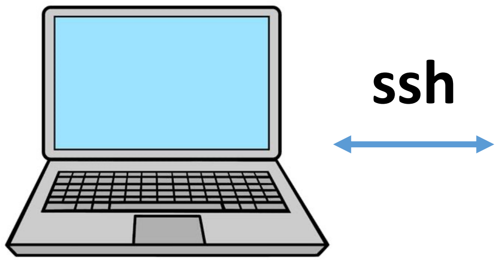
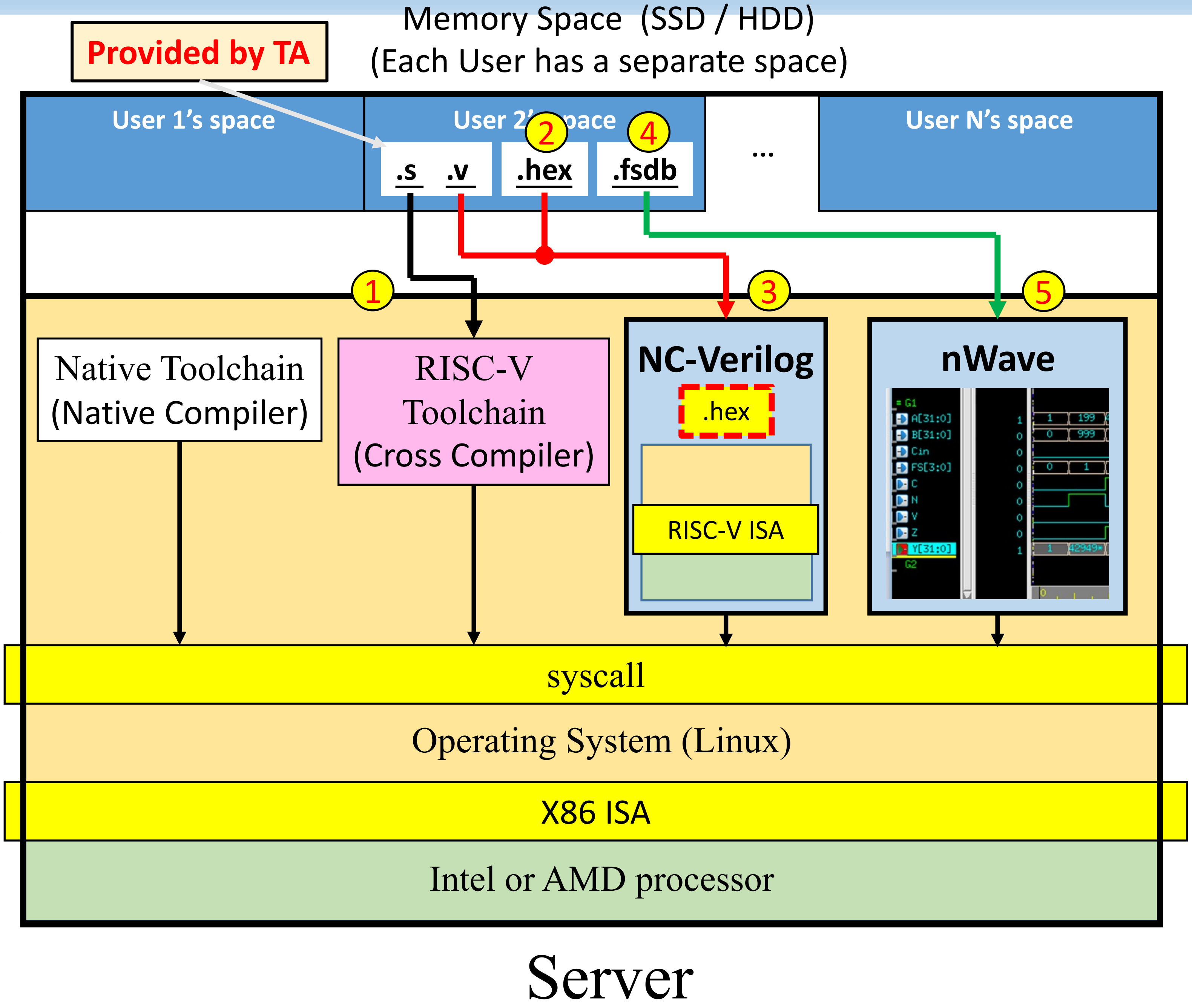
Computer Organization 2022
Lab1 – Introduction & Environment Lab
(Part 3 : A Simple Experiment)

Video link : https://youtu.be/z8znz83HR9c

Part 3
A Simple Experiment

Steps





1. Cross-compile source files using RISC-V Toolchain
   => Generate .hex
2. Simulate with NC-Verilog
   => Generate .fsdb
3. Check waveform with nWave

Outline

1. CPU (Provided by TA)

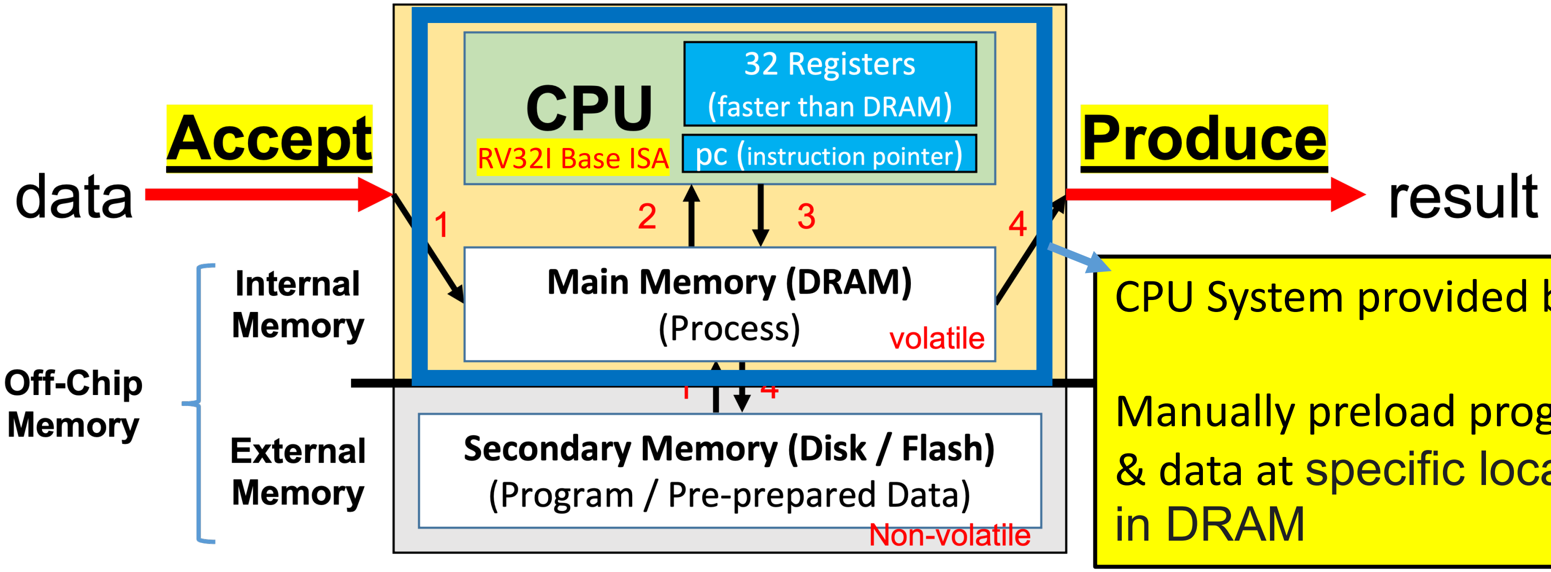
2. Program & RISC-V Toolchain

3. Experiment

CPU (Provided by TA)

