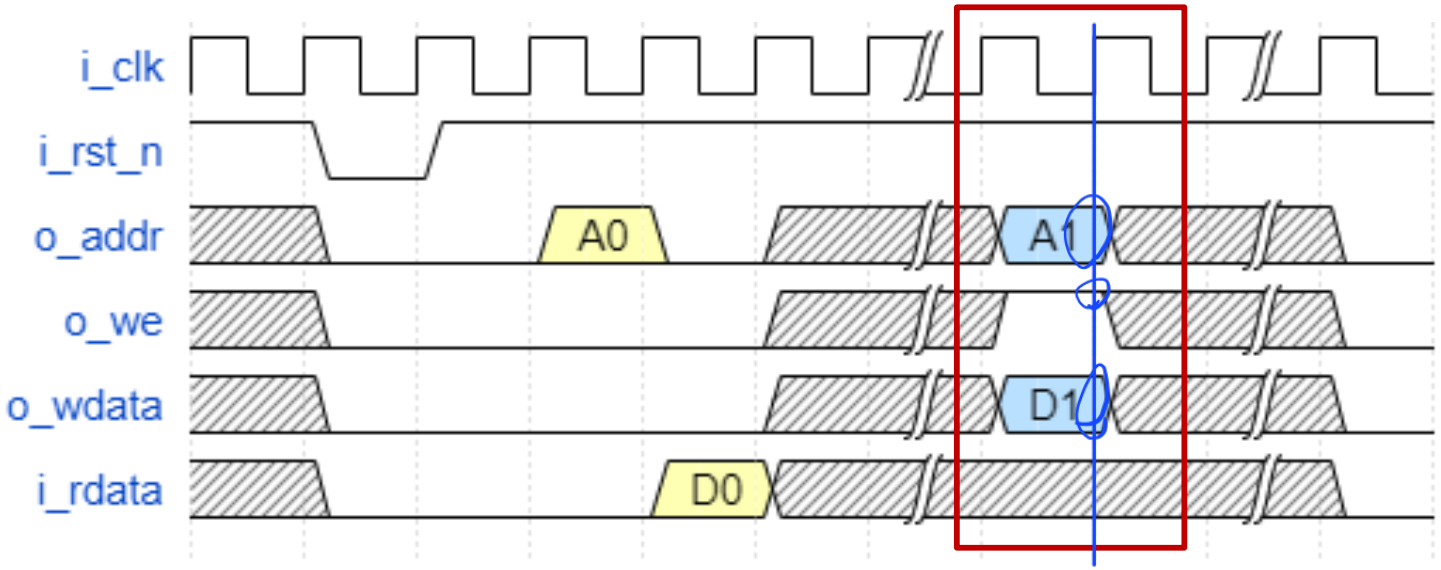
- To load data from the data memory, set o_d_we to **0** and o_d_addr to relative address value. i_d_rdata can be received at the next rising edge of the clock.



SRAM IO

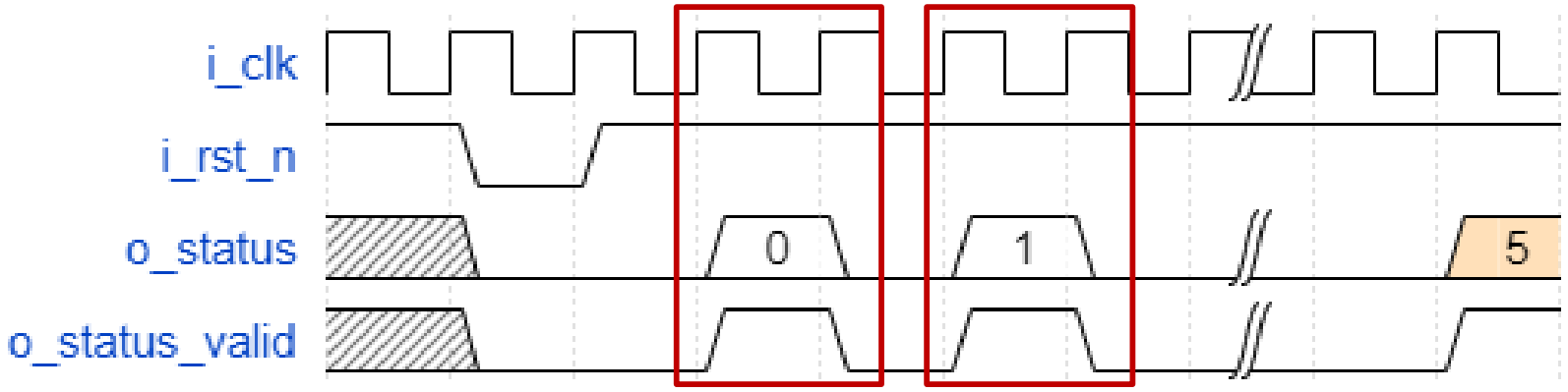
Specification

- To save data to the data memory, set o_d_we to **1**, o_d_addr to relative address value, and o_d_wdata to the written data.



