Lab 7
CPU Test & Debug

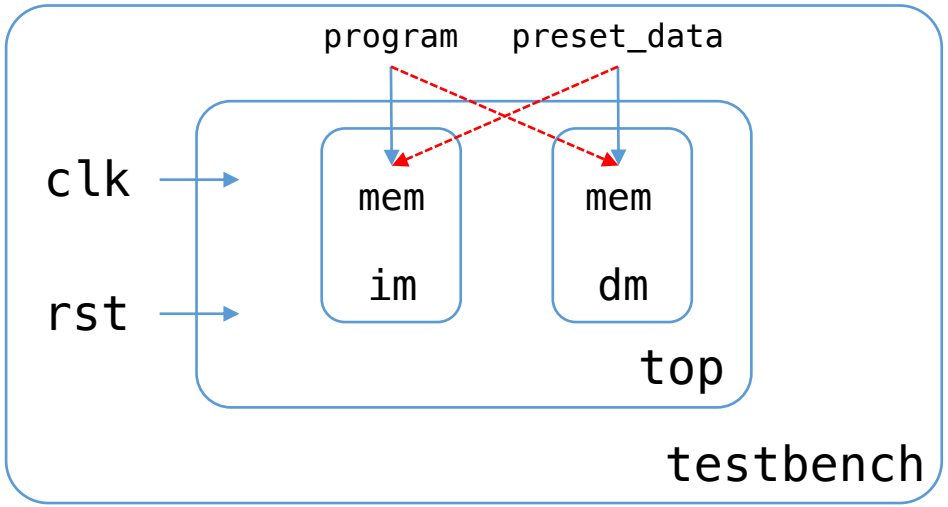
Video Link :