CPU (provided by TA)

- The CPU provided by TA is a Single-Cycle RTL-Level RISC-V CPU
- It supports RV32I Base ISA
  (You will write your CPU in Lab7)
- It has been verified, we will use this CPU in lab2 to practice writing assembly program



CPU System provided by TA

Manually preload program & data at specific locations in DRAM

RV32I Instruction Type

- **Computation Instruction**

  int a = 10 (← *immediate*, 立即數)

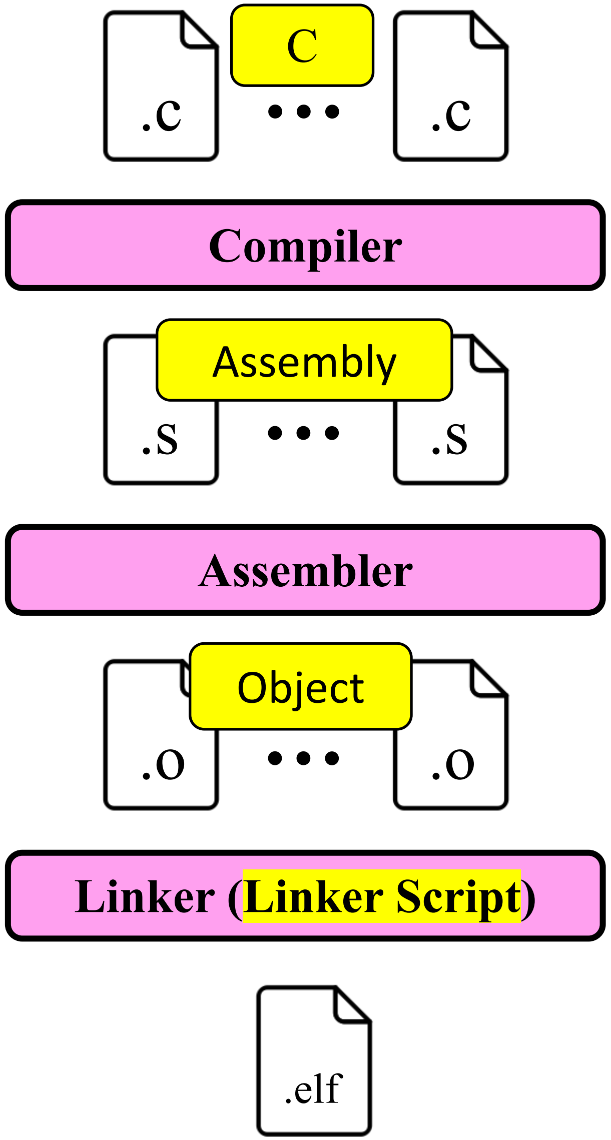
  - (Register(or PC) $\overset{\text{Operation}}{\xLeftrightarrow{(+,-,\ll,\gg,\ldots)}}$ Register(or immediate)) → **Register**
- **Load & Store Instruction**
  - Load : DRAM → **Register**
  - Store : Register → **DRAM**
- **Control Transfer Instruction**
  - (Register(or PC) + immediate) → **PC**

3 Memory Units

- Inside CPU
  - PC (can be seen as a register)
  - 32 Registers
- Outside CPU
  - Main Memory (DRAM)

- **Check instruction behavior by observing the value changes in the memory units**

Program Memory Space



Each program has its own Memory Space which is organized by linker script

Run OS

DRAM for multiple processes

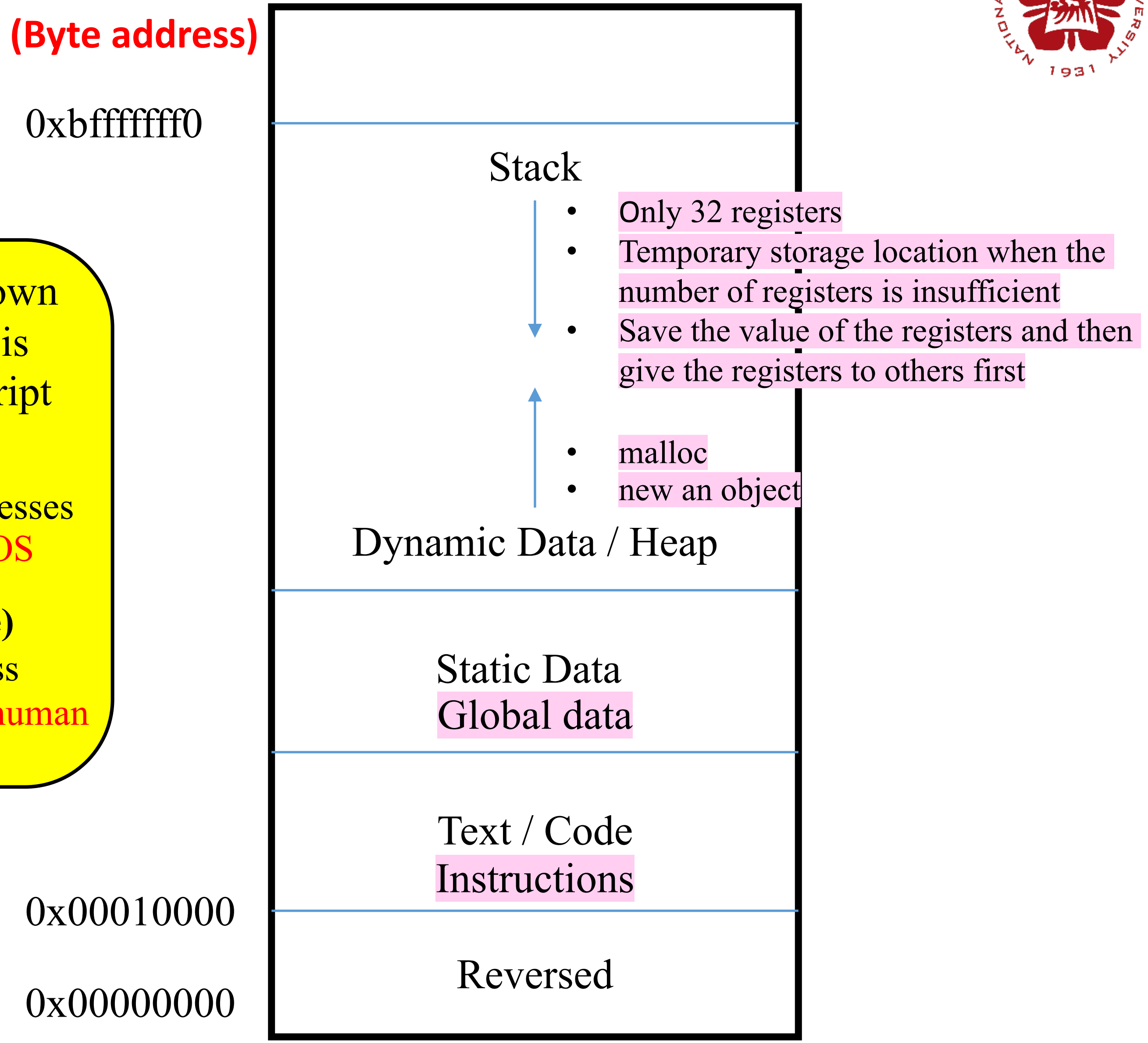
- DRAM managed by OS

Baremetal (We are here)