Specification

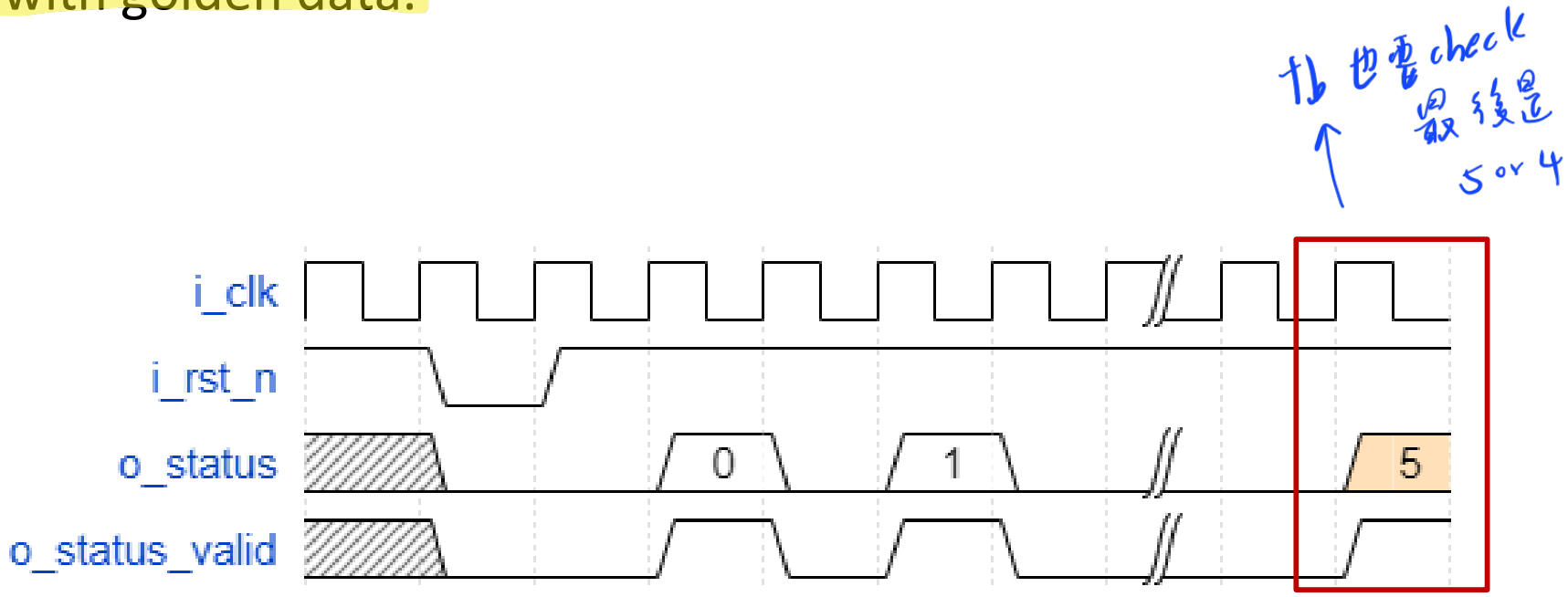
- Your o_status_valid should be turned to **high** for only one cycle for every o_status.
- The testbench will get your output at negative clock edge to check the o_status if your o_status_valid is **high**.



每个 instruction 結完
都要 valid = 1

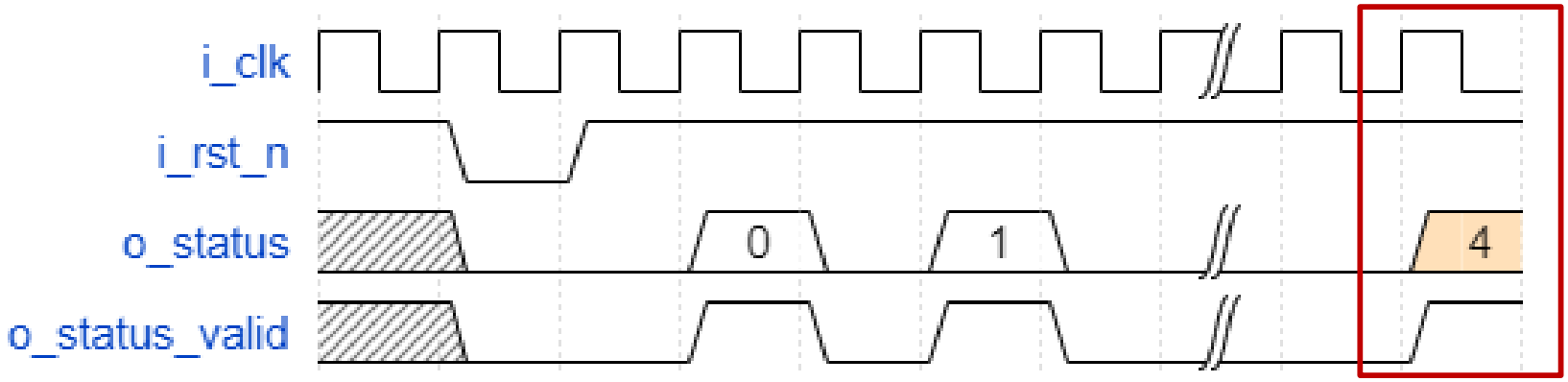
Specification

- When you set o_status_valid to **high** and o_status to **5**, stop processing. The testbench will check your data memory value with golden data.



Specification

- If overflow happened, stop processing and raise o_status_valid to **high** and set o_status to **4**. The testbench will check your data memory value with golden data.

