CPU Lab (7 & 8)
CPU Test & Debug

Video Link : https://youtu.be/O68zw7YfQGY

How to test your CPU design

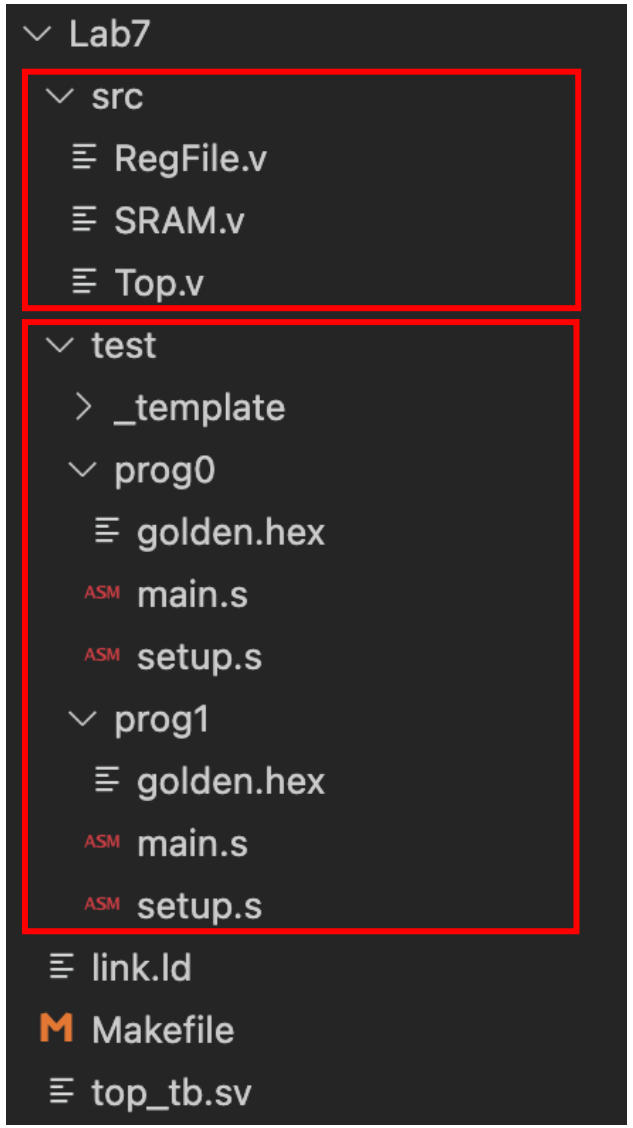
After you write all .v files (Top.v / ALU.v / RegFile.v …)

1. Use testbench to feed the test programs to CPU
   - Testbench is provided by TA
   - Test program 0 is provided by TA
   - Test program 1 is your MergeSort

2. Check result (Compare answer and golden by testbench)

3. Debug with waveform and .dump file
   - You need to know what is the behavior of the program first
     => Read .s first

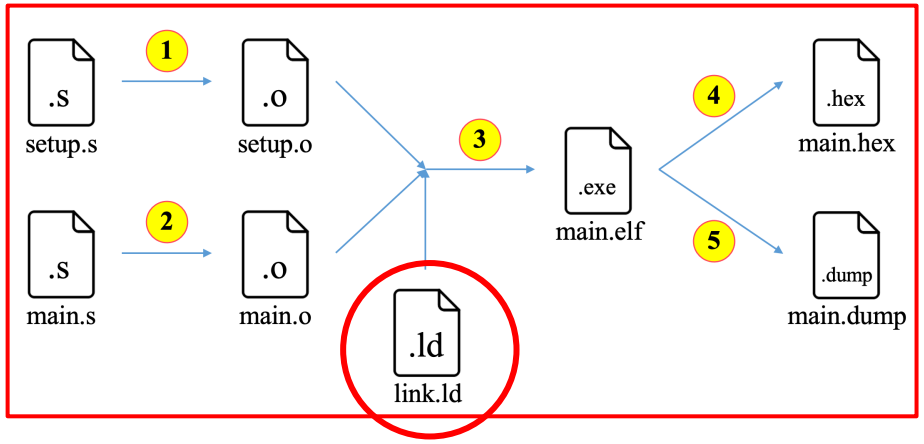
File Structure

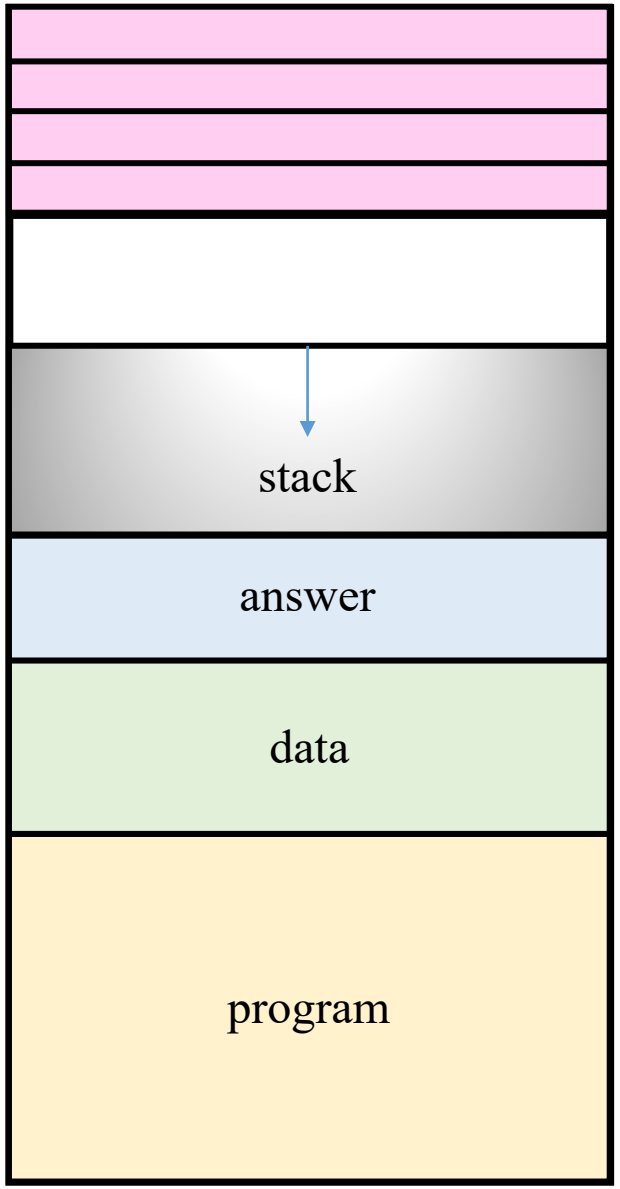


- **src** : All source files of your CPU
- **test** : The programs to test your CPU
- **Makefile** : A script used to convert the test program from $.s \rightarrow .hex$
  $(*.s \rightarrow *.o \rightarrow .elf \rightarrow .hex)$
  $(.elf \rightarrow .dump(disassemble, used\ to\ debug))$
  We load .hex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