DRAM all for one process

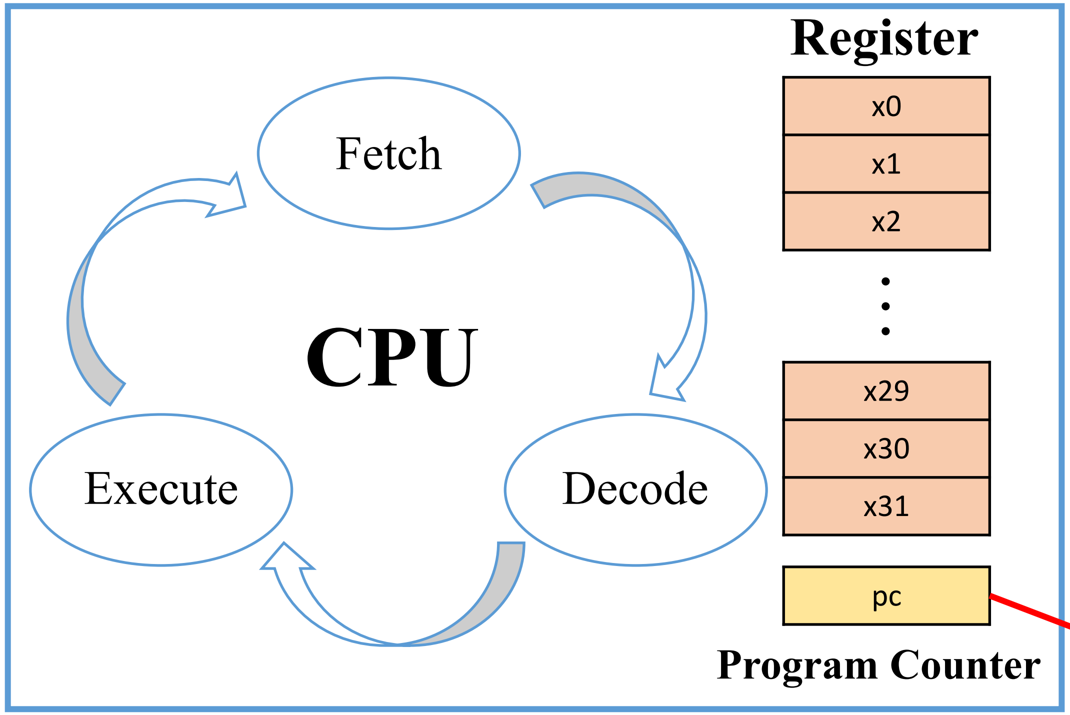
- DRAM managed by human

Program Memory Space



DRAM (Baremetal)

CPU - Workflow



Memory Space is organized by linker script

Run OS

DRAM for multiple processes

- DRAM managed by OS

Baremetal (We are here)

DRAM all for one process

- DRAM managed by human

next_pc = current_pc + 4

Points to the head of the instruction

3 stages :

- Fetch : Fetch an **instruction that "pc" points to** from DRAM
- Decode : Decode the fetched instruction to know what it mean
- Execute : Execute the instruction according the decode result

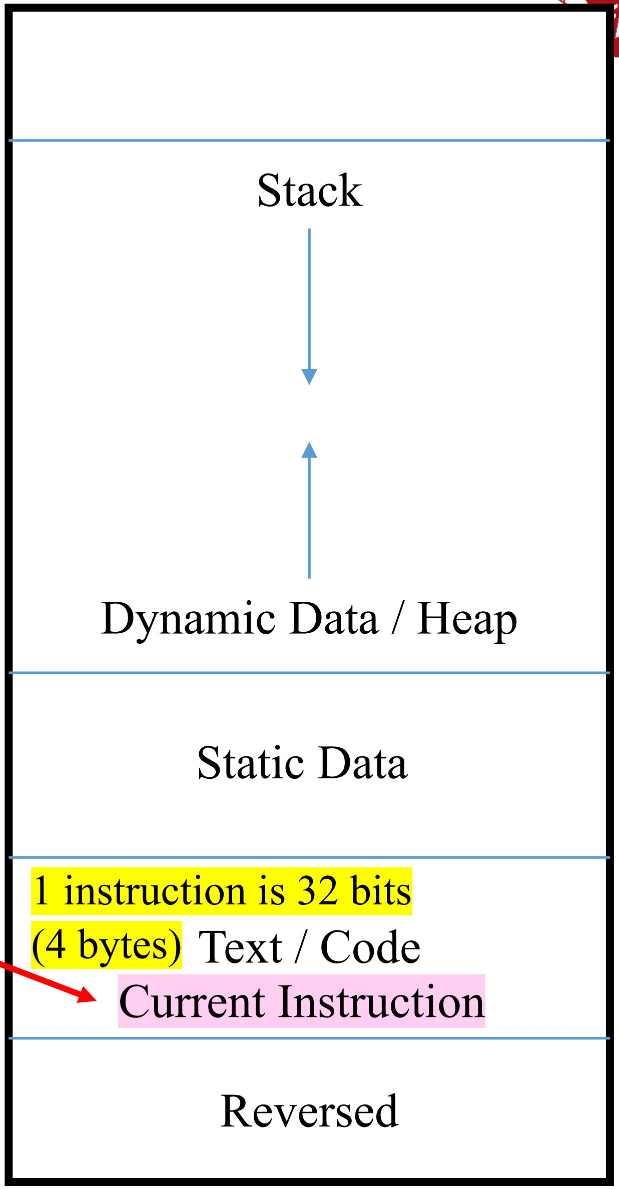
Program Memory Space

(Byte address)

0xbffffff0

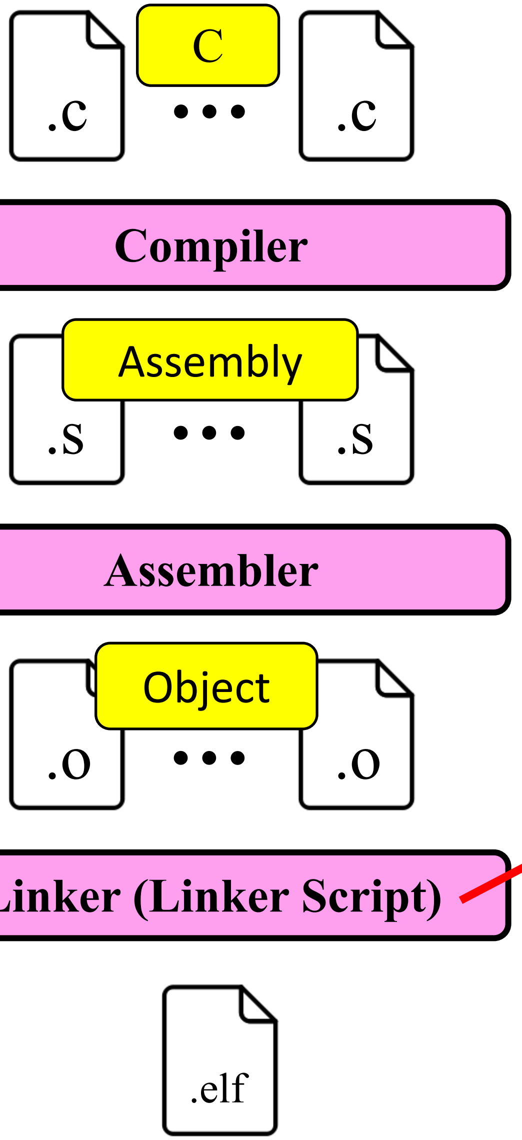
0x00010000

0x00000000



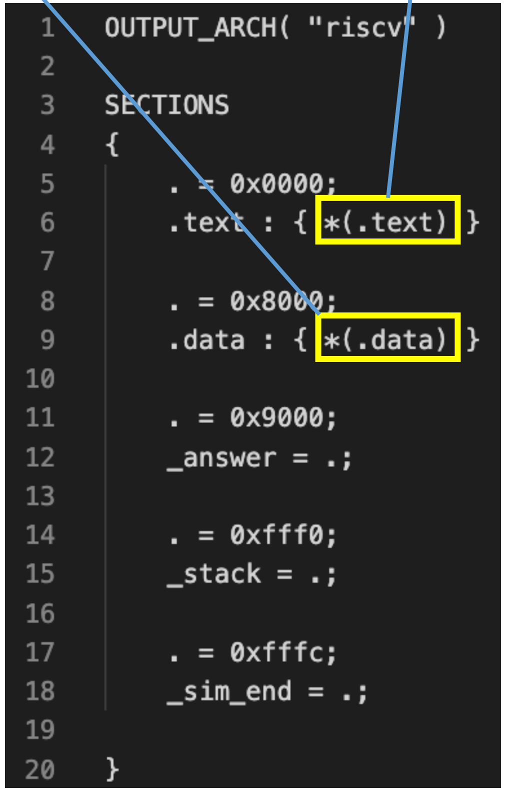
DRAM (Baremetal)