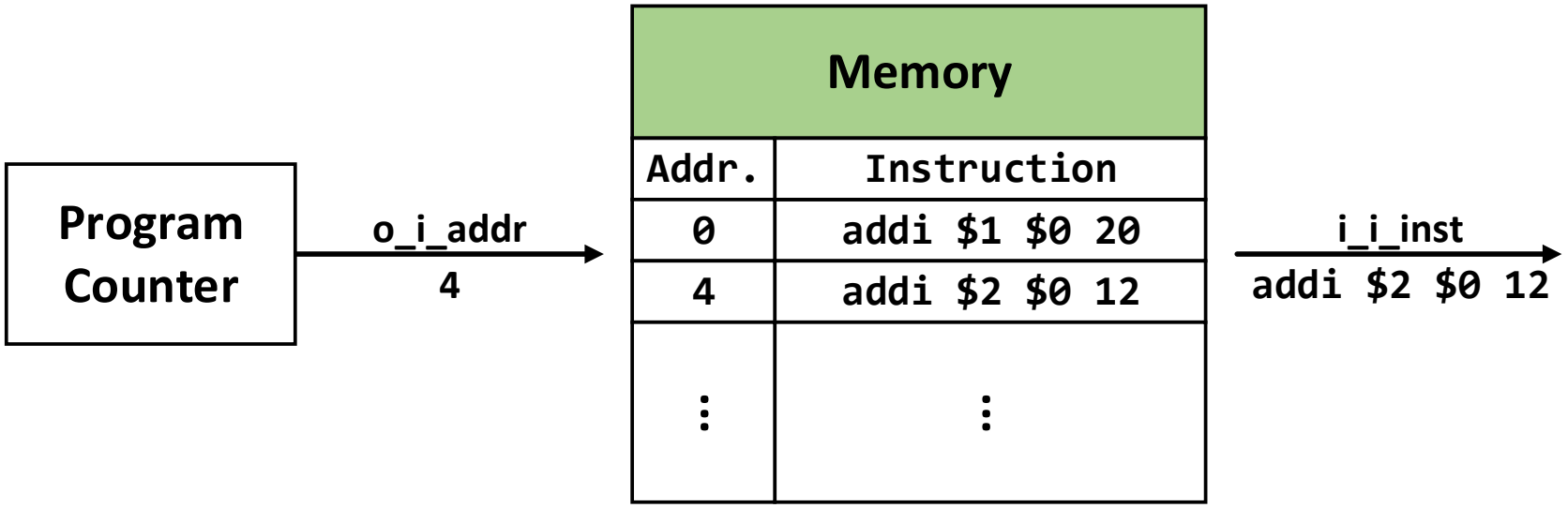


tb 看到 4 就
finish
(也可一个 cycle)

Program Counter

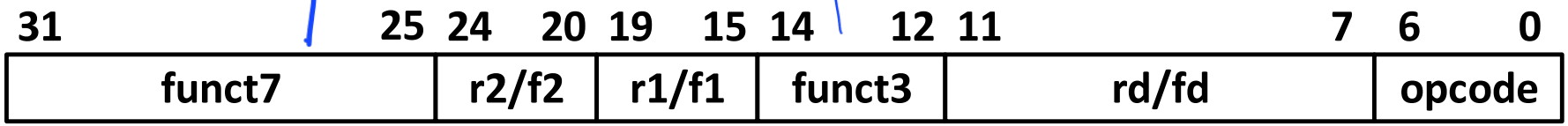
- Program counter is used to control the address of instruction memory.

`$pc = $pc + 4` for every instruction (except `beq, blt`)

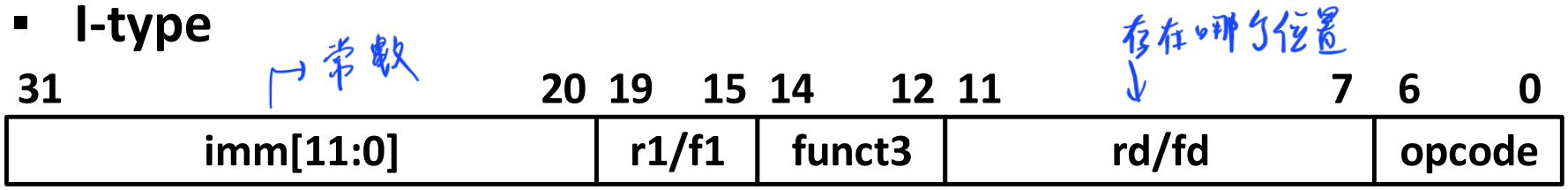


Instruction mapping

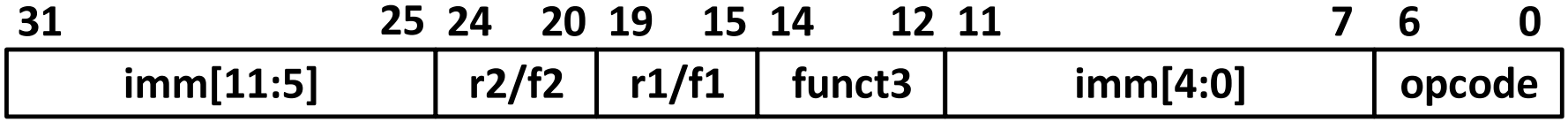
- R-type



- I-type

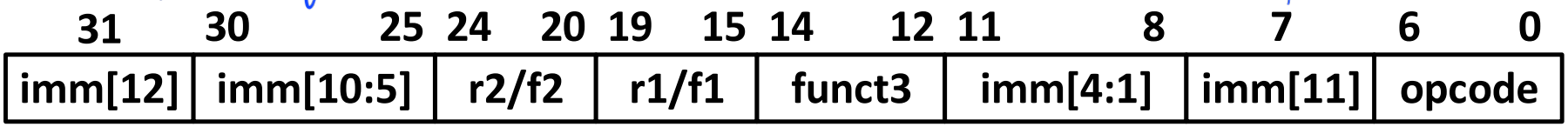


- S-type



Instruction mapping (cont'd)

- B-type



- EOF → 停止 program

tb 看到就 cut 掉

4、5 可以持 1 or 多个 cycle

$0 可被寫入

只有一開始會 reset (依 template)

Instruction

integer (no fixed point)

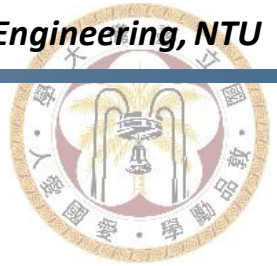
Operation	Assemble	Type	Meaning	Note
Add	add	R	$rd = $r1 + $r2 → 2's complement	Signed Operation
Subtract	sub	R	$rd = $r1 - $r2	Signed Operation
Add immediate	addi	I	$rd = $r1 + im	Signed Operation
Load word	lw	I	$rd = Mem[$r1 + im]	Signed Operation
Store word	sw	S	Mem[$r1 + im] = $r2	Signed Operation
Branch on equal	beq	B	if($r1==$r2), $pc = $pc + im; else, $pc = $pc + 4 → like other ops	PC-relative Signed Operation
Branch less than	blt	B	if($r1<$r2), $pc = $pc + im; else, $pc = $pc + 4	PC-relative Signed Operation
Set on less than	slt	R	if($r1<$r2), $rd = 1; else, $rd = 0	Signed Operation
Shift left logical	sll	R	$rd = $r1 << $r2	Unsigned Operation
Shift right logical	srl	R	$rd = $r1 >> $r2	Unsigned Operation

Instruction (cont'd)

Operation	Assemble	Type	Meaning	Note
Floating-point add	fadd	R	$fd = $f1 + $f2	Floating-point Operation
Floating-point substract	fsub	R	$fd = $f1 - $f2	Floating-point Operation
Load floating-point	flw	I	$fd = Mem[$r1 + im]	Signed Operation
Store floating-point	fsw	S	Mem[$r1 + im] = $f2	Signed Operation
Floating-point classify	fclass	R	Classify floating-point format	Floating-point Operation
Floating-point set less than	flt	R	if($f1<$f2), $rd = 1; else, $rd = 0	Floating-point Operation
End of File	eof	EOF	Stop processing	Last instruction in the pattern

Note: The notation of **`im`** in I-type 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**.