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  }
```





Outline

1. Testbench (Simulated by NC-Verilog)
2. Program (Compiled from .s to .hex by RISC-V Toolchain)
3. Summary

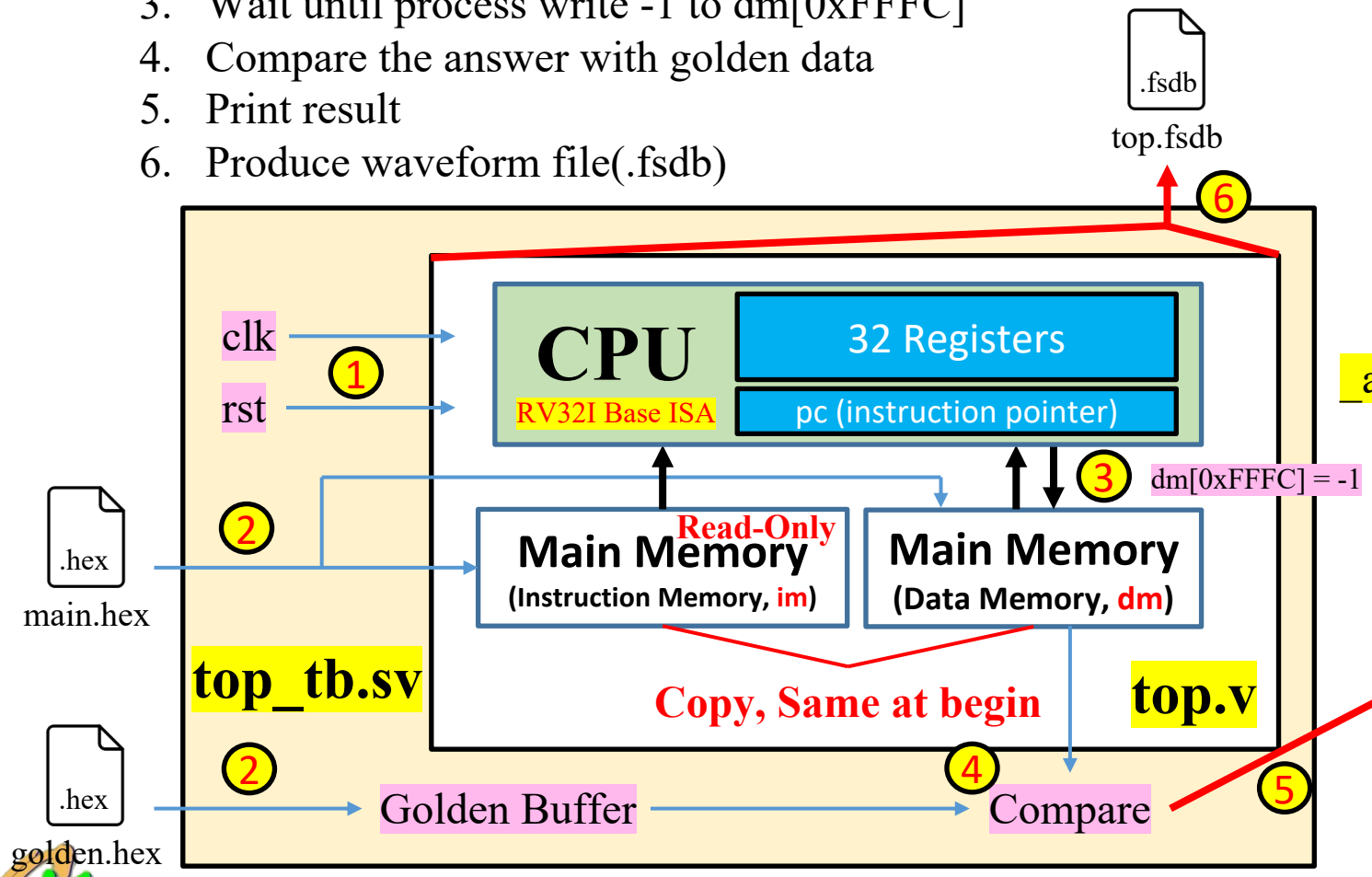
Testbench

Testbench

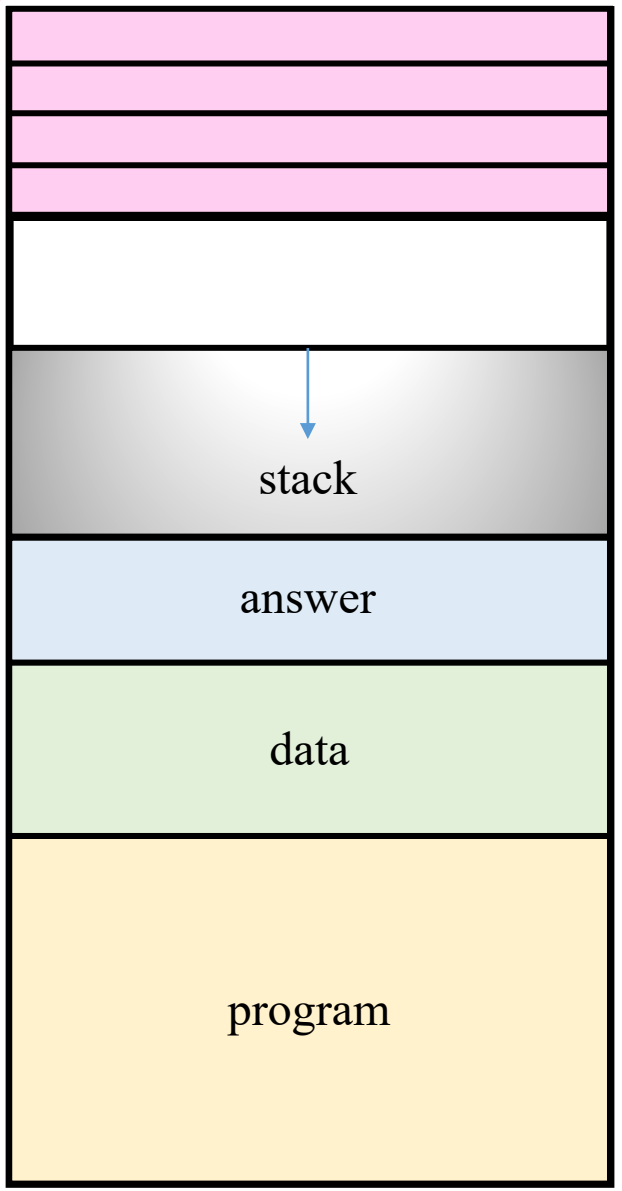
- **Testbench : Simulation Environment**
  1. Given clock(clk), reset(rst) signals
  2. Load “Program” & “Pre-prepared data” into im & dm
  3. Wait until process write -1 to dm[0xFFFC]
  4. Compare the answer with golden data
  5. Print result
  6. Produce waveform file(.fsdb)

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

$2^{16} = 65536$



Program Memory Space DRAM (Baremetal)



Testbench

```
6   `define CYCLE 10.0      // Cycle time
7   `define MAX 10000000    // Max cycle number
```

```
18  `include "./src/Top.v"
```

1. Include & Instance Top module

```
34  Top top (
35    .clk(clk),
36    .rst(rst)
37  );
```

2. clk start from 0, and rst is 1 at begin

```
43    clk = 0; rst = 1;
```

3. Get path from command argument

```
45    // Get Path (main.hex / golden.hex)
46    if($value$plusargs("PROG=%s", prog)) begin
47      prog_path = $sformatf("./test/%s/main.hex", prog);
48      gold_path = $sformatf("./test/%s/golden.hex", prog);
49    end
50    else begin
51      prog_path = "./test/prog0/main.hex";
52      gold_path = "./test/prog0/golden.hex";
53    end
```

4. Load main.hex into im & dm