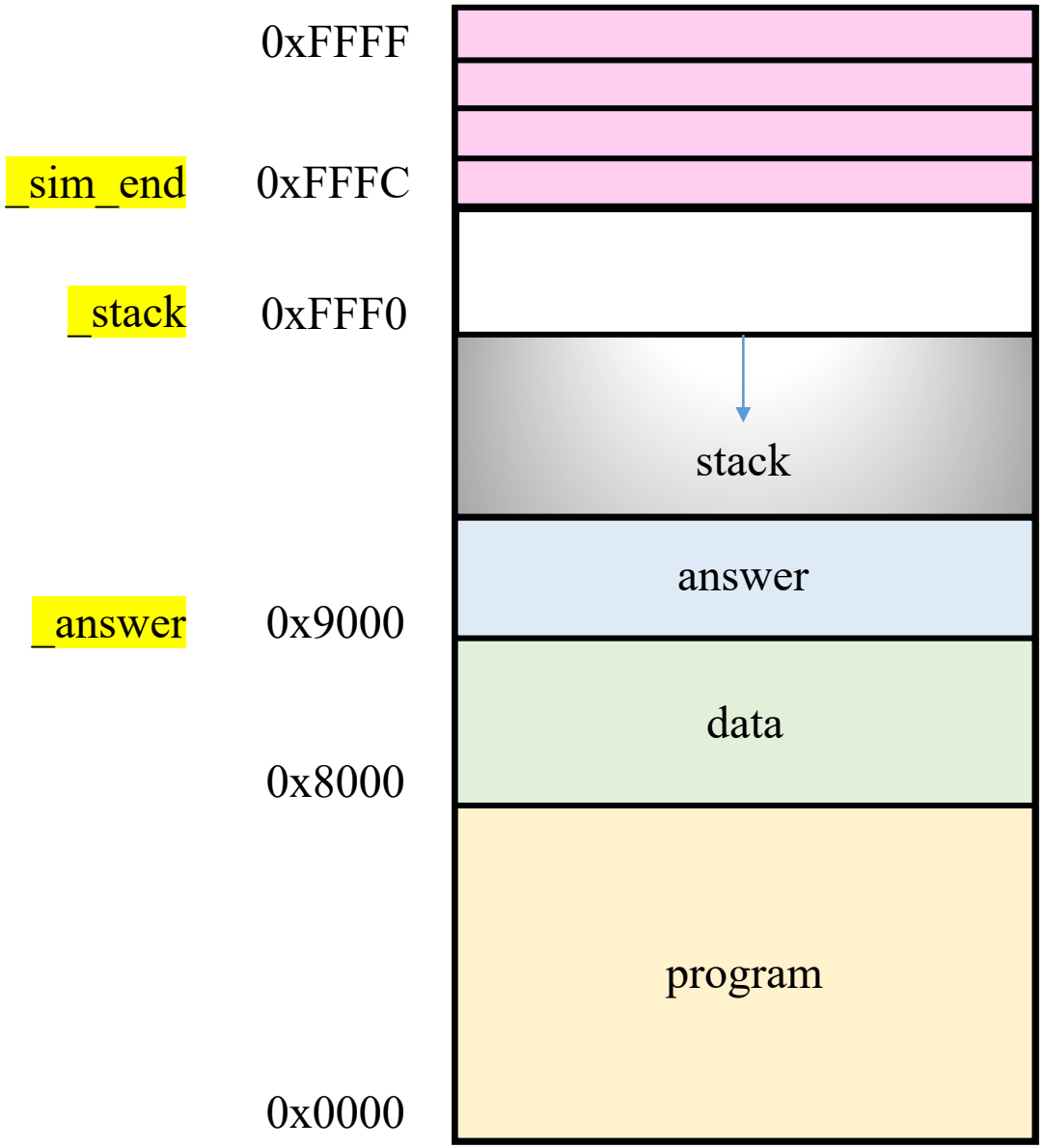
Outline

1. File Structure
2. TestBench
3. How to debug with waveform
4. Generate .hex file
5. Summary

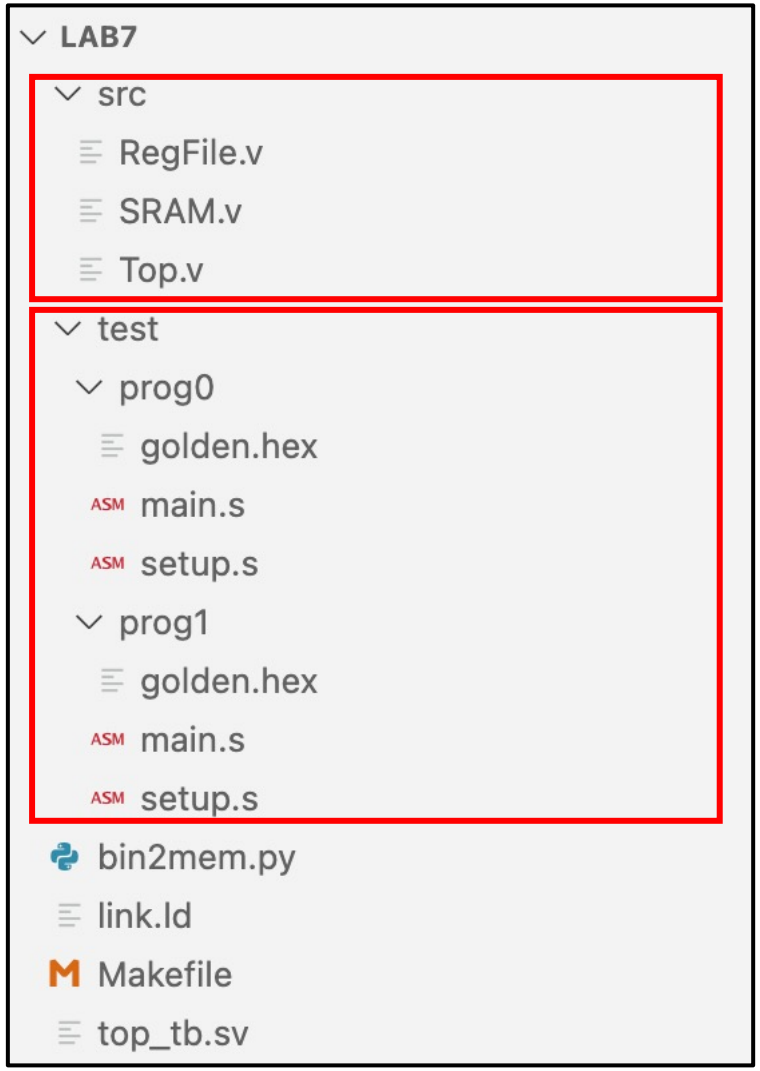
How to test CPU

current_pc reset from 32'd0



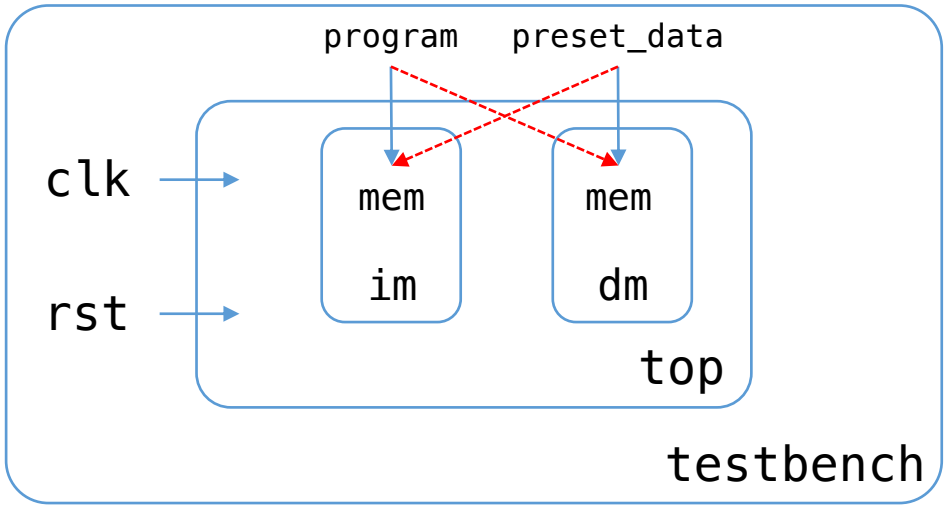


File Structure



- **src** : All source files of your CPU
- **test** : The programs to test your CPU
- **Makefile** : A script used to convert the test program from $.s \rightarrow .mem$
  We load .hex into the mem of im & dm
  $(*.s \rightarrow *.o \rightarrow .elf \rightarrow .hex(byte) \rightarrow .mem(word))$
  $(.elf \rightarrow .dump(disassemble, used\ to\ debug))$
- **link.ld** : A link script used to link $*.o \rightarrow .elf$
- **bin2mem.py** : A python program used to convert the test program from $.hex \rightarrow .mem$
- **top_tb.sv** : Set the test environment, run the CPU and compare the results with golden data

TestBench (top_tb.sv)




```
6   `define CYCLE 10.0    // Cycle time
7   `define MAX 100000    // Max cycle number
8   `define prog_path "./test/prog0/main.hex"
9   `define gold_path "./test/prog0/golden.hex"
19  `include "./src/Top.v"
```

1. Load .hex into the mem of im & dm

```
46  // Load program and preset data to im & dm
47  $readmemh(`prog_path, top.im.mem);
48  $readmemh(`prog_path, top.dm.mem);
```

2. Initialize some registers & memory

```
58  // Initialize register[0] = 0 (hardwire to ground)
59  top.regfile.registers[0] = 32'd0;
```

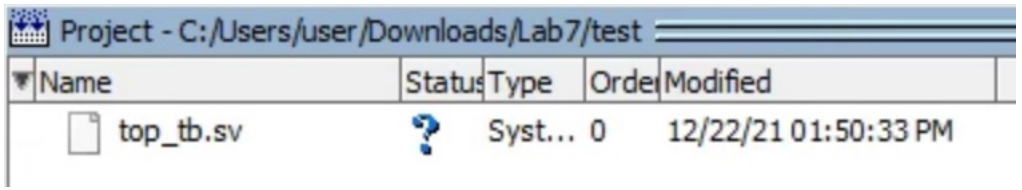
3. Load Golden Data
4. Wait until mem[0xFFFC] == 8'hFF (end of execution)