Program Memory Space we used



Concatenate all data sections

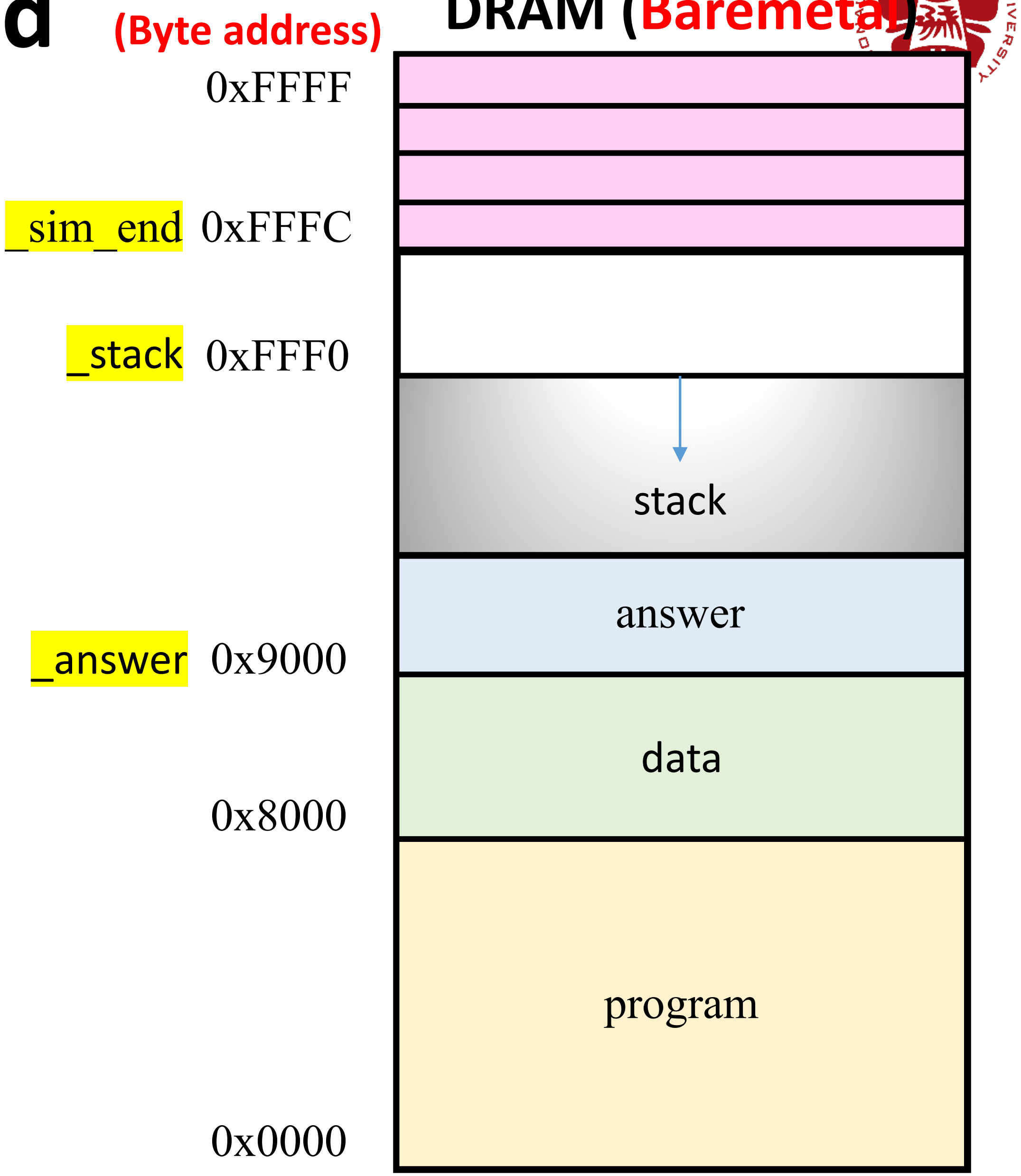
Concatenate all text sections



```
OUTPUT_ARCH( "riscv" )

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

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

    . = 0x9000;
    _answer = .;

    . = 0xfff0;
    _stack = .;

    . = 0xfffc;
    _sim_end = .;

}
```

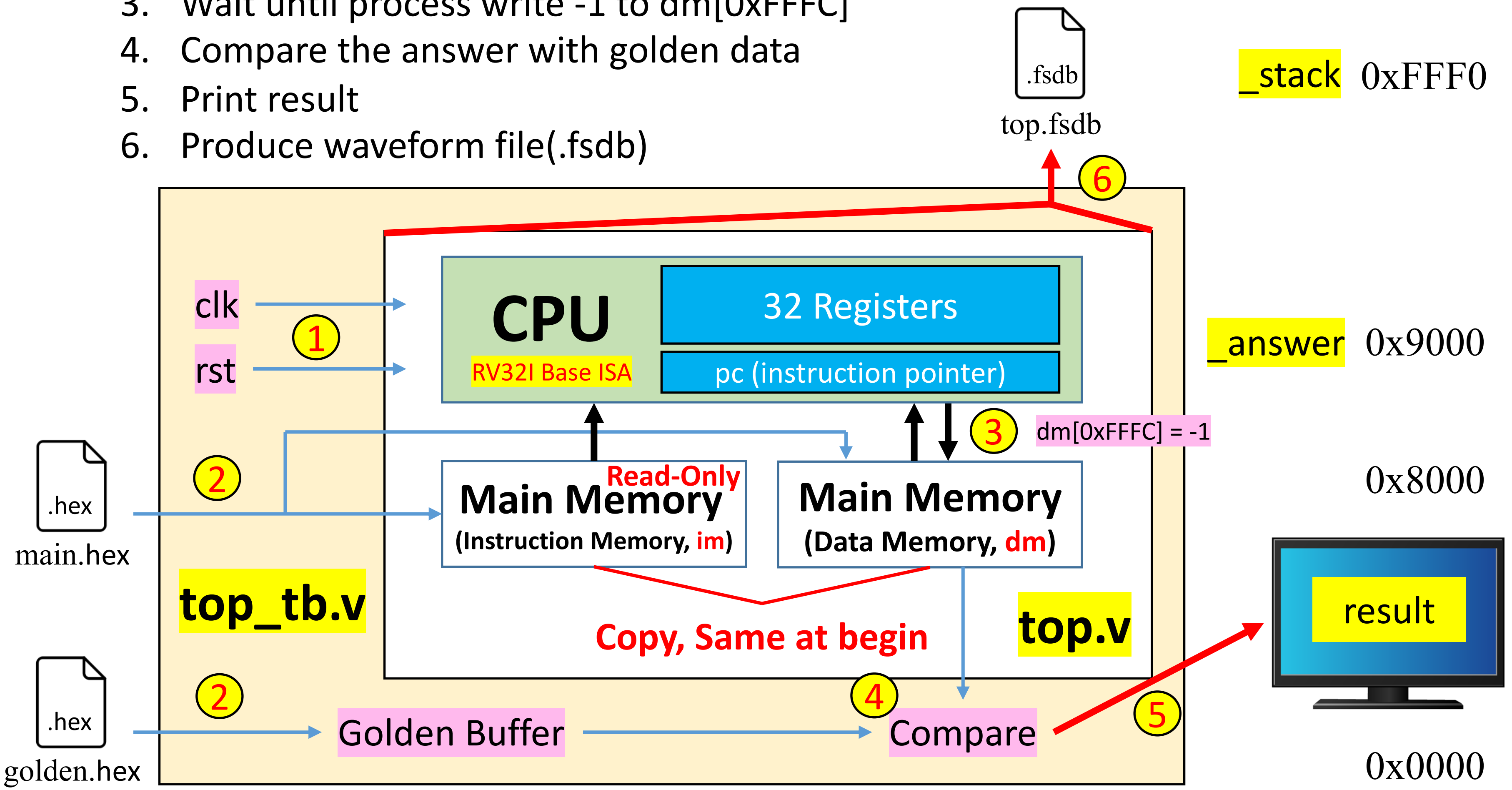
link.ld
(Linker Script)