```
56    // Load main.hex (Program & Preset data)
57    handler = $fopen(prog_path, "r");
58    if (handler == 0) begin
59      $display("\n\n\nError !!! No found \"%s\"\n\n\n", prog_path);
60      $finish;
61    end
62
63    $readmemh(prog_path, top.im.mem);
64    $readmemh(prog_path, top.dm.mem);
```

5. After Loading main.hex & golden.hex
   → rst = 0 (start running CPU)
   → Wait until mem[0xFFFC] == 8'hFF (end of execution)

```
88    // Begin Running !
89    #(`CYCLE) rst = 0;
```

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

6. Compare answer and golden data
7. Max Cycle limitation

```
186   initial begin
187     #(`CYCLE*`MAX)

219     $finish;
220   end
```

Testbench Result

- If you haven't produced main.hex

```
Open failed on file "./test/prog0/main.hex". No such file or directory


Error !!! No found "./test/prog0/main.hex"


Simulation complete via $finish(1) at time 0 FS + 0
./top_tb.sv:60        $finish;
```

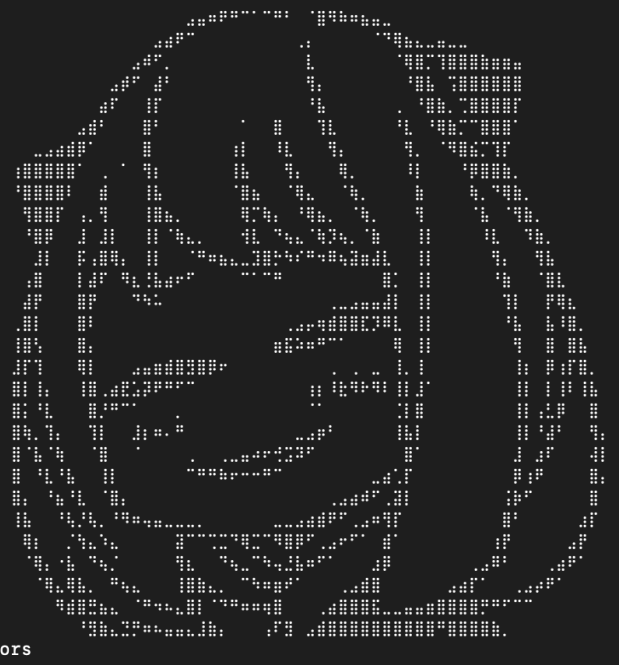
- If you Success

```
DM['h9088] = 12345678, pass
DM['h908c] = ce780000, pass
DM['h9090] = fffff000, pass
DM['h9094] = fffff000, pass
DM['h9098] = fffff000, pass
DM['h909c] = fffff000, pass
DM['h90a0] = fffff000, pass
DM['h90a4] = fffff000, pass
DM['h90a8] = 13579d7c, pass
DM['h90ac] = 13578000, pass
DM['h90b0] = fffff004, pass

    ****************************
    **                        **
    **  Waku Waku !!          **
    **                        **
    **  Simulation PASS !!    **
    **                        **
    ****************************

Simulation complete via $finish(1) at time 6225 NS + 2
./top_tb.sv:115     $finish;
```

Testbench Result

- If you have any error

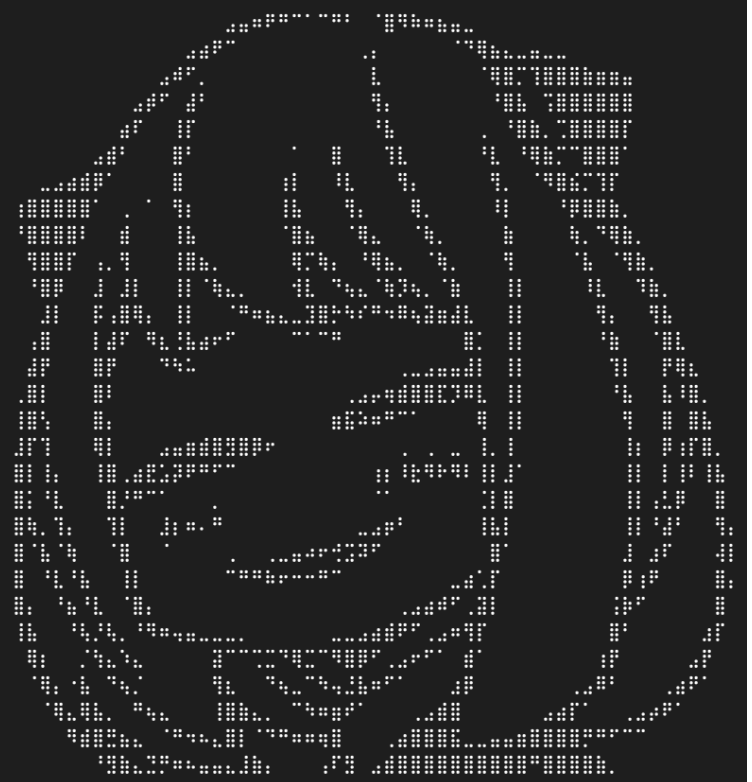
```
DM['h9084] = xxxxxx78, expect = 00000078
DM['h9088] = 12345678, pass
DM['h908c] = ce780000, pass
DM['h9090] = fffff000, pass
DM['h9094] = fffff000, pass
DM['h9098] = fffff000, pass
DM['h909c] = fffff000, pass
DM['h90a0] = fffff000, pass
DM['h90a4] = fffff000, pass
DM['h90a8] = 13579d7c, pass
DM['h90ac] = 13578000, pass
DM['h90b0] = fffff004, pass
```



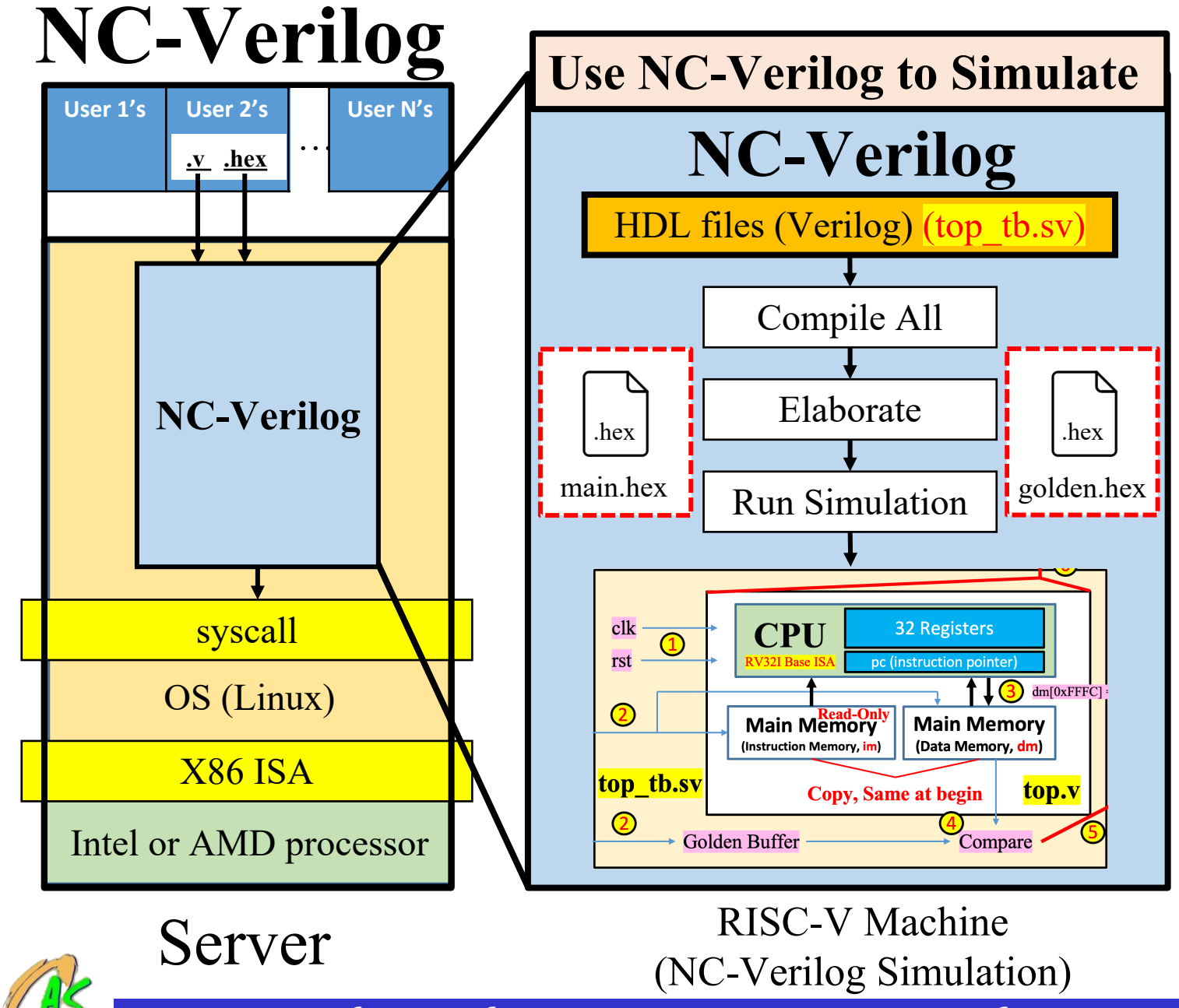
```
   ****************************
   **                        **
   **  OOPS !!               **
   **                        **
   **  Simulation Failed !!  **
   **                        **
   ****************************