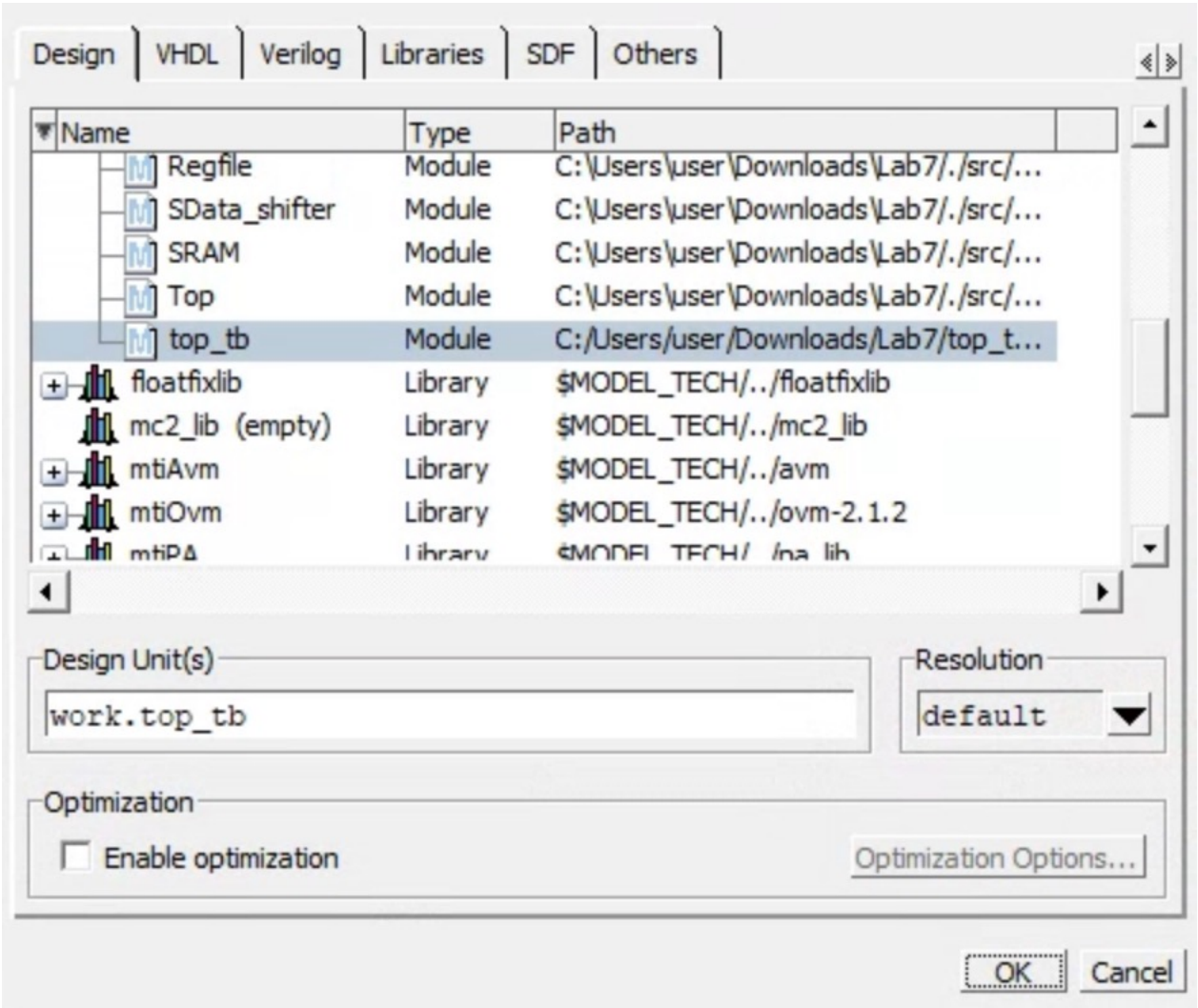
```
56  `mem_word('hfffc) = 32'd0;
```

```
72  // Wait until end of execution
73  wait(top.dm.mem[16'hfffc] == 8'hff);
74  $display("\nDone\n");
```