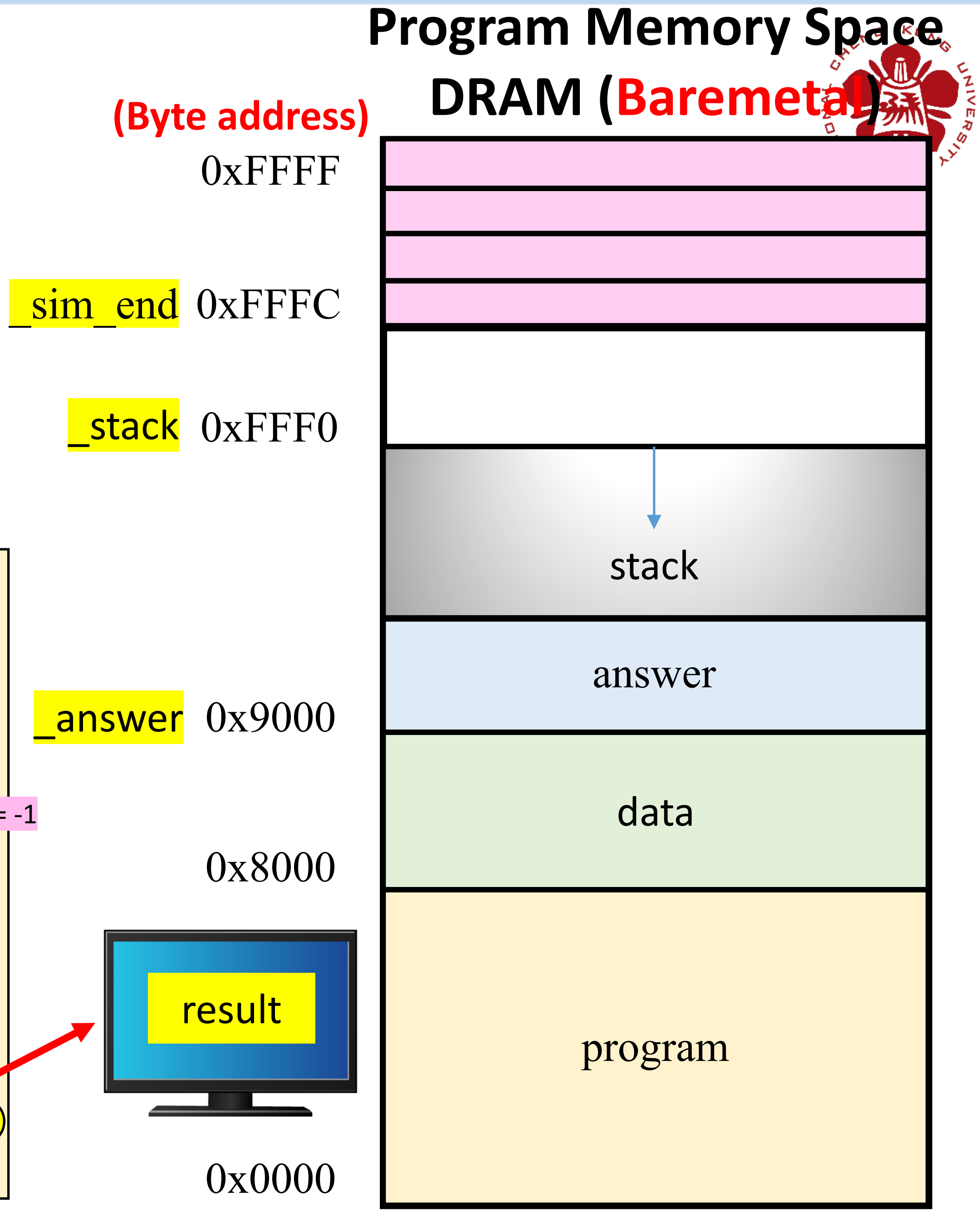
Program Memory Space DRAM (Baremetal)



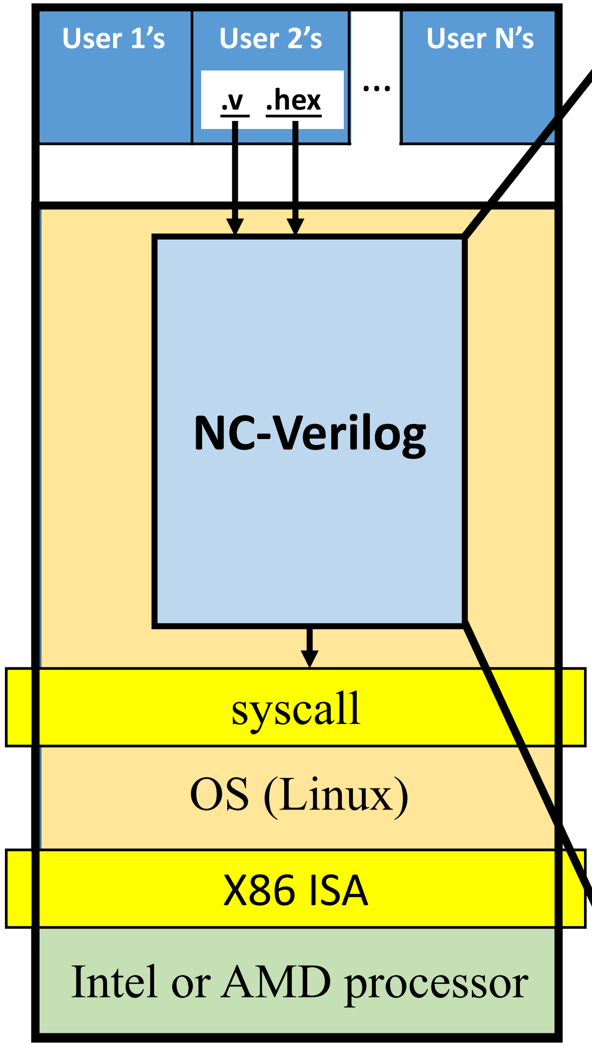
Design & Testbench

- **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)