        Totally has          1 errors

Simulation complete via $finish(1) at time 6225 NS + 2
./top_tb.sv:115     $finish;
```

- If your CPU can't stop
  (still running and achieve MAX Cycle)

```
xcelium> run
```



```
   ****************************
   **                        **
   **  Time Out !!           **
   **                        **
   **  Simulation Failed !!  **
   **                        **
   ****************************

        Totally has          x errors

Simulation complete via $finish(1) at time 100 MS + 0
./top_tb.sv:219     $finish;
```



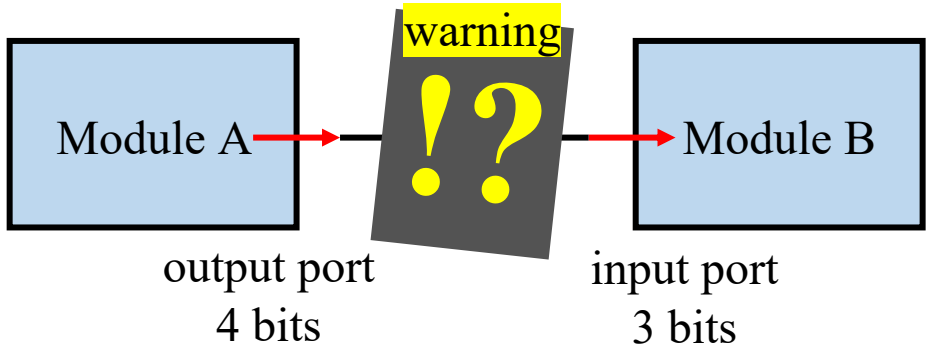
1. Compilation

Analyze the source code for syntax and semantic errors

```
$ ncvlog top_tb.sv -define FSDB
```

2. Elaboration

- Computes parameter values
- Binds modules
- Establishes net connectivity
- … etc, prepares all for simulation



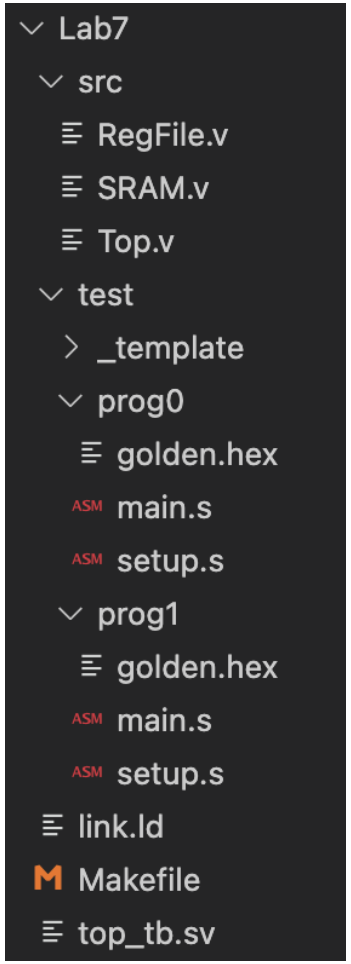

```
$ ncelab top_tb –access +r
```

3. Simulation

Run with the given execution model generated after elaboration

```
$ ncsim top_tb +PROG=prog0
```

NC-Verilog Command




```
45    // Get Path (main.hex / golden.hex)
46    if($value$plusargs("PROG=%s", prog)) begin
47      prog_path = $sformatf("./test/%s/main.hex", prog);
48      gold_path = $sformatf("./test/%s/golden.hex", prog);
49    end
50    else begin
51      prog_path = "./test/prog0/main.hex";
52      gold_path = "./test/prog0/golden.hex";
53    end
```

```
179   initial begin
180     `ifdef FSDB
181       $fsdbDumpfile("top.fsdb");
182       $fsdbDumpvars("+struct", "+mda",top);
183     `endif
184   end
```

```
$ ncverilog top_tb.sv +PROG=prog0 +define+FSDB +access+r
```

ncverilog can do “compilation(ncvlog) / elaboration(ncelab) / simulation(ncsim)” at once

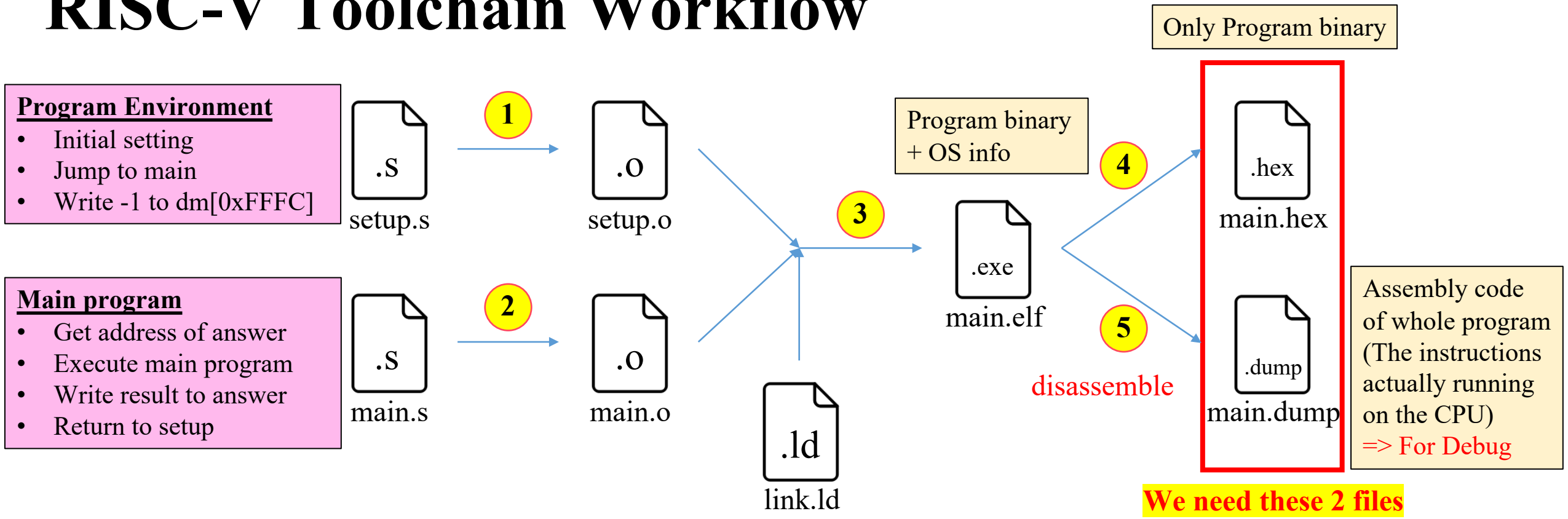
NC-Verilog Command