5. Compare the results with Golden Data

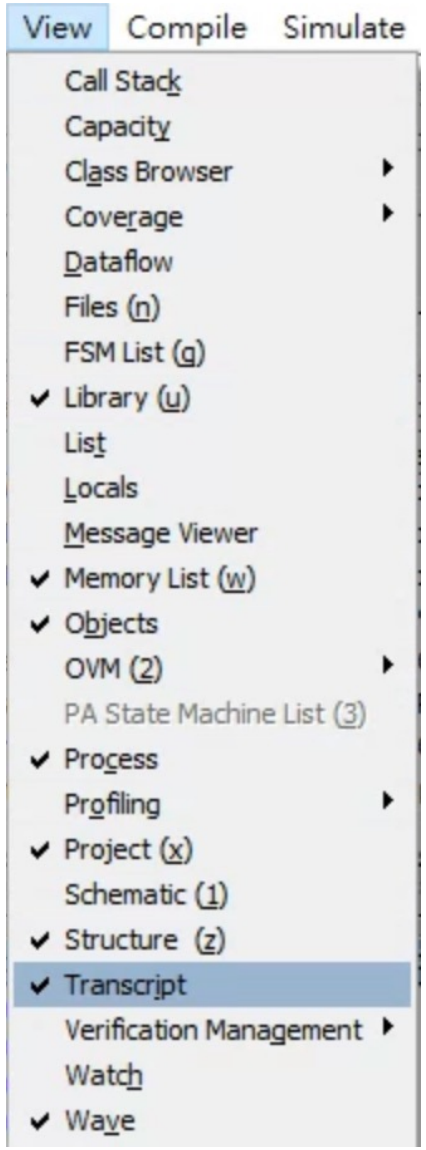
ModelSim

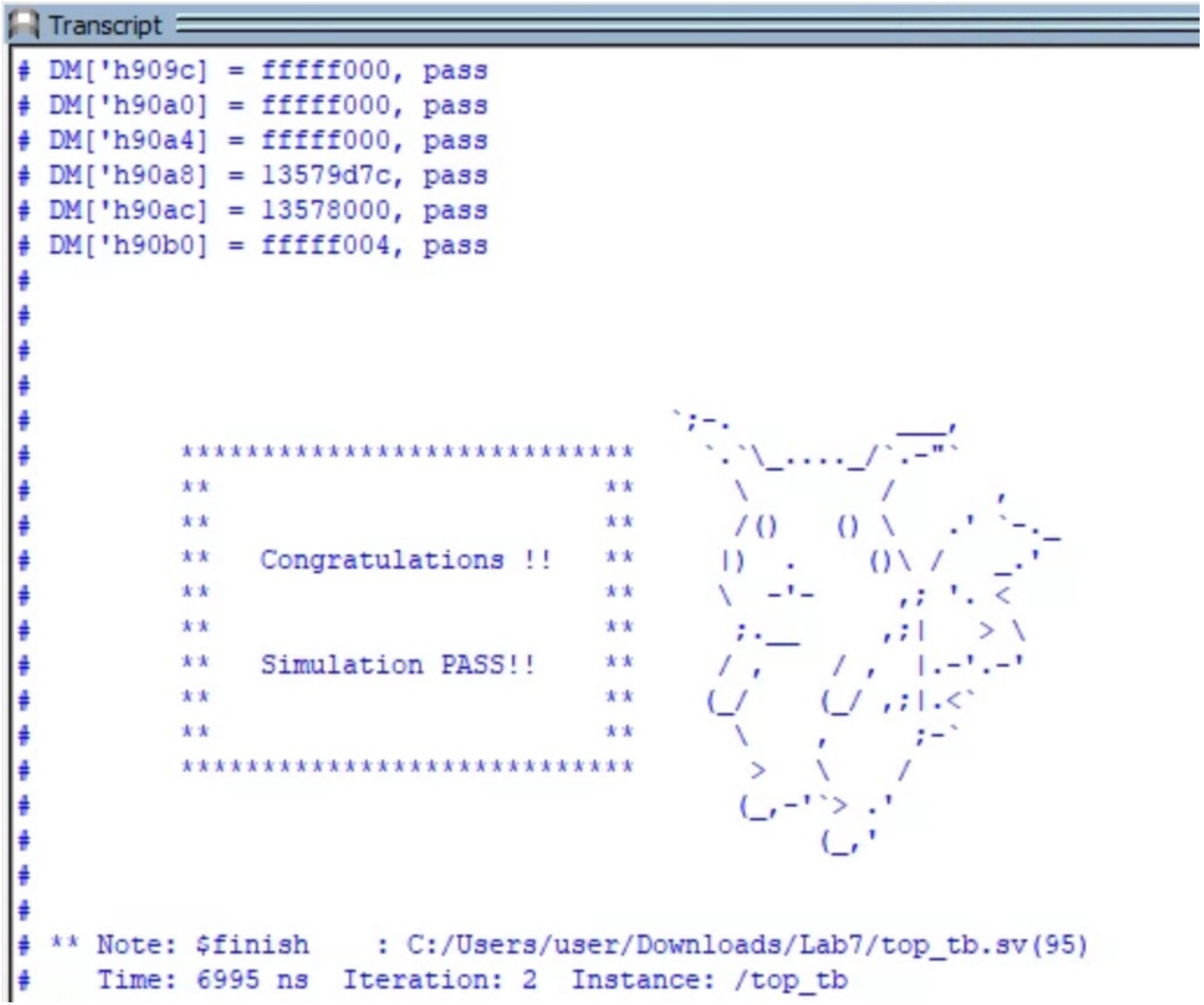

Project - C:/Users/user/Downloads/Lab7/test
Name
Status
Type
Order
Modified
top_tb.sv
Syst...
0
12/22/21 01:50:33 PM


Design
VHDL
Verilog
Libraries
SDF
Others
Name
Type
Path
Regfile
Module
C:\Users\user\Downloads\Lab7/./src/...
SData_shifter
Module
C:\Users\user\Downloads\Lab7/./src/...
SRAM
Module
C:\Users\user\Downloads\Lab7/./src/...
Top
Module
C:\Users\user\Downloads\Lab7/./src/...
top_tb
Module
C:/Users/user/Downloads/Lab7/top_t...
floatfixlib
Library
$MODEL_TECH/../floatfixlib
mc2_lib (empty)
Library
$MODEL_TECH/../mc2_lib
mtiAvm
Library
$MODEL_TECH/../avm
mtiOvm
Library
$MODEL_TECH/../ovm-2.1.2
Design Unit(s)
work.top_tb
Resolution
default
Optimization
Enable optimization
Optimization Options...
OK
Cancel