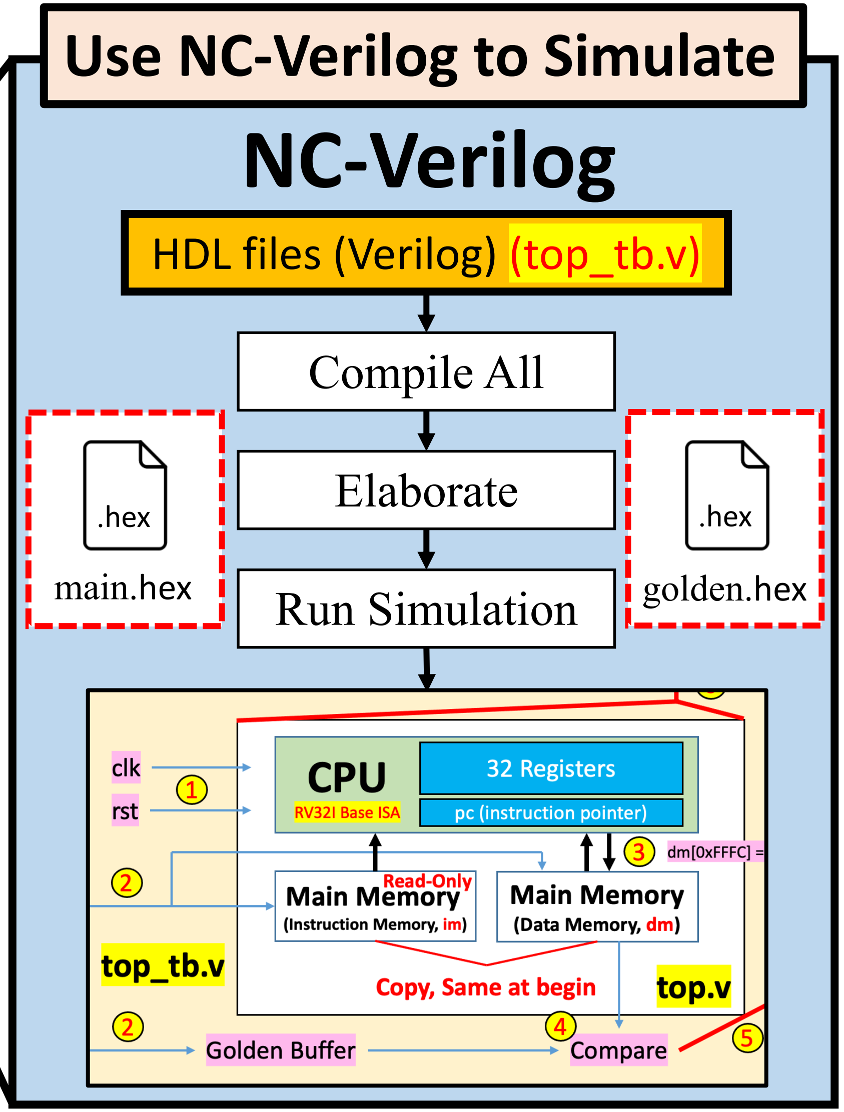




NC-Verilog



Server



RISC-V Machine (NC-Verilog Simulation)

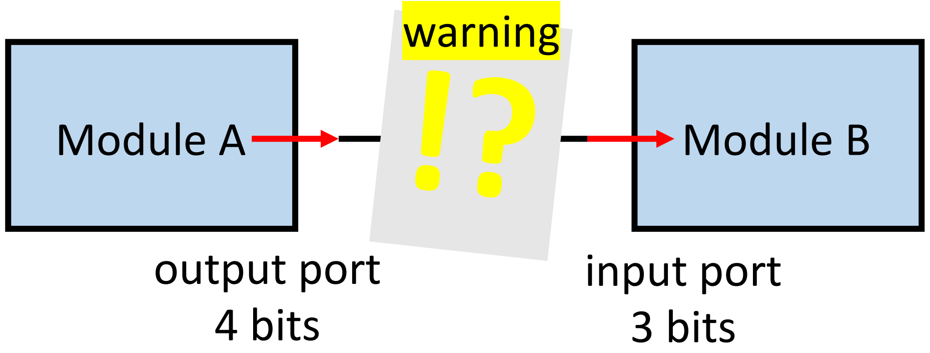
1. Compilation (Finished by TA)

Analyze the source code for syntax and semantic errors

```
$ ncvlog top_tb.v
```

2. Elaboration (Finished by TA)

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



```
$ ncelab top_tb –access +r
```

3. Simulation (You just need to do this)

Run with the given execution model generated after elaboration

```
$ ncsim top_tb
```