```
user:~/workspace> cd Lab7
user:~/workspace/Lab7> ncverilog top_tb.sv +PROG=prog0 +define+FSDB +access+r
TOOL:   xmverilog       22.03-s003: Started on Jan 08, 2023 at 22:41:38 CST
xmverilog(64): 22.03-s003: (c) Copyright 1995-2022 Cadence Design Systems, Inc.
xmverilog: *W,NCEXDEP: Executable (ncverilog) is deprecated. Use (xmverilog) instead.
Recompiling... reason: unable to stat '/./home/CO2022/co2022_ta01/workspace/Lab7_work/
               Caching library 'worklib' ....... Done
       Elaborating the design hierarchy:
xmelab: *W,DSEMEL: This SystemVerilog design will be simulated as per IEEE 1800-2009 S
       Building instance overlay tables: .................... Done
       Building instance specific data structures.
       Loading native compiled code:     .................... Done
       Design hierarchy summary:
                            Instances  Unique
              Modules:             18      13
              Registers:           21      20
              Scalar wires:        11       -
              Expanded wires:      32       1
              Vectored wires:      25       -
              Always blocks:        9       8
              Initial blocks:       3       3
              Cont. assignments:   17      23
```

```
Done

DM['h9000] = fffffff0, pass
DM['h9004] = fffffff8, pass
DM['h9008] = 00000008, pass
DM['h900c] = 00000001, pass
DM['h9010] = 00000001, pass
DM['h9014] = 78787878, pass
DM['h9018] = 000091a2, pass
DM['h901c] = 00000003, pass
DM['h9020] = fefcfefd, pass
DM['h9024] = 10305070, pass
DM['h9028] = cccccccc, pass
DM['h902c] = fffffcc, pass
DM['h9030] = ffffcccc, pass
DM['h9034] = 000000cc, pass
DM['h9038] = 0000cccc, pass
DM['h903c] = 00000d9d, pass
DM['h9040] = 00000004, pass
DM['h9044] = 00000003, pass
DM['h9048] = 000001a6, pass
DM['h904c] = 00000ec6, pass
DM['h9050] = 2468b7a8, pass
DM['h9054] = 5dbf9f00, pass
DM['h9058] = 00012b38, pass
DM['h905c] = fa2817b7, pass
DM['h9060] = ff000000, pass
DM['h9064] = 12345678, pass
DM['h9068] = 0000f000, pass
DM['h906c] = 00000f00, pass
DM['h9070] = 000000f0, pass
DM['h9074] = 0000000f, pass
DM['h9078] = 56780000, pass
DM['h907c] = 78000000, pass
DM['h9080] = 00005678, pass
DM['h9084] = 00000078, pass
DM['h9088] = 12345678, pass
DM['h908c] = ce780000, pass
DM['h9090] = fffff000, pass
DM['h9094] = fffff000, pass
DM['h9098] = fffff000, pass
DM['h909c] = fffff000, pass
DM['h90a0] = fffff000, pass
DM['h90a4] = fffff000, pass
DM['h90a8] = 13579d7c, pass
DM['h90ac] = 13578000, pass
DM['h90b0] = fffff004, pass

**************************
**                      **
**  Waku Waku !!        **
**                      **
**  Simulation PASS !!  **
**                      **
**************************
```

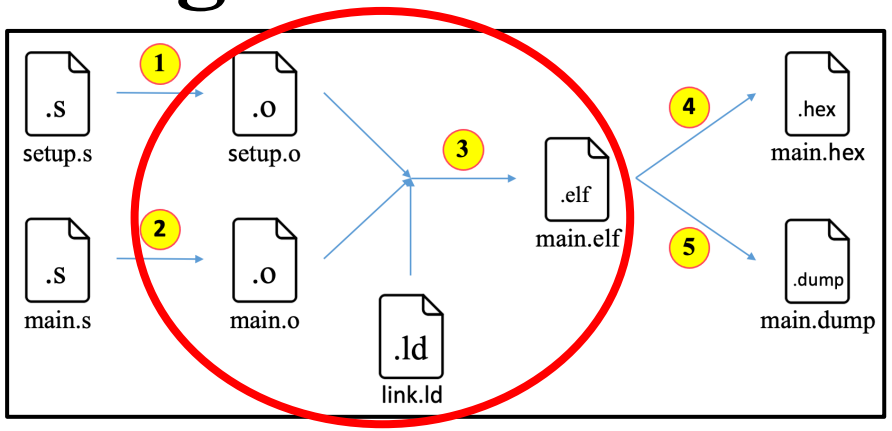
Program

RISC-V Toolchain Workflow



1. riscv32-unknown-elf-as -march=rv32i -mabi=ilp32 [setup.s] -o [setup.o]
2. riscv32-unknown-elf-as -march=rv32i -mabi=ilp32 [main.s] -o [main.o]
3. riscv32-unknown-elf-ld -b elf32-littleriscv -T [link.ld] [setup.o] [main.o] -o [main.elf]
4. riscv32-unknown-elf-objcopy -O verilog [main.elf] [main.hex]
5. riscv32-unknown-elf-objdump -xsd [main.elf] > [main.dump]

Program Structure



Program Environment (setup.s)

- Initial setting
- Jump to main
- Write -1 to dm[0xFFFC]

```
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
```

setup.s

Main program (main.s)

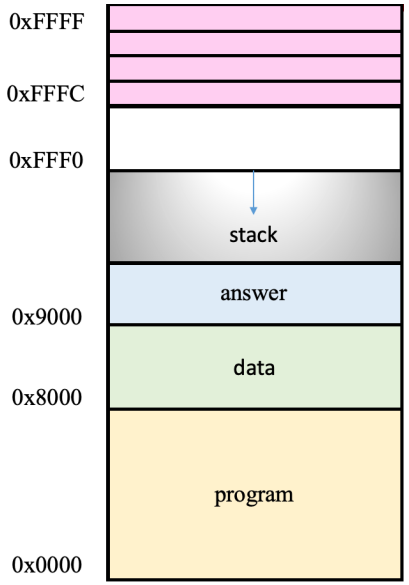
- Get address of answer
- Execute main program
- Write result to answer
- Return to setup