ModelSim – Transcript

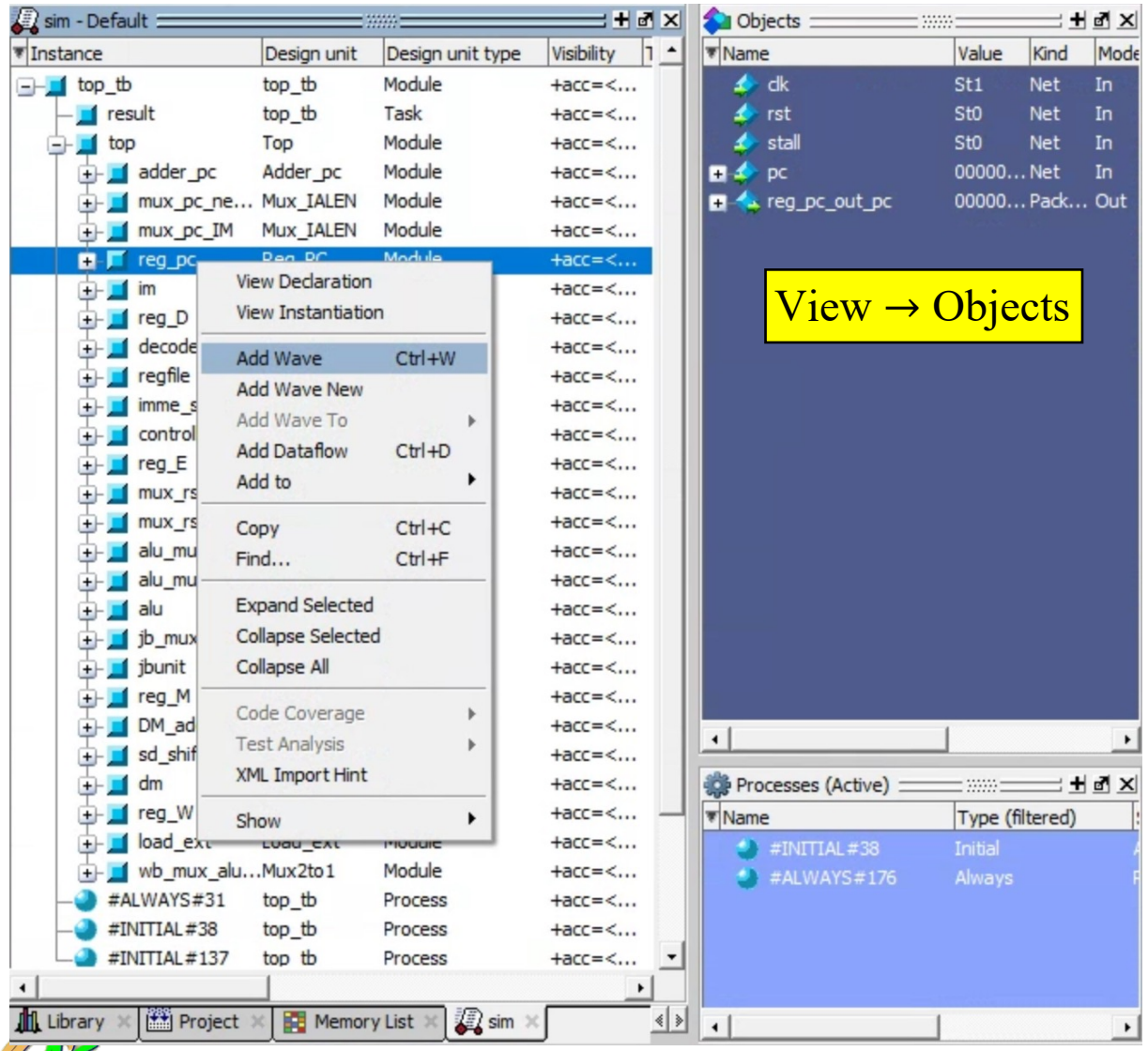
View → Transcript





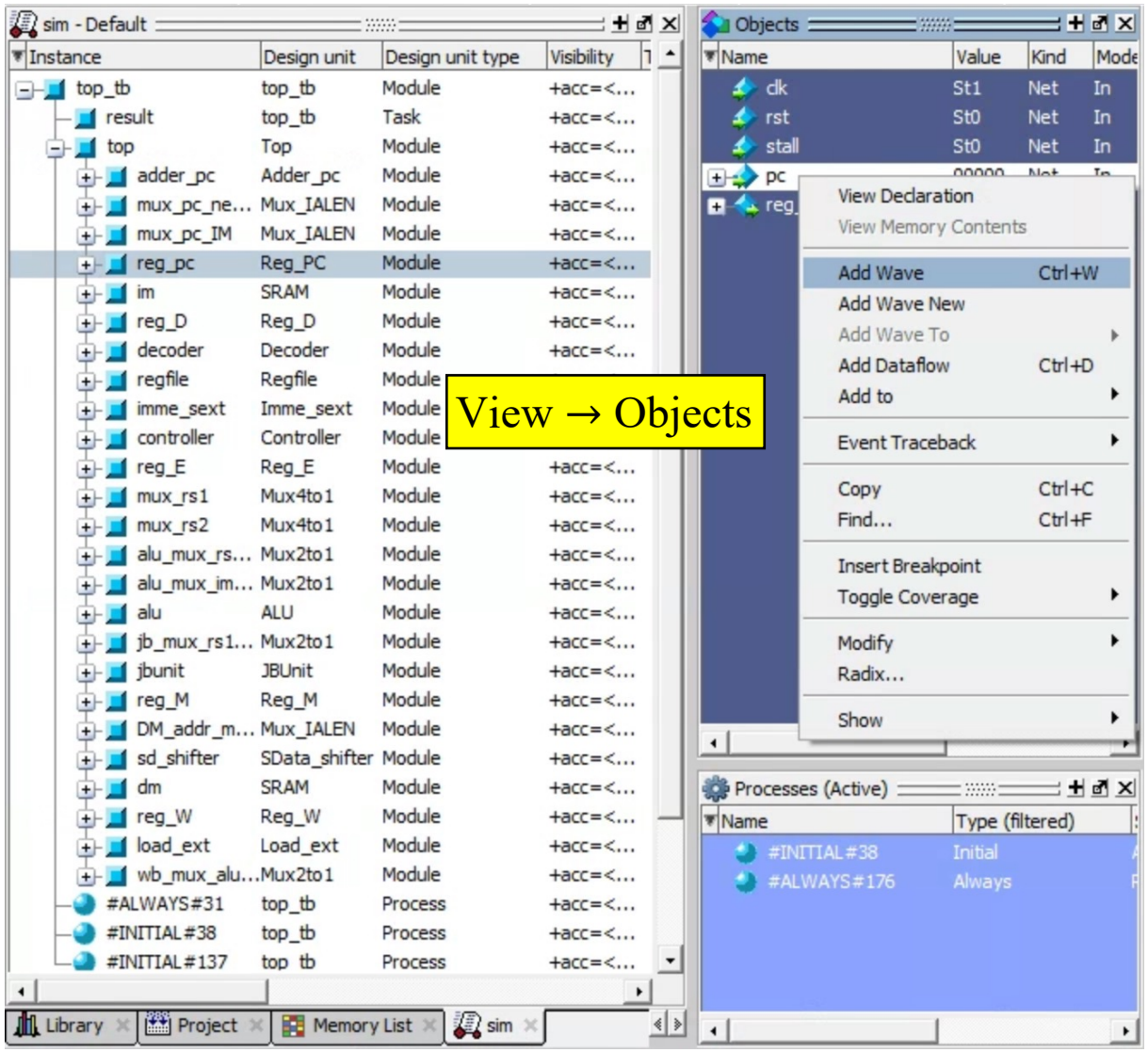
ModelSim – Add wave

Add almost all signals in this module



View → Objects

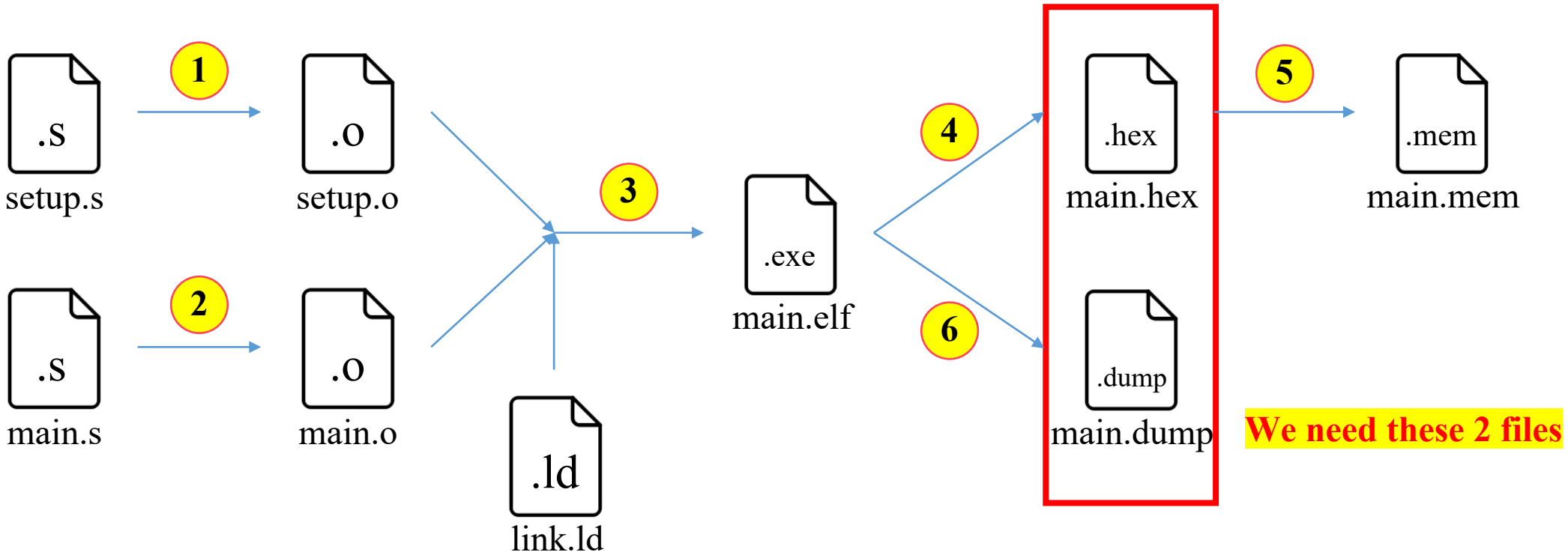
Add specific signal in this module



View → Objects

Generate .hex file

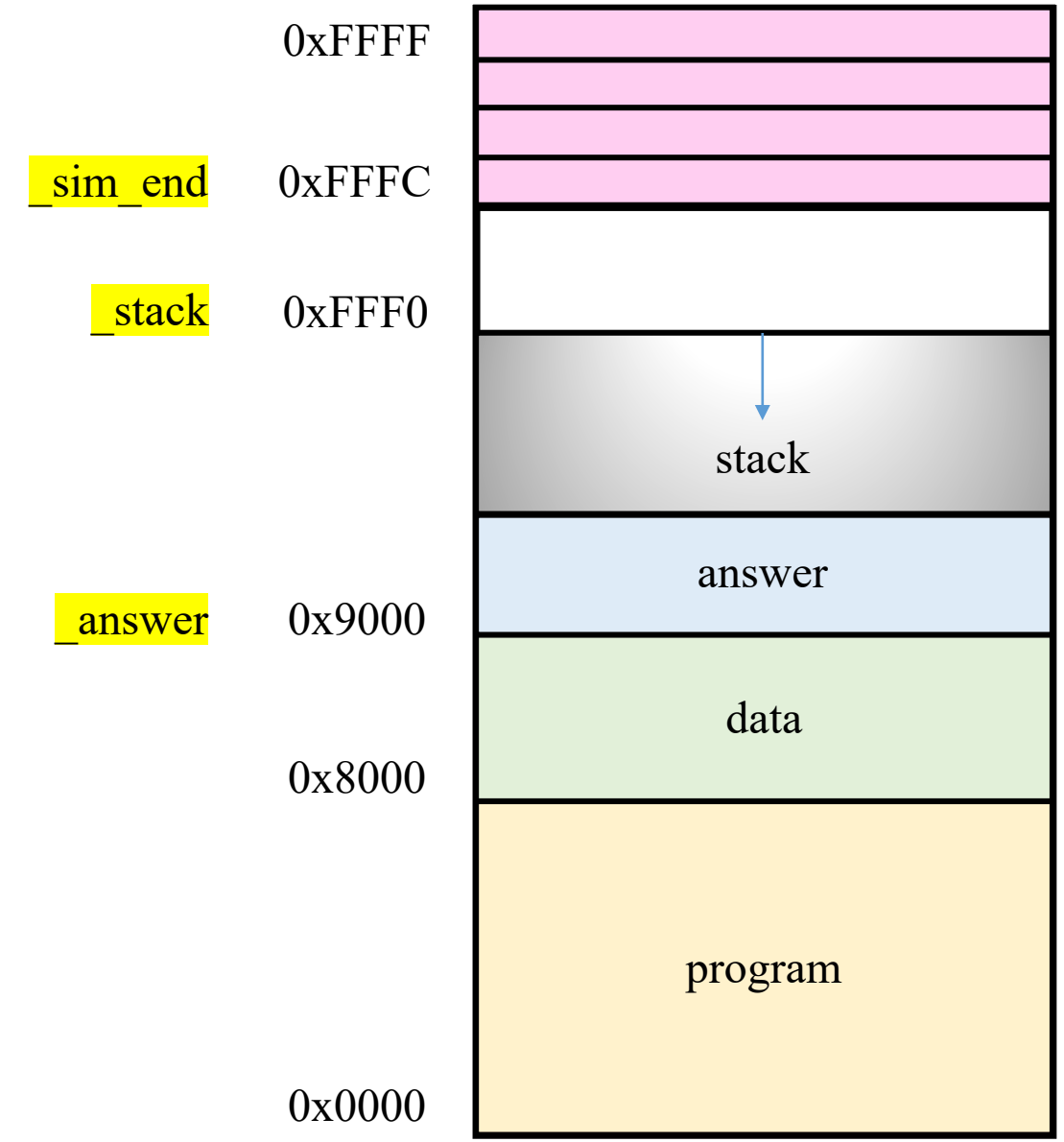
Generate .hex from .s



1. riscv32-unknown-elf-as -march=rv32i -mabi=ilp32 ./test/prog0/setup.s -o ./test/prog0/setup.o
2. riscv32-unknown-elf-as -march=rv32i -mabi=ilp32 ./test/prog0/main.s -o ./test/prog0/main.o
3. riscv32-unknown-elf-ld -b elf32-littleriscv -T link.ld ./test/prog0/setup.o ./test/prog0/main.o -o ./test/prog0/main.elf
4. riscv32-unknown-elf-objcopy -O verilog ./test/prog0/main.elf ./test/prog0/main.hex
5. python3 bin2mem.py --bin ./test/prog0/main.hex
6. riscv32-unknown-elf-objdump -xsd ./test/prog0/main.elf > ./test/prog0/main.dump

Main memory (link.ld)

```
1   OUTPUT_ARCH( "riscv" )
2
3   SECTIONS
4   {
5       . = 0x0000;
6       .text : { *(.text) }
7
8       . = 0x8000;
9       .data : { *(.data) }
10
11      . = 0x9000;
12      _answer = .;
13
14      . = 0xfff0;
15      _stack = .;
16
17      . = 0xfffc;
18      _sim_end = .;
19
20  }
```