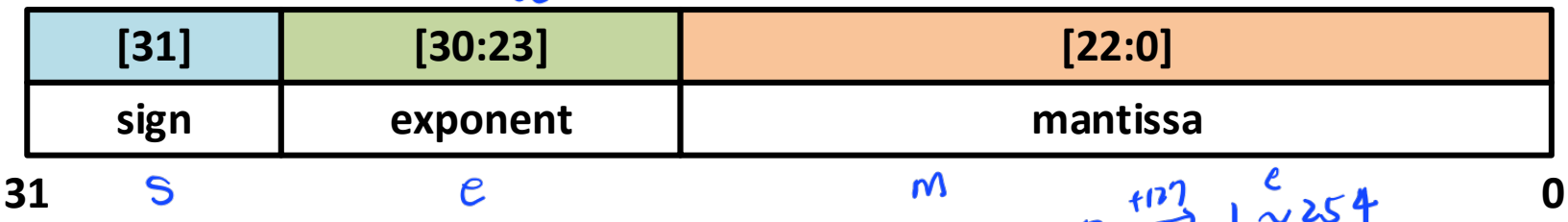
Floating Point

- For instructions **`fadd, fsub, fclass, flt`**, 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]	[30:23]	[22:0]
sign	exponent	mantissa

31 0

IEEE-754 Single Precision Format



Single-Format Bit Pattern	Value
0 < e < 255	$(-1)^s \times 2^{e-127} \times 1.m$ (normal numbers)
e = 0; m ≠ 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 ≠ 0 (at least one bit in f is nonzero)	NaN (Not-a-Number)

There are special rules for adding or subtracting signed zero:

- $x + (\pm 0) = x$ (for x different from 0)
- $(-0) + (-0) = (-0) - (+0) = -0$
- $(+0) + (+0) = (+0) - (-0) = +0$
- $x - x = x + (-x) = +0$ (for any finite x, −0 when rounding toward negative)

$x-(\pm 0)=x$] 不用額外處理?

$-0 + -0 \Rightarrow -0$

$-0 - +0 \Rightarrow -0$

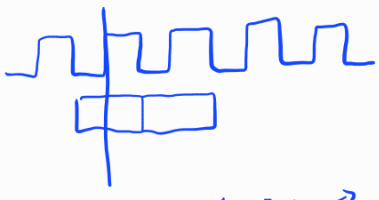
others $\Rightarrow +0$

} but this HW all result to (+0)

```cpp
#include <iostream>

int main() {
    std::cout << "+0.0 + +0.0 == " << +0.0 + +0.0 << std::endl;
    std::cout << "+0.0 + -0.0 == " << +0.0 + -0.0 << std::endl;
    std::cout << "-0.0 + +0.0 == " << -0.0 + +0.0 << std::endl;
    std::cout << "-0.0 + -0.0 == " << -0.0 + -0.0 << std::endl;
    return 0;
}
```

Output:

```
+0.0 + +0.0 == 0
+0.0 + -0.0 == 0
-0.0 + +0.0 == 0
-0.0 + -0.0 == -0
```



add 1 unsign to sign → 5 or 2 bit

sign_sum 3 . 23 253

unsign_sum 3 . 23 253

normalized unsignsum 1 . 23 253

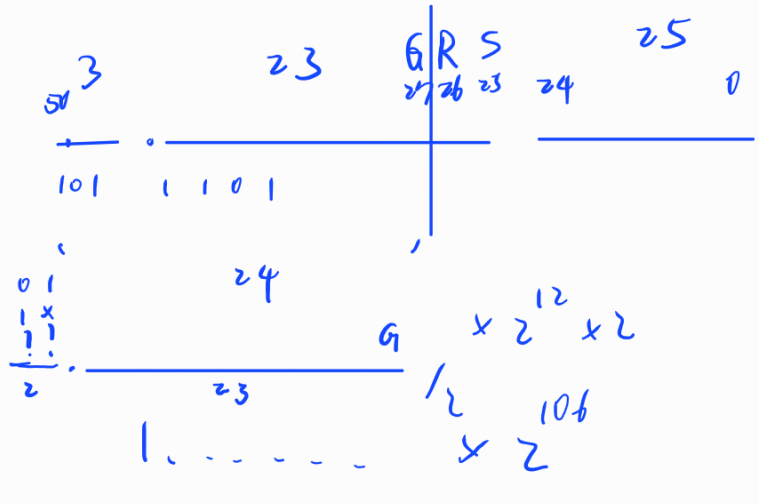
GRS { 0 1 . 2 21 GRS

round mantissa 2 . 23

3 23 253

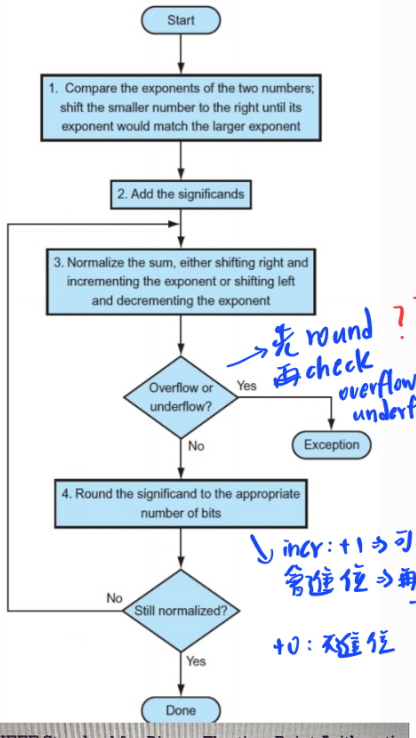
27 27

0 1



1 . 23 G R S 25 24 25 0

1 25



先 round 再 check ?!