```
1   .data
2   # ...
3
4   .text
5   .globl main
6
7   main:
8
9   # ####################################
10  # ### Load address of _answer to s0
11  # ####################################
12
13    addi sp, sp, -4
14    sw s0, 0(sp)
15    la s0, _answer
16
17  # ####################################
18
19
20  # ####################################
21  # ### Main Program
22  # ####################################
23
24  # ...
25
26  # ####################################
27
28
29  main_exit:
30
31  # ####################################
32  # ### Return to end the simulation
33  # ####################################
34
35    lw s0, 0(sp)
36    addi sp, sp, 4
37    ret
38
39  # ######################
40
```

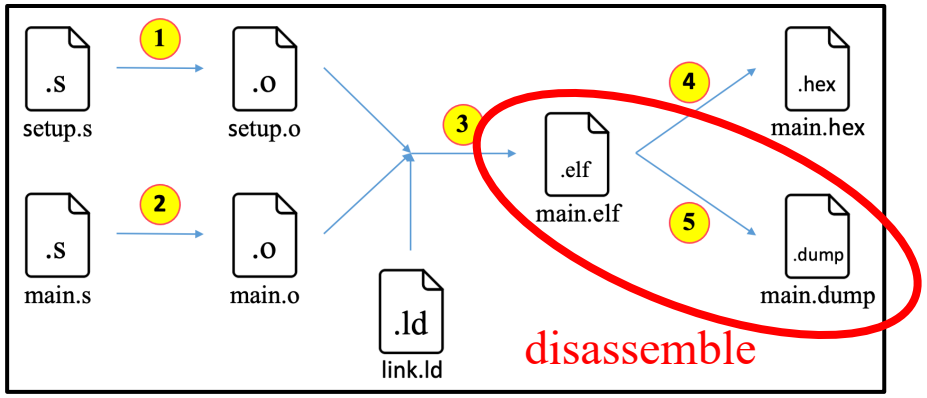
main.s

```
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  }
```

link.ld



Program Structure



main.dump

```
Disassembly of section .text:

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

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

                                      la t0, _sim_end
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>:                      la s0, _answer
  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>

00000030 <main_exit>:
  30: 00012403            lw  s0,0(sp)
  34: 00410113            addi  sp,sp,4
  38: 00008067            ret
```

① ③ ②

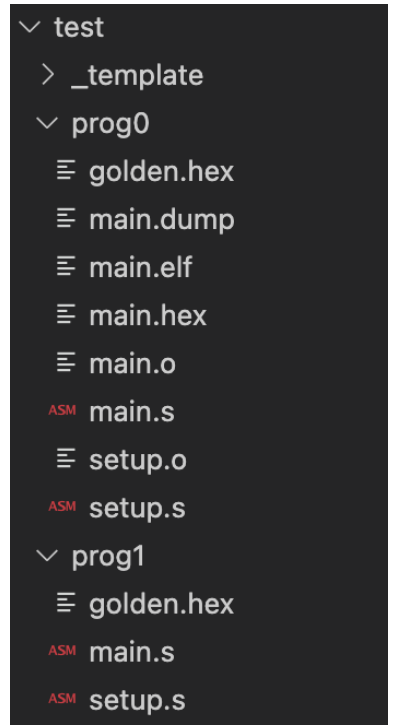
Makefile

[Introduction Link]

```
1   PROG ?= prog0
2   PRO_PATH ?= ./test/$(PROG)/
```

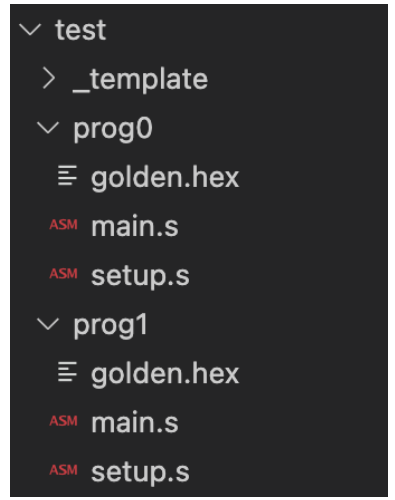
$ make PROG=prog0

```
user:~/workspace> cd Lab7
user:~/workspace/Lab7> make PROG=prog0
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
riscv32-unknown-elf-objdump -xsd ./test/prog0/main.elf > ./test/prog0/main.dump
```



$ make clean PROG=prog0

```
user:~/workspace/Lab7> make clean PROG=prog0
rm -rf BSSLib.lib++ INCA_libs nWaveLog *.history *.log *.conf *.fsdb *.svf *.txt *.key *.X *.. *
.d novas.rc ./test/prog0/*.elf ./test/prog0/*.dump  ./test/prog0/main.hex  ./test/prog0/*.o ./te
st/prog0/*.mem
```



Program 0 (1/5) – main.s

```
7   main:
8
9   # ######################################
10  # ### Load address of _answer to s0
11  # ######################################
12
13    addi sp, sp, -4
14    sw s0, 0(sp)
15    la s0, _answer
16
17  # ######################################
```