Program Structure

Little endian

The order is important

```
$ riscv32-unknown-elf-ld -b elf32-littleriscv -T link.ld ./test/prog0/setup.o ./test/prog0/main.o -o ./test/prog0/main.elf
```

```
1   .text
2
3   _start:
4
5   init_stack:
6     # set stack pointer
7     la sp, _stack
8
9   SystemInit:
10    # jump to main
11    jal main
12
13  SystemExit:
14    # End simulation
15    # Write -1 at _sim_end(0xfffc)
16    la t0, _sim_end
17    li t1, -1
18    sw t1, 0(t0)
19
20  dead_loop:
21    # infinite loop
22    j dead_loop
```

```
1   .data
2   # ...
3
4   .text
5   .globl main
6
7   main:
8     addi sp, sp, -4
9     sw s0, 0(sp)
10    la s0, _answer
11
12    # ...
13
14
15  main_exit:
16    /* Simulation End */
17    lw s0, 0(sp)
18    addi sp, sp, 4
19    ret
```

```
243  Disassembly of section .text:
244
245  00000000 <_start>:
246      0: 00010117            auip
247      4: ff010113            addi
248
249  00000008 <SystemInit>:
250      8: 018000ef            jal
251
252  0000000c <SystemExit>:
253      c: 00010297            auip
254     10: ff028293            addi
255     14: fff00313            li
256     18: 0062a023            sw
257
258  0000001c <dead_loop>:
259     1c: 0000006f            j 1c
260
261  00000020 <main>:
262     20: ffc10113            addi
263     24: 00812023            sw
264     28: 00009417            auip
265     2c: fd840413            addi
```

Get rid of OS info, Only Preserve the memory info

```
$ riscv32-unknown-elf-objcopy -O verilog ./test/prog0/main.elf ./test/prog0/main.hex
```

- Get rid of OS information
- Only Preserve the memory info

```
Disassembly of section .text:

00000000 <_start>:
   0: 00010117            auipc sp,0x10
   4: ff010113            addi  sp,sp,-16 # fff0 <_stack>

00000008 <SystemInit>:
   8: 018000ef            jal ra,20 <main>

0000000c <SystemExit>:
   c: 00010297            auipc t0,0x10
  10: ff028293            addi  t0,t0,-16 # fffc <_sim_end>
  14: fff00313            li  t1,-1
  18: 0062a023            sw  t1,0(t0)

0000001c <dead_loop>:
  1c: 0000006f            j 1c <dead_loop>

00000020 <main>:
  20: ffc10113            addi  sp,sp,-4
  24: 00812023            sw  s0,0(sp)
  28: 00009417            auipc s0,0x9
  2c: fd840413            addi  s0,s0,-40 # 9000 <_answer>
```

```
1  @00000000
2  17 01 01 00 13 01 01 FF EF 00 80 01 97 02 01 00
3  93 82 02 FF 13 03 F0 FF 23 A0 62 00 6F 00 00 00
4  13 01 C1 FF 23 20 81 00 17 94 00 00 13 04 84 FD
5  93 02 F0 FF 13 03 F0 FF B3 82 62 00 B3 82 62 00
6  B3 82 62 00 B3 82 62 00 B3 82 62 00 13 03 E0 FF
7  B3 02 53 00 B3 02 53 00 B3 02 53 00 B3 02 53 00
8  B3 02 53 00 23 20 54 00 13 04 44 00 93 02 00 00
9  13 03 F0 FF B3 82 62 40 B3 82 62 40 B3 82 62 40
10 B3 82 62 40 B3 82 62 40 13 03 D0 FF B3 02 53 40
11 B3 02 53 40 B3 02 53 40 B3 02 53 40 B3 02 53 40
12 23 20 54 00 13 04 44 00 93 02 10 00 13 03 10 00
13 B3 92 62 00 B3 92 62 00 B3 92 62 00 B3 92 62 00
14 B3 92 62 00 13 03 20 00 B3 12 53 00 B3 12 53 00
15 B3 12 53 00 B3 12 53 00 B3 12 53 00 23 20 54 00
16 13 04 44 00 93 02 F0 FF 13 03 10 00 B3 A2 62 00
17 B3 A2 62 00 B3 A2 62 00 B3 A2 62 00 B3 A2 62 00
18 13 03 F0 FF B3 22 53 00 B3 22 53 00 B3 22 53 00
```

(Optional) Convert byte to word

.hex
(byte)
Liitle Endian

```
1   @00000000
2   17 01 01 00 13 01 01 FF EF 00 80 01 97 02 01 00
3   93 82 02 FF 13 03 F0 FF 23 A0 62 00 6F 00 00 00
4   13 01 C1 FF 23 20 81 00 17 94 00 00 13 04 84 FD
5   93 02 F0 FF 13 03 F0 FF B3 82 62 00 B3 82 62 00
6   B3 82 62 00 B3 82 62 00 B3 82 62 00 13 03 E0 FF
7   B3 02 53 00 B3 02 53 00 B3 02 53 00 B3 02 53 00
8   B3 02 53 00 23 20 54 00 13 04 44 00 93 02 00 00
9   13 03 F0 FF B3 82 62 40 B3 82 62 40 B3 82 62 40
10  B3 82 62 40 B3 82 62 40 13 03 D0 FF B3 02 53 40
11  B3 02 53 40 B3 02 53 40 B3 02 53 40 B3 02 53 40
12  23 20 54 00 13 04 44 00 93 02 10 00 13 03 10 00
13  B3 92 62 00 B3 92 62 00 B3 92 62 00 B3 92 62 00
14  B3 92 62 00 13 03 20 00 B3 12 53 00 B3 12 53 00
15  B3 12 53 00 B3 12 53 00 B3 12 53 00 23 20 54 00
16  13 04 44 00 93 02 F0 FF 13 03 10 00 B3 A2 62 00
17  B3 A2 62 00 B3 A2 62 00 B3 A2 62 00 B3 A2 62 00
18  13 03 F0 FF B3 22 53 00 B3 22 53 00 B3 22 53 00
```

.mem
(word)

```
1   @00000000
2   00010117
3   FF010113
4   018000EF
5   00010297
6   FF028293
7   FFF00313
8   0062A023
9   0000006F
10  FFC10113
11  00812023
12  00009417
13  FD840413
14  FFF00293
15  FFF00313
16  006282B3
17  006282B3
18  006282B3
```

.dump

```
243   Disassembly of section .text:
244
245   00000000 <_start>:
246       0: 00010117           auipc sp,0x10
247       4: ff010113           addi  sp,sp,-16 # fff0 <_stack>
248
249   00000008 <SystemInit>:
250       8: 018000ef           jal ra,20 <main>
251
252   0000000c <SystemExit>:
253       c: 00010297           auipc t0,0x10
254      10: ff028293           addi  t0,t0,-16 # fffc <_sim_end>
255      14: fff00313           li  t1,-1
256      18: 0062a023           sw  t1,0(t0)
257
258   0000001c <dead_loop>:
259      1c: 0000006f           j 1c <dead_loop>
260
261   00000020 <main>:
262      20: ffc10113           addi  sp,sp,-4
263      24: 00812023           sw  s0,0(sp)
264      28: 00009417           auipc s0,0x9
265      2c: fd840413           addi  s0,s0,-40 # 9000 <_answer>
```