overflow or underflow

↓ incr: +1 ⇒ 可能會進位 ⇒ 再 norm 一次

+0：不進位

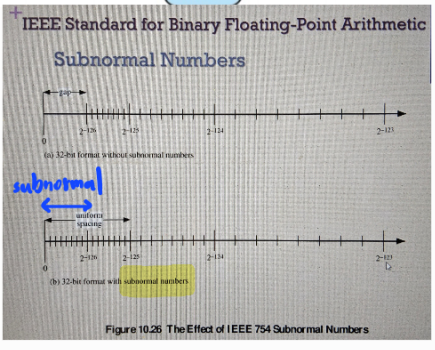
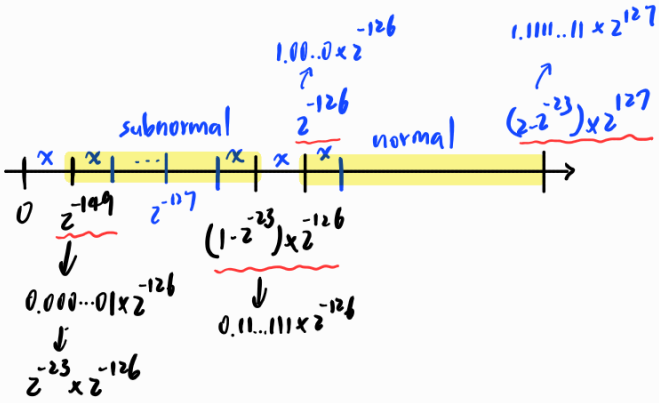


Figure 10.26 The Effect of IEEE 754 Subnormal Numbers



smallest result value: $X = 2^{-149}$

biggest result value: $(2-2^{-23})\times 2^{127} + (2-2^{-23})\times 2^{127} = (2\times 2^{-23})\times 2^{128}$

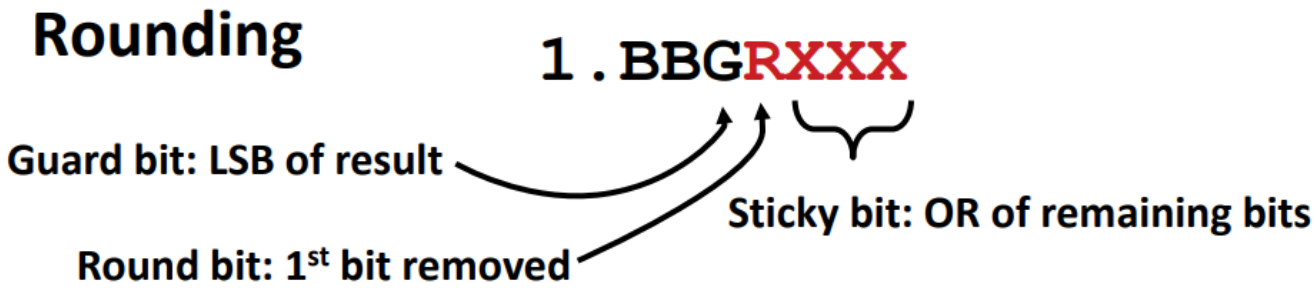
$1.11...111\times 2^{128}$

underflow	subnormal / normal		overflow
-149	-126	~ 127	128
	↓ bias 127		
-22	1	~ 254	255
	↓ bias 128		
106	129	~ 382	383

bias 255

Round to Nearest Even

- For instructions **`fadd`**, **`fsub`**, you will have to round the mantissa with **round to nearest even** [3]



- **Round up conditions** 進位
  - Round = 1, Sticky = 1 → > 0.5
  - Guard = 1, Round = 1, Sticky = 0 → Round to even

Value	*Fraction*	*GRS*	*Incr?*	*Rounded*
128	1.0000000	000	N	1.000 ①
15	1.1010000	100	N	1.101 ①
17	1.0001000	010	N	1.000 ④
19	1.0011000	110	Y	1.010 ③
138	1.0001010	011	Y	1.001 ②
63	1.1111100	111	Y	10.000 ②

① R=0 不進位
② R=1 S=1 進
③ R=1 S=0 G=1 進
④ R=1 S=0 G=0 不進

Floating Point Classification

- For instruction **`fclass`**, you will have to classify the floating-point number stored in registers

Class	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	NaN

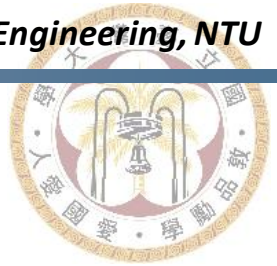
分類 floating-point

signed zero

Memory IP

- Size: 2048 × 32 bit
- i_add[12:2] for address mapping in memory
- Instructions are stored in address 0 - address 4095
- Data are should be write to address 4096 - address 8191

```
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
);
```

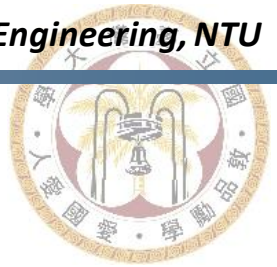


Status

- 6 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	INVALID_TYPE
3'd5	EOF_TYPE