```
20  # ######################################
21  # ### Main Program
22  # ######################################
23
24  add:
25    li t0, 0xffffffff   # -1
26    li t1, 0xffffffff   # -1
27    add t0, t0, t1      # t0 = -2
28    add t0, t0, t1      # t0 = -3
29    add t0, t0, t1      # t0 = -4
30    add t0, t0, t1      # t0 = -5
31    add t0, t0, t1      # t0 = -6
32    li t1, 0xfffffffe   # -2
33    add t0, t1, t0      # t0 = -8
34    add t0, t1, t0      # t0 = -10
35    add t0, t1, t0      # t0 = -12
36    add t0, t1, t0      # t0 = -14
37    add t0, t1, t0      # t0 = -16
38    sw t0, 0(s0)        # Write Answer (DM[_answer + 0])
39    addi s0, s0, 4
40  sub:
41    li t0, 0x00000000   # 0
42    li t1, 0xffffffff   # -1
43    sub t0, t0, t1      # t0 = 1
44    sub t0, t0, t1      # t0 = 2
45    sub t0, t0, t1      # t0 = 3
46    sub t0, t0, t1      # t0 = 4
47    sub t0, t0, t1      # t0 = 5
48    li t1, 0xfffffffd   # -3
49    sub t0, t1, t0      # t0 = -8
50    sub t0, t1, t0      # t0 = 5
51    sub t0, t1, t0      # t0 = -8
52    sub t0, t1, t0      # t0 = 5
53    sub t0, t1, t0      # t0 = -8
54    sw t0, 0(s0)        # Write Answer (DM[_answer + 4])
55    addi s0, s0, 4
```

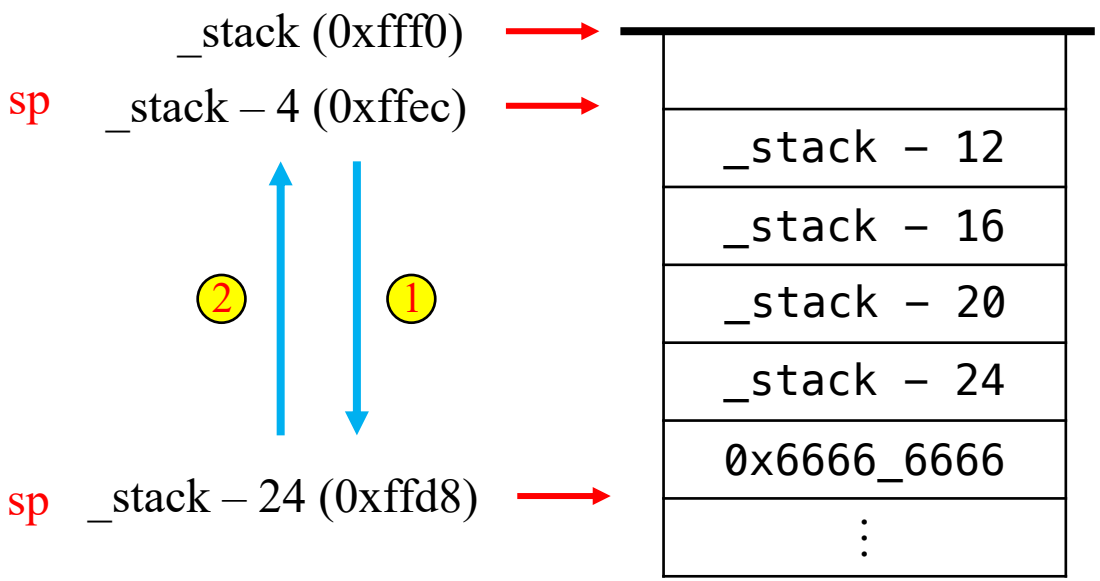
```
56  sll:
57    li t0, 0x00000001   # 1
58    li t1, 0x00000001   # 1
59    sll t0, t0, t1      # t0 = 2
60    sll t0, t0, t1      # t0 = 4
61    sll t0, t0, t1      # t0 = 8
62    sll t0, t0, t1      # t0 = 16
63    sll t0, t0, t1      # t0 = 32
64    li t1, 0x00000002   # 2
65    sll t0, t1, t0      # t0 = 2
66    sll t0, t1, t0      # t0 = 8
67    sll t0, t1, t0      # t0 = 512
68    sll t0, t1, t0      # t0 = 2
69    sll t0, t1, t0      # t0 = 8
70    sw t0, 0(s0)        # Write Answer (DM[_answer + 8])
71    addi s0, s0, 4
72  slt:
73    li t0, 0xffffffff   # -1
74    li t1, 0x00000001   # 1
75    slt t0, t0, t1      # t0 = 1
76    slt t0, t0, t1      # t0 = 0
77    slt t0, t0, t1      # t0 = 1
78    slt t0, t0, t1      # t0 = 0
79    slt t0, t0, t1      # t0 = 1
80    li t1, 0xffffffff   # -1
81    slt t0, t1, t0      # t0 = 1
82    slt t0, t1, t0      # t0 = 1
83    slt t0, t1, t0      # t0 = 1
84    slt t0, t1, t0      # t0 = 1
85    slt t0, t1, t0      # t0 = 1
86    sw t0, 0(s0)        # Write Answer (DM[_answer + 12])
87    addi s0, s0, 4
```


Program 0 (2/5) – main.s

```
1   .data
2   lw_data: .word 0x66666666
```

```
185 load:
186   la t1, lw_data
187   lw t0, 0(t1)          # t0 = 0x66666666
188   addi sp, sp, -20      # sp = _stack - 24
189   sw t0, 0(sp)          # 0x66666666 -> DM[_stack - 24]
190   addi t0, sp, 0        # t0 = _stack - 24
191   sw t0, 4(sp)          # t0 -> DM[_stack - 20]
192   addi t0, sp, 4        # t0 = _stack - 20
193   sw t0, 8(sp)          # t0 -> DM[_stack - 16]
194   addi t0, sp, 8        # t0 = _stack - 16
195   sw t0, 12(sp)         # t0 -> DM[_stack - 12]
196   addi t0, sp, 12       # t0 = _stack - 12
197   sw t0, 16(sp)         # t0 -> DM[_stack - 8]
198   addi t0, sp, 16       # t0 = _stack - 8
199   lw t0, 0(t0)          # t0 = DM[_stack - 8] = _stack - 12
200   lw t0, 0(t0)          # t0 = DM[_stack - 12] = _stack - 16
201   lw t0, 0(t0)          # t0 = DM[_stack - 16] = _stack - 20
202   lw t0, 0(t0)          # t0 = DM[_stack - 20] = _stack - 24
203   lw t0, 0(t0)          # t0 = DM[_stack - 24] = 0x66666666
204   addi sp, sp, 20       # sp = _stack - 4
205   lw t1, -4(sp)         # t1 = DM[_stack - 8] = _stack - 12
206   lw t2, -8(sp)         # t2 = DM[_stack - 12] = _stack - 16
207   lw t3, -12(sp)        # t3 = DM[_stack - 16] = _stack - 20
208   lw t4, -16(sp)        # t4 = DM[_stack - 20] = _stack - 24
209   lw t5, -20(sp)        # t5 = DM[_stack - 24] = 0x66666666
210   add t1, t1, t4        # t1 = (_stack - 12) + (_stack - 24)
211   add t2, t2, t3        # t2 = (_stack - 16) + (_stack - 20)
212   sub t1, t1, t2        # t1 = 0
213   add t1, t1, t5        # t1 = 0 + 0x66666666 = 0x66666666
214   add t0, t0, t1        # t0 = 0x66666666 + 0x66666666 = 0xcccccccc
```

```
215   addi s0, s0, 16
216   sw t0, -16(s0)        # Write Answer (DM[_answer + 40])
217   lb t1, -16(s0)        # t1 = 0xffffffcc
218   lh t2, -16(s0)        # t2 = 0xffffcccc
219   lbu t3, -16(s0)       # t3 = 0x000000cc
220   lhu t4, -16(s0)       # t4 = 0x0000cccc
221   sw t1, -12(s0)        # Write Answer (DM[_answer + 44])
222   sw t2, -8(s0)         # Write Answer (DM[_answer + 48])
223   sw t3, -4(s0)         # Write Answer (DM[_answer + 52])
224   sw t4, 0(s0)          # Write Answer (DM[_answer + 56])
225   addi s0, s0, 4
```



Program 0 (3/5) – main.s

```
406   jalr:
407     li t0, 0xfffff000   #
408     la t1, 1f           # t1 = Addr (Forward Local Label 1)
409   1 jalr t1, t1, 0      # First jump
410     ori t0, t0, 1       # t0 = 0xfffff001 // Shouldn't be execute
411   3 jalr t1, t1, 4      # Third jump
412     ori t0, t0, 4       # t0 = 0xfffff004 // Shouldn't be execute
413   5 jalr t1, t1, 4      # Fifth jump
414     ori t0, t0, 16      # t0 = 0xfffff010 // Shouldn't be execute
415   1:
416   2 jalr t1, t1, 4      # Second jump
417     ori t0, t0, 1       # t0 = 0xfffff002 // Shouldn't be execute
418   4 jalr t1, t1, 4      # Fourth jump
419     ori t0, t0, 8       # t0 = 0xfffff008 // Shouldn't be execute
420   # // ====================
421   6 la t1, 2f           # t2 = Addr (Forward Local Label 2)
422   7 jalr t2, t1, -32    # t2 = t1 - 36    // First jump
423     ori t0, t0, 32      # t0 = 0xfffff020 // Shouldn't be execute
424   8 jalr t3, t1, -24    # t3 = t1 - 28    // Second jump
425     ori t0, t0, 64      # t0 = 0xfffff040 // Shouldn't be execute
426   9 jalr t4, t1, -16    # t4 = t1 - 20    // Third jump
427     ori t0, t0, 128     # t0 = 0xfffff080 // Shouldn't be execute
428  10 jalr t5, t1, -8     # t5 = t1 - 12    // Fourth jump
429     ori t0, t0, 256     # t0 = 0xfffff100 // Shouldn't be execute
430  11 jalr t6, t1, 0      # t6 = t1 - 4     // Fifth jump
431     ori t0, t0, 512     # t0 = 0xfffff200 // Shouldn't be execute
```

```
432   2:
433  12 sub t2, t2, t3      # t2 = -8
434     sub t4, t4, t5      # t4 = -8
435     sub t6, t6, t1      # t6 = -4
436     add t2, t2, t4      # t2 = -16
437     add t1, t2, t6      # t1 = -20
438     sll t0, t0, t1      # t0 = 0xfffff000 << 5'b01100 = 0xff000000
439     sw t0, 0(s0)        # Write Answer (DM[_answer + 96])
440     addi s0, s0, 4
```

link

Labels

Text labels are used as branch, unconditional jump targets and symbol offsets. Text labels are added to the symbol table of the compiled module.