Program & RISC-V Toolchain

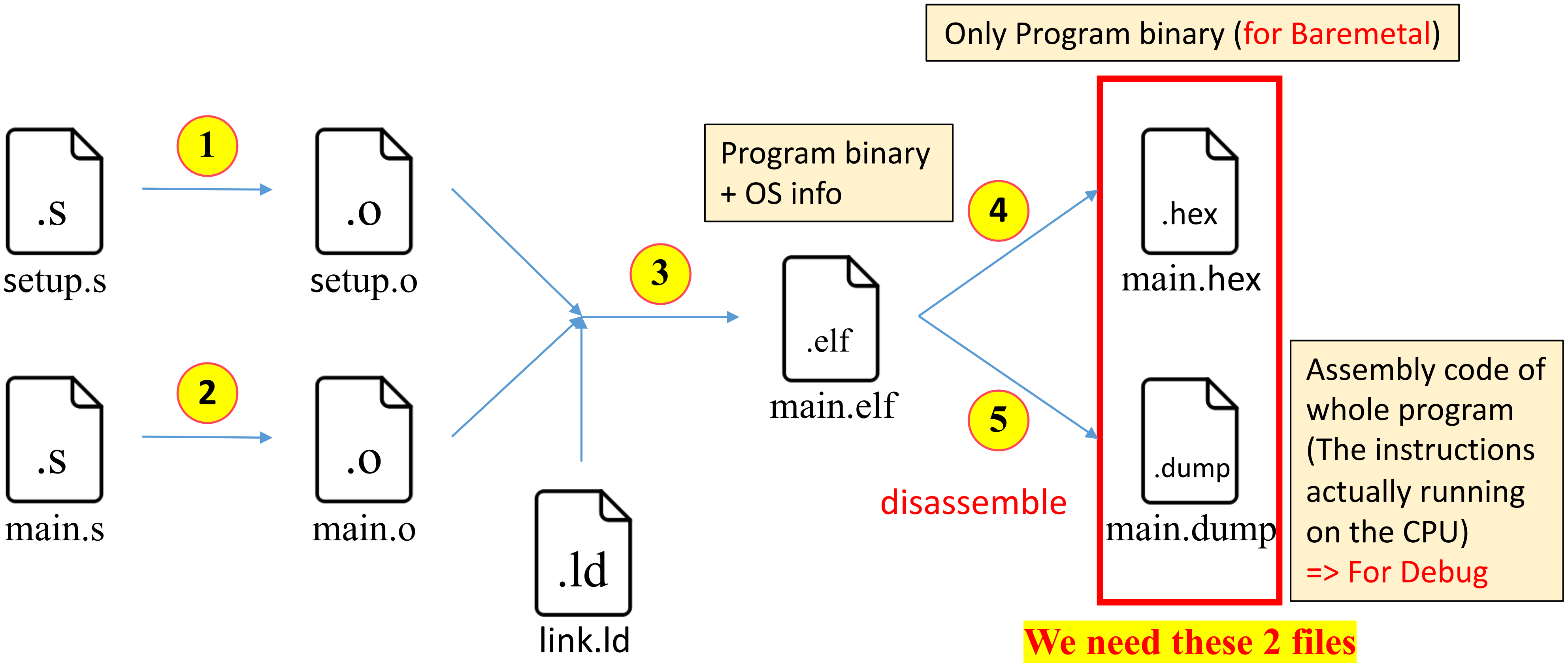
RISC-V Toolchain Workflow

Program Environment

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

Main program

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



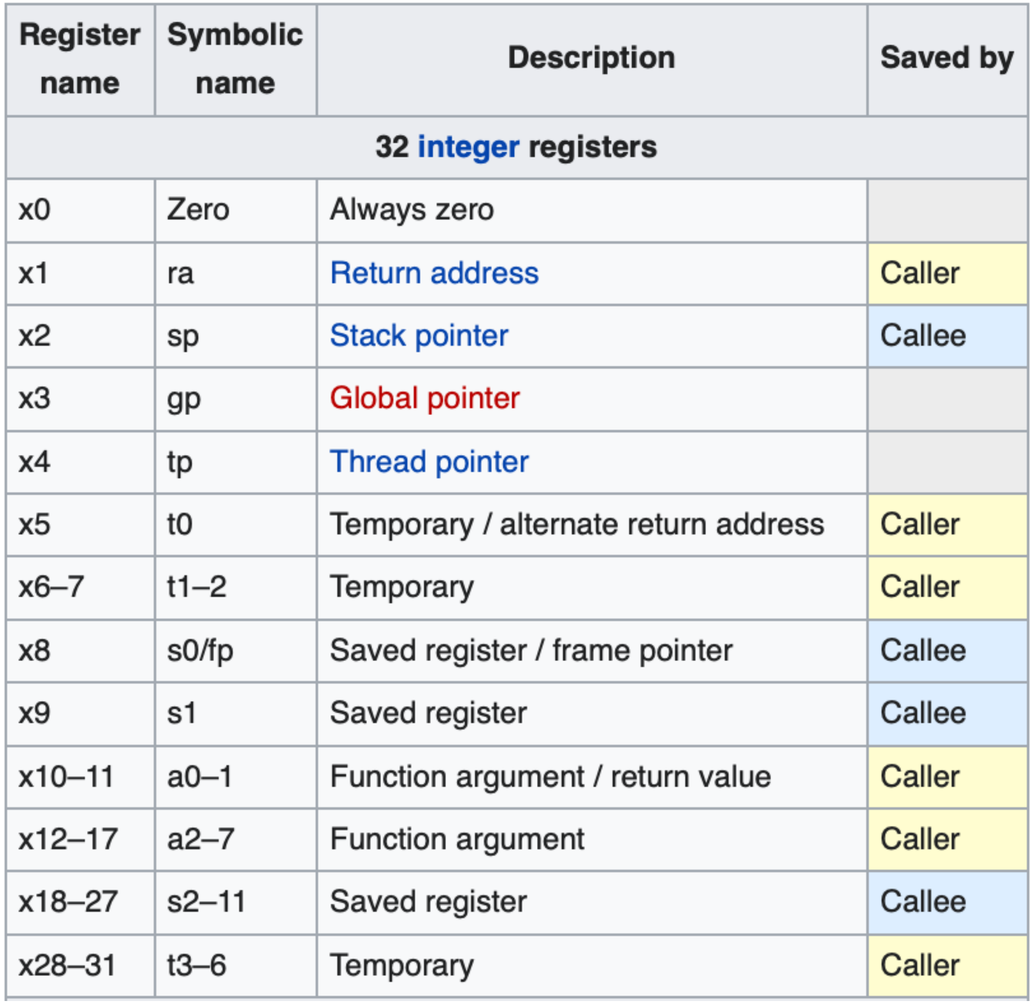
Only Program binary (for Baremetal)

Program binary + OS info

Assembly code of whole program (The instructions actually running on the CPU) => For Debug

We need these 2 files

RV32I - Registers

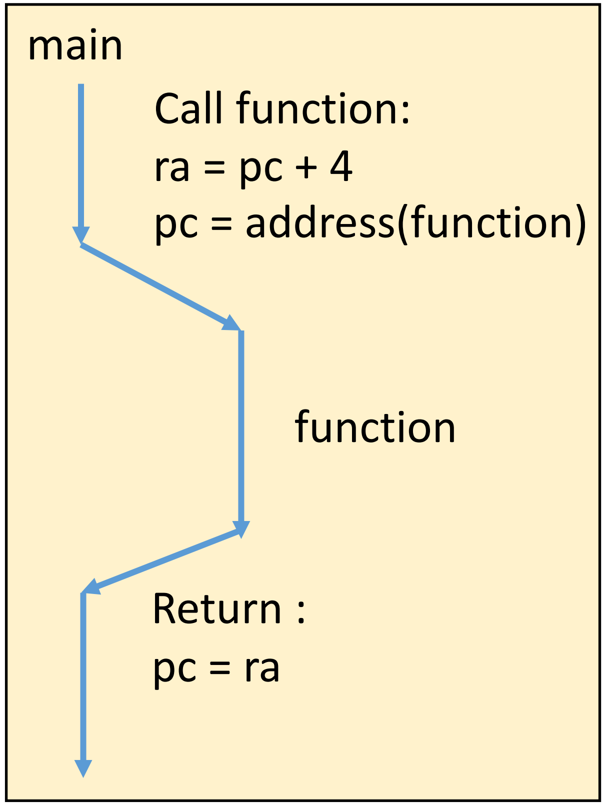


Register name	Symbolic name	Description	Saved by
32 integer registers			
x0	Zero	Always zero	
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	
x4	tp	Thread pointer	
x5	t0	Temporary / alternate return address	Caller
x6–7	t1–2	Temporary	Caller
x8	s0/fp	Saved register / frame pointer	Callee
x9	s1	Saved register	Callee
x10–11	a0–1	Function argument / return value	Caller
x12–17	a2–7	Function argument	Caller
x18–27	s2–11	Saved register	Callee
x28–31	t3–6	Temporary	Caller

Additional Register : **Program Counter**
points to the current fetched instruction

RV32I

- 32 Registers, each are 32 bits
- x : Integer Registers
  - x0(Zero) : always zero
  - x1(ra) : save return address
  - x2(sp) : point to top of stack

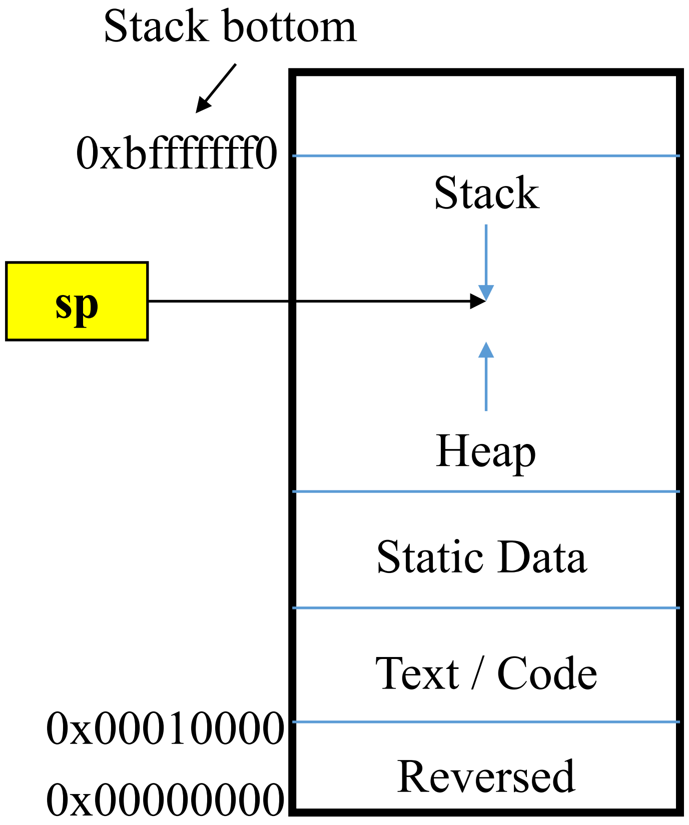


main

Call function:
ra = pc + 4
pc = address(function)

function

Return :
pc = ra



Program Structure (template)