有 invalid 產生

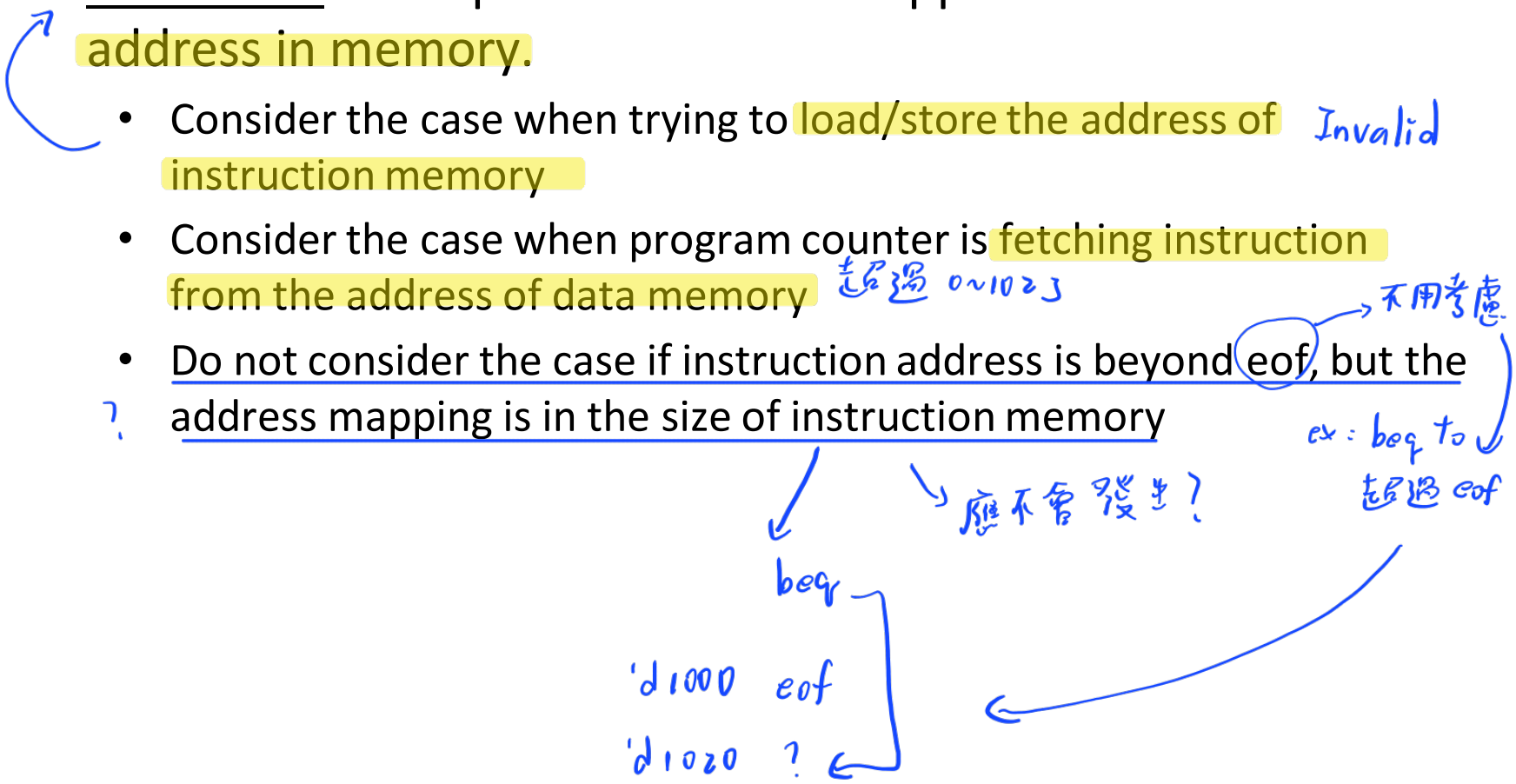


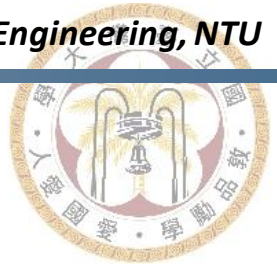
Invalid operation

- Invalid operation may happened.
  - **Situation1**: Overflow happened at integer arithmetic instructions (**`add`**, **`sub`**, **`addi`**)
  - **Situation2**: Infinite, NaN happened at floating-point arithmetic instructions (**`fadd`**, **`fsub`**, **`flt`**)
    - 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 happened.
  - **Situation3**: If output address are mapped to unknown address in memory.
    - Consider the case when trying to load/store the address of instruction memory
    - Consider the case when program counter is fetching instruction from the address of data memory
    - Do not consider the case if instruction address is beyond eof, but the address mapping is in the size of instruction memory



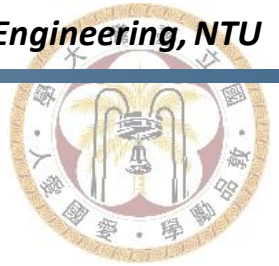


rtl.f

- Filelist

```
// ------------------------------------------------------------------------
// Simulation: HW2 simple RISC-V CPU
// ------------------------------------------------------------------------

// define files: Do not modify
// ------------------------------------------------------------------------
../00_TESTBED/define.v

// testbench: Do not modify
// ------------------------------------------------------------------------
../00_TESTBED/testbed.v
../00_TESTBED/data_mem.vp

// design files: Be free to add your design files
// ------------------------------------------------------------------------
./core.v
```

↑ add other files



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

如有其它 define：
放其它 file

- Do not modify

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

// opcode definition
`define OP_ADD     7'b0110011
`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_BNE     7'b1100011
`define OP_SLT     7'b0110011
`define OP_FADD    7'b1010011
`define OP_FSUB    7'b1010011
`define OP_FLW     7'b0000111
`define OP_FSW     7'b0100111
`define OP_FCLASS  7'b1010011
`define OP_FLT     7'b1010011
`define OP_ECALL   7'b1110011
```

```
// funct7 definition
`define FUNCT7_ADD     7'b0000000
`define FUNCT7_SUB     7'b0100000
`define FUNCT7_SLT     7'b0000000
`define FUNCT7_FADD    7'b0000000
`define FUNCT7_FSUB    7'b0000100
`define FUNCT7_FCLASS  7'b1110000
`define FUNCT7_FLT     7'b1010000