```
loop:
        j loop
```

Numeric labels are used for local references. References to local labels are suffixed with 'f' for a forward reference or 'b' for a backwards reference.

```
1:
        j 1b
```


Program 0 (4/5) – main.dump

```
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>

00000030 <add>:
  30: fff00293          li  t0,-1
  34: fff00313          li  t1,-1
  38: 006282b3          add t0,t0,t1
  3c: 006282b3          add t0,t0,t1
  40: 006282b3          add t0,t0,t1
  44: 006282b3          add t0,t0,t1
```

```
   a00: 00542023          sw  t0,0(s0)
   a04: 00440413          addi  s0,s0,4

00000a08 <jal>:
   a08: fffff2b7          lui t0,0xfffff
   a0c: 0080036f          jal t1,a14 <jal+0xc>
   a10: 0012e293          ori t0,t0,1
   a14: 00000397          auipc t2,0x0
   a18: 00038393          mv  t2,t2
   a1c: 40638333          sub t1,t2,t1
   a20: 006282b3          add t0,t0,t1
   a24: 00542023          sw  t0,0(s0)
   a28: 00440413          addi  s0,s0,4

00000a2c <main_exit>:
   a2c: 00012403          lw  s0,0(sp)
   a30: 00410113          addi  sp,sp,4
   a34: 00008067          ret
```

pc : 0x0 -> 0x4 -> 0x8 -> 0x20 -> …… -> 0xa34 -> 0xc -> 0x10 -> 0x14 -> 0x18

Program 0 (5/5)

```
Done

DM['h9000] = fffffff0, pass  add
DM['h9004] = fffffff8, pass  sub
DM['h9008] = 00000008, pass  sll
DM['h900c] = 00000001, pass  slt
DM['h9010] = 00000001, pass  sltu
DM['h9014] = 78787878, pass  xor
DM['h9018] = 000091a2, pass  srl
DM['h901c] = 00000003, pass  sra
DM['h9020] = fefcfefd, pass  or
DM['h9024] = 10305070, pass  and
DM['h9028] = cccccccc, pass
DM['h902c] = ffffffcc, pass
DM['h9030] = ffffcccc, pass  load
DM['h9034] = 000000cc, pass
DM['h9038] = 0000cccc, pass
DM['h903c] = 00000d9d, pass  addi
DM['h9040] = 00000004, pass  slti
DM['h9044] = 00000003, pass  sltiu
DM['h9048] = 000001a6, pass  xori
DM['h904c] = 00000ec6, pass  ori
DM['h9050] = 2468b7a8, pass  andi
DM['h9054] = 5dbf9f00, pass  slli
DM['h9058] = 00012b38, pass  srli
```

```
DM['h905c] = fa2817b7, pass  srai
DM['h9060] = ff000000, pass  jalr
DM['h9064] = 12345678, pass
DM['h9068] = 0000f000, pass
DM['h906c] = 00000f00, pass
DM['h9070] = 000000f0, pass
DM['h9074] = 0000000f, pass
DM['h9078] = 56780000, pass  store
DM['h907c] = 78000000, pass
DM['h9080] = 00005678, pass
DM['h9084] = 00000078, pass
DM['h9088] = 12345678, pass
DM['h908c] = ce780000, pass
DM['h9090] = fffff000, pass  beq
DM['h9094] = fffff000, pass  bne
DM['h9098] = fffff000, pass  blt
DM['h909c] = fffff000, pass  bge
DM['h90a0] = fffff000, pass  bltu
DM['h90a4] = fffff000, pass  bgeu
DM['h90a8] = 13579d7c, pass  auipc
DM['h90ac] = 13578000, pass  lui
DM['h90b0] = fffff004, pass  jal
```

Program 1 - MergeSort

1. Copy all test data from Lab3 MergeSort

```
.data
# ...
```
main.s

```
.data
num_test: .word 3
TEST1_SIZE: .word 34
TEST2_SIZE: .word 19
TEST3_SIZE: .word 29
test1: .word 3,41,18,8,40,6,45,1,18,10,24,46,37,23,43,12,3,37,0,15,11,49,47,27,23,30,16,10,45,39,1,23,40,38
test2: .word -3,-23,-22,-6,-21,-19,-1,0,-2,-47,-17,-46,-6,-30,-50,-13,-47,-9,-50
test3: .word -46,0,-29,-2,23,-46,46,9,-18,-23,35,-37,3,-24,-18,22,0,15,-43,-16,-17,-42,-49,-29,19,-44,0,-18,23
```

2. Modify some addresses in your assembly code

```
int main(){
    int num_test = * (int *) 0x10000000;
    int *size =      (int *) 0x10000004;
    int *test =      (int *) 0x10000004 + num_test;
    int *answer =    (int *) 0x01000000;
```

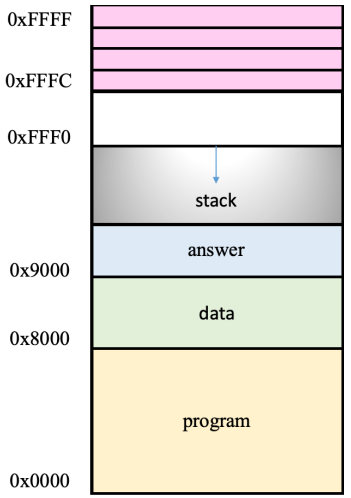
```
int num_test = * (int *) 0x8000
int *size    =   (int *) 0x8004
int *test    =   size + num_test
int *answer  =   (int *) 0x9000 = _answer
```

3. Copy the modified code to Main Program of "main.s"

```
# ######################################
# ### Main Program
# ######################################

# ...

# ######################################
```
main.s

```
.text
setup:
    li    ra, -1
    li    sp, 0x7ffffff0
main:
```

```
OUTPUT_ARCH( "riscv" )

SECTIONS
{
    . = 0x0000;
    .text : { *(.text) }

    . = 0x8000;
    .data : { *(.data) }

    . = 0x9000;
    _answer = .;

    . = 0xfff0;
    _stack = .;

    . = 0xfffc;
    _sim_end = .;

}
```
link.ld



4. Write golden.hex

Summary

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

2. Prepare your program1(.s) and write golden.hex

3. Change directory to Lab7 (use $ cd)

4. Use makefile to produce .hex file from .s file

```
$ make PROG=prog0
```

You can replace prog0 to whatever you want to run

5. Use NC-Verilog to run simulation

```
$ ncverilog top_tb.sv +PROG=prog0 +define+FSDB +access+r
```

6. Check results and debug with waveform & .dump file

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

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

SRAM im(
    ...
);

SRAM dm(
    ...
);

...          Top.v
```