Makefile

[Introduction Link]

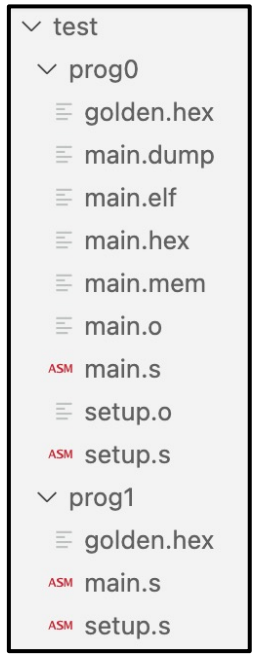

```
1   PRO_PATH ?= ./test/prog0/
2   SRC_NAME ?= main
3   ELF_NAME ?= main
4   SET_NAME ?= setup
5
6   SRC ?= $(PRO_PATH)$(SRC_NAME)
7   ELF ?= $(PRO_PATH)$(ELF_NAME)
8   SET ?= $(PRO_PATH)$(SET_NAME)
9
10  export CROSS_PREFIX ?= riscv32-unknown-elf-
11  export RISCV_AS ?= $(CROSS_PREFIX)as
12  export RISCV_LD ?= $(CROSS_PREFIX)ld
13  export RISCV_OBJDUMP ?= $(CROSS_PREFIX)objdump -xsd
14  export RISCV_OBJCOPY ?= $(CROSS_PREFIX)objcopy -O verilog
15
16  LDFILE := link.ld
17  CFLAGS := -march=rv32i -mabi=ilp32
18  LDFLAGS := -b elf32-littleriscv -T $(LDFILE)
```

```
21  .PHONY: all
22
23  all: build_mem build_dump
24
25  # Transfer format (byte -> word) (.hex -> .mem)
26  build_mem: build_hex
27      python3 bin2mem.py --bin $(ELF).hex
28
29  # Generate binary file (format: verilog byte by byte) (.elf -> .hex)
30  build_hex: build_elf
31      $(RISCV_OBJCOPY) $(ELF).elf $(ELF).hex
32
33  # Generate Executable and Linkable Format (elf) file (.o .o .o ... -> .elf)
34  build_elf: build_object
35      $(RISCV_LD) $(LDFLAGS) $(SET).o $(SRC).o -o $(ELF).elf
36
37  # Generate object files (*.s -> *.o)
38  build_object :
39      $(RISCV_AS) $(CFLAGS) $(SET).s -o $(SET).o
40      $(RISCV_AS) $(CFLAGS) $(SRC).s -o $(SRC).o
41
42  # Disassemble for debugging (.elf -> .dump)
43  build_dump: build_elf
44      $(RISCV_OBJDUMP) $(ELF).elf > $(ELF).dump
45
46  .PHONY: clean
47
48  clean:
49      rm -rf $(PRO_PATH)*.elf $(PRO_PATH)*.dump  $(ELF).hex  $(PRO_PATH)*.o $(PRO_PATH)*.mem
```

Makefile

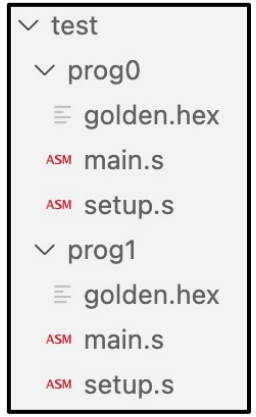
$ make

```
(base) wayne@wayne-Linux:~/tmp$ cd Lab7
(base) wayne@wayne-Linux:~/tmp/Lab7$ make
riscv32-unknown-elf-as -march=rv32i -mabi=ilp32 ./test/prog0/setup.s -o ./test/prog0/setup.o
riscv32-unknown-elf-as -march=rv32i -mabi=ilp32 ./test/prog0/main.s -o ./test/prog0/main.o
riscv32-unknown-elf-ld -b elf32-littleriscv -T link.ld ./test/prog0/setup.o ./test/prog0/main.o -o ./test/prog0/main.elf
riscv32-unknown-elf-objcopy -O verilog ./test/prog0/main.elf ./test/prog0/main.hex
python3 bin2mem.py --bin ./test/prog0/main.hex
riscv32-unknown-elf-objdump -xsd ./test/prog0/main.elf > ./test/prog0/main.dump
```



$ make clean

```
(base) wayne@wayne-Linux:~/tmp/Lab7$ make clean
rm -rf ./test/prog0/*.elf ./test/prog0/*.dump  ./test/prog0/main.hex  ./test/prog0/*.o ./test/prog0/*.mem
```



Summary

1. Put your CPU into src folder
   (Pay attention to the name : Top.v / Top / im / dm / mem / regfile / registers)

2. Use makefile to produce .hex file from .s file (in Ubuntu)
   (1) Change directory to Lab7 (use cd)
   (2) $ make / $ make all

3. Copy main.hex & main.dump into test/prog0/ folder (from Ubuntu to Windows)
   (main.hex is the byte memory that needs to be loaded into mem of im & dm)
   (main.dump is used for debugging)

4. Use ModelSim to check results and debug with waveform & .dump file

```
1   module RegFile (
2       // ...
3       // Finish by yourself
4       // ...
5   );
6
7   reg [31:0] registers [0:31];
```

```
1   module SRAM (
2       // ...
3       // Finish by yourself
4       // ...
5   );
6
7   reg [7:0] mem [0:65535];
```

Top.v

```
module Top (
    input clk,
    input rst
);

SRAM im(
    ...
);

SRAM dm(
    ...
);

RegFile regfile(
    ...
);

...
```

Format of the Lab Report (PDF !!!)

- Cover (There is a default format of the report on the Moodle.)
- Content of the report
  1. Draw an Architecture Diagram
     - Please use draw.io or powerpoint or any other painting software
     - Don't use paper & pen
     - Draw the Architecture Diagram of your CPU by yourself
       (You can draw a same diagram provided by TA if you use the same architecture as TA.)
  2. Introduce each module (function / corner case / and so on…)
  3. Screenshot the successful result (prog0)
  4. (Optional) Your prog1, prog2, …

File structure for submission