// funct3 definition
`define FUNCT3_ADD     3'b000
`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_SLT     3'b010
`define FUNCT3_FADD    3'b000
`define FUNCT3_FSUB    3'b000
`define FUNCT3_FLW     3'b010
`define FUNCT3_FSW     3'b010
`define FUNCT3_FCLASS  3'b000
`define FUNCT3_FEQ     3'b010
`define FUNCT3_FLT     3'b001
```

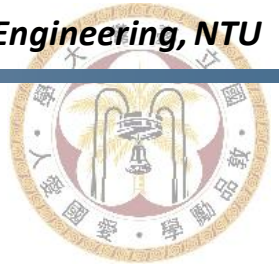
```
// floating class definition
`define FLOAT_NEG_INF      4'b0000
`define FLOAT_NEG_NORM     4'b0001
`define FLOAT_NEG_SUBNORM  4'b0010
`define FLOAT_NEG_ZERO     4'b0011
`define FLOAT_POS_ZERO     4'b0100
`define FLOAT_POS_SUBNORM  4'b0101
`define FLOAT_POS_NORM     4'b0110
`define FLOAT_POS_INF      4'b0111
`define FLOAT_NAN          4'b1000
```

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;
```

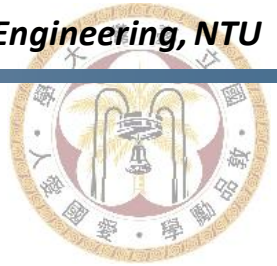
```
core u_core (
    .i_clk(),
    .i_rst_n(),
    .o_status(),
    .o_status_valid(),
    .o_we(),
    .o_addr(),
    .o_wdata(),
    .i_rdata()
);

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#\H8O&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(UXOSg2?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>FC6LaO\2g9Q+,Ad7fBF?
b6XA=:M7[_3COF+_59;H6E-Dfc7#U+&/A/A]WDWU>QUW.124=b>LE5EEO4f6J:W)
44Za5?:](CHHVagBN[2/dBWMJ?2NgZ6,WN^P[W@YaI+,0]=Yb_W+?5AK/\a>SBF-
Z6M;_KM/.e05RCFK+_M?\IJII8)@,@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()
P4OgM;E>@Y1V1d9/fIP7PN1:#ffG-FUS=@?bU9SE(>=^dL,;]DOOX0RU0OZKaX,\
@,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

grading command

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

- 99_clean_up

```
rm -rf *.history *.key *.log
rm -rf novas.rc novas.fsdb novas.conf
rm -rf INCA_libs nWaveLog BSSLib.lib++
```

PATTERN (1/2)

- Files in PATTERN are for your references

inst_assemble.dat

```
----------------------
R-type  $rd  $r1  $r2
I-type  $rd  $r1  im
S-type  $r2  $r1  im
B-type  $r1  $r2  im
----------------------
addi    $0    $0    1024
addi    $1    $0    32
addi    $3    $30   127
addi    $1    $31   -64
sub     $4    $2    $0
add     $5    $2    $3
beq     $5    $0    12
addi    $6    $8    2
sll     $0    $0    $6
sw      $4    $0    0
lw      $4    $0    64
slt     $2    $8    $10
add     $4    $3    $3
```

machine code →

inst.dat

```
01000000000000000000000000010011
00000010000000000000000010010011
00000111111111110000000110010011
11111100000011111000000010010011
01000000000000010000001000110011
00000000001100010000001010110011
00000000000000010100011000100011
00000000001001000000001100010011
00000000011000000001000000110011
00000000010000000010000000100011
00000100000000000010001000000011
00000000101001000010000100110011
00000000001100011000001000110011
00000000010000101010010000110011
00000000000010000101000111010011
00000000010000000010011000100011
00000000010000000010001110000011
```

PATTERN (2/2)

- Data in data.dat includes golden data in instruction memory and data memory
- You can compare the data in memory with the golden data directly

→ 直接比全部 即可

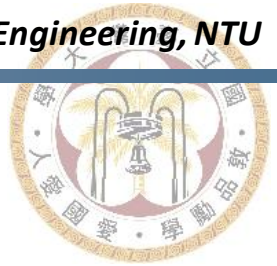
in mem_r[0]

status.dat

```
001
001
001
001
000
000
011
001
000
010
001
000
000
000
000
010
001
```

data.dat → 有 inst & data mem

```
01000000000000000000000000010011
00000010000000000000000010010011
00000111111111110000000110010011
11111100000011111000000010010011
01000000000000010000001000110011
00000000001100010000001010110011
00000000000000101000110001100011
00000000001001000000001100010011
00000000011000000001000000110011
00000000010000000010000000100011
00000100000000000010001000000011
00000000101001000010000100110011
00000000001100011000001000110011
00000000010000101010010000110011
00000000000100001010001110110011
00000000010000000010011000100011
00000000010000000010001110000011
```



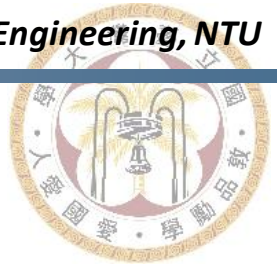
Grading Policy

- TA will run your code with following command

```
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
  - Hidden pattern: **30%** (20 patterns in total)
    - **2%** for each test (data & status both correct)

最多拿到 30%



Grading Policy

Deadline: 2024/10/15 13:59:59 (UTC+8)

- **No late submission is allowed**
  - Any submissions after the deadline will receive 0 points
- **5-point deduction** for incorrect naming or format
  - Pack all files into a single folder and compress the folder
  - Ensure that the files submitted can be decompressed and executed without issues
- **No plagiarism**
  - Plagiarism in any form, including copying from online sources, is strictly prohibited

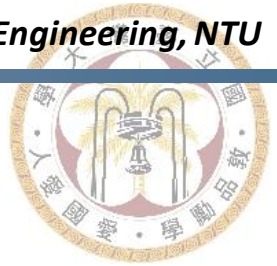
Submission

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

```
r11943133_hw2/
└── 01_RTL
    ├── core.v
    ├── rtl.f
    └── (other design files)
```

- Compress the folder **studentID_hw2** in a tar file named **studentID_hw2_v*k*.tar** (*k* is the number of version, *k* =1,2,…)
  - Use lower case for student ID. (Ex. r11943133_hw2_v1.tar)

- Submit to NTU Cool



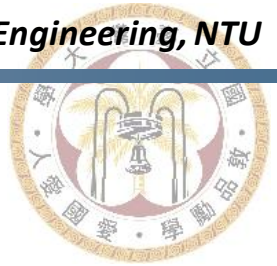
Hint

- 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:** r11943133@ntu.edu.tw
  - Title should start with **[CVSD 2024 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

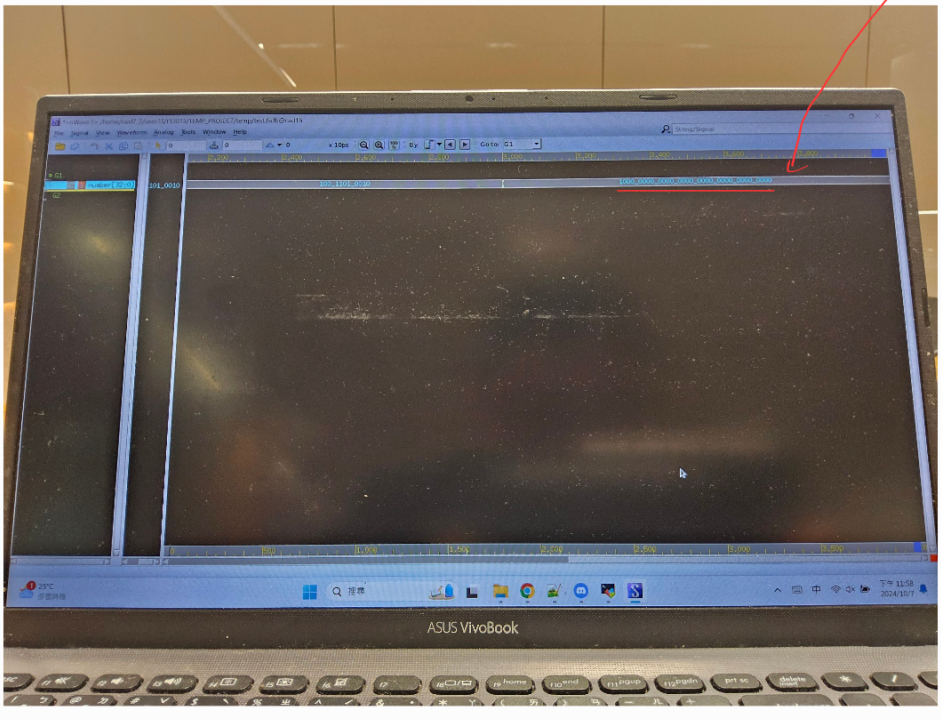
problem:

有多个 module 時，自己 module 取名用 module name 開頭 ex: alu_operand_1

```
wire signed [7:0]out;

assign out = a + b;


reg signed [32:0]number;
reg signed[32:0] upperbound_32bit, lowerbound_32bit;
reg[1:0] result, IsOverflow;
reg signed [31:0]lower_bound_32b;
initial begin
    // VER 1
    number = 1234;
    lower_bound_32b = -2147483648;
    result = (number < lower_bound_32b);
    $display("VER 1 result = %d", result);

    // VER 2
    #10;
    number = 1234;
    result = (number < -2147483648);
    $display("VER 2 result = %d", result);

    // VER 3
    #10;
    number = 1234;
    result = (number < -2147483647);
    $display("VER 3 result = %d", result);

    // ver 4
    #10;
    number = -2147483648;
    upperbound_32bit = 33'b0_0111_1111_1111_1111_1111_1111_1111_1111; // `define UPPERBOUND_32BIT 2147483647
    lowerbound_32bit = 33'b1_1000_0000_0000_0000_0000_0000_0000_0000; // `define LOWERBOUND_32BIT -2147483648
```

lower bound of 32 bit

no sign extention => wrong



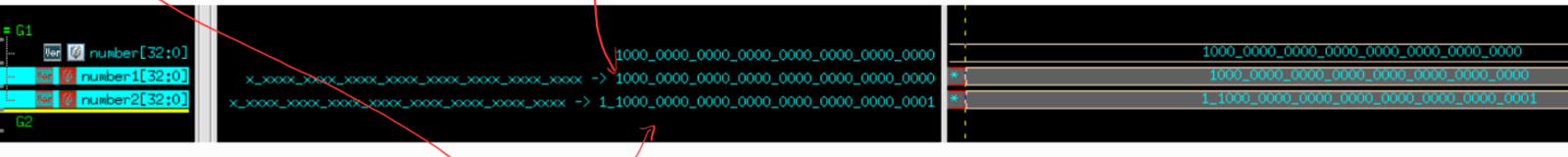
```
reg signed [32:0]number;
reg signed [32:0]number1, number2;
reg signed[32:0] upperbound_32bit, lowerbound_32bit;
reg[1:0] result, IsOverflow;
reg signed [31:0]lower_bound_32b;
initial begin
    // VER 1
    number = 1234;
    lower_bound_32b = -2147483648;
    result = (number < lower_bound_32b);
    $display("VER 1 result = %d", result);

    // VER 2
    #10;
    number = 1234;
    result = (number < -2147483648);
    $display("VER 2 result = %d", result);

    // VER 3
    #10;
    number = 1234;
    result = (number < -2147483647);
    $display("VER 3 result = %d", result);

    // ver 4
    # 10;
    number = -2147483648;
    upperbound_32bit = 33'b0_0111_1111_1111_1111_1111_1111_1111_1111; // `define UPPERBOUND_32BIT 2147483647
    lowerbound_32bit = 33'b1_1000_0000_0000_0000_0000_0000_0000_0000; // `define LOWERBOUND_32BIT -2147483648
    if(number > upperbound_32bit) IsOverflow = 1;
    else if(number < lowerbound_32bit) IsOverflow = 2;
    else IsOverflow = 3;
    $display("VER 4 IsOverflow = %d", IsOverflow);

    #10;
    number1 = -2147483648;
    number2 = -2147483647;

    #10;
```

if 用 decimal constant，用 number = 33'b1_1000_0...0 => it's ok

用 constant 時，若要用 32 bit

(-2^{31}) -2147483648 ~ $(2^{31}-1)$ 2147483647

-2147483647 → 用此 range



wrong, no sign-ext

correct: with sign-ext