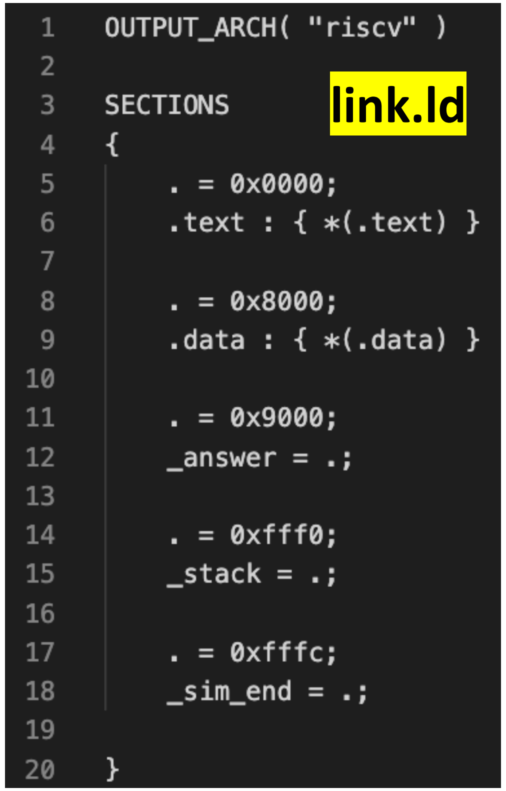
- We will use this template throughout the semester

Program Environment (setup.s)

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

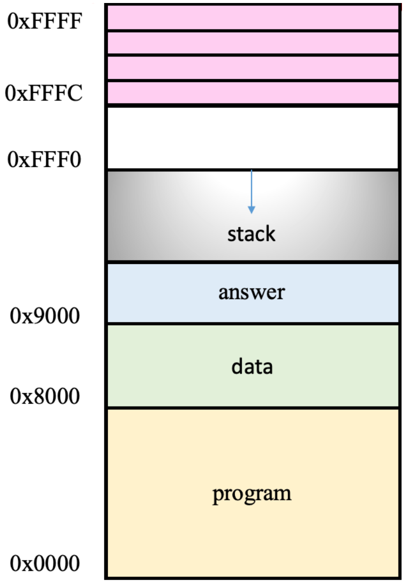
Main program (main.s)

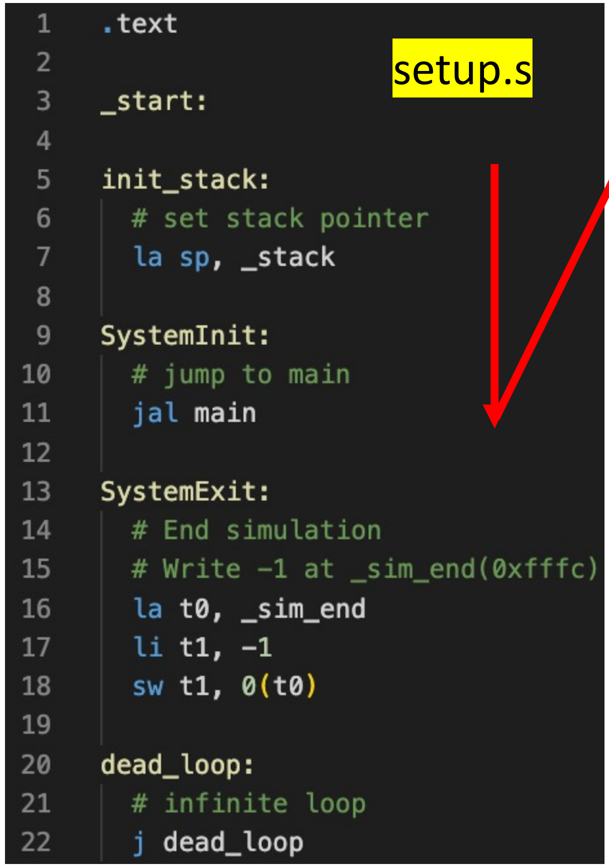
- Get address of answer
- Execute main program
- Write result to answer
- Return to setup



link.ld

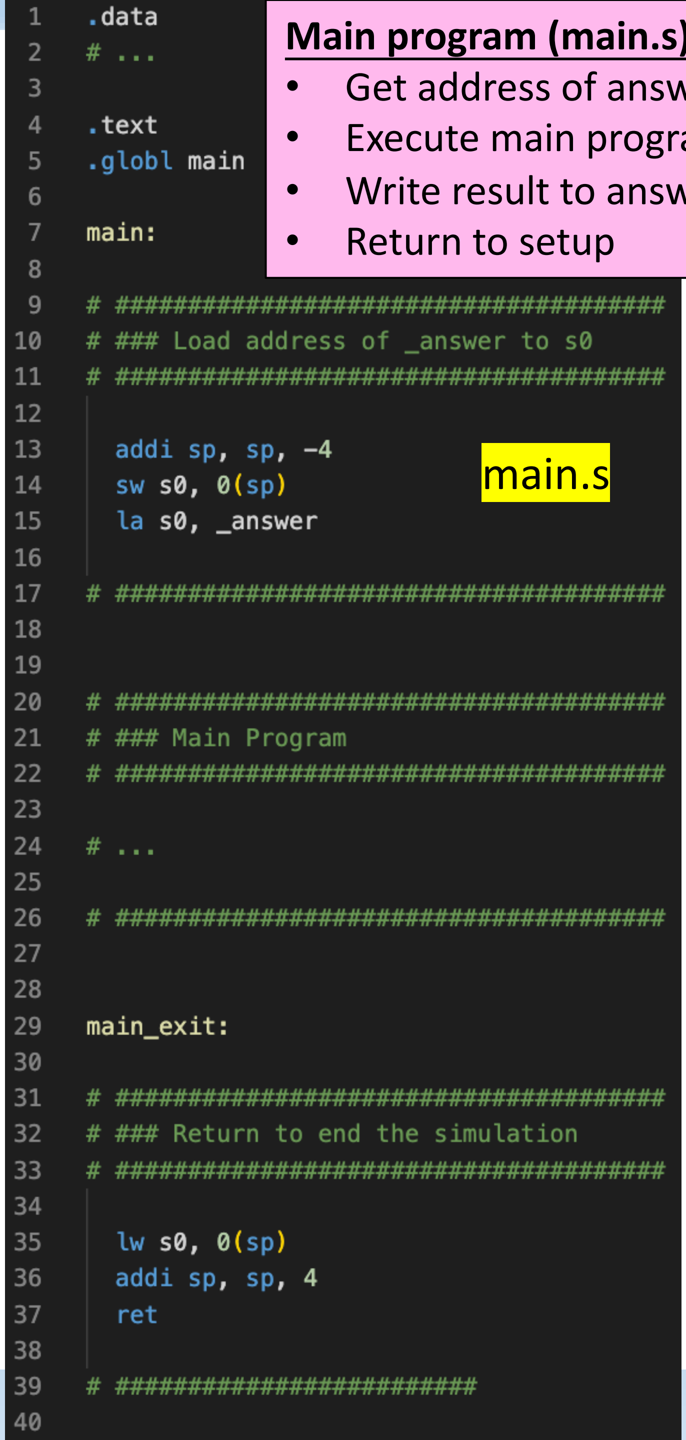
```
OUTPUT_ARCH( "riscv" )

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

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

    . = 0x9000;
    _answer = .;

    . = 0xfff0;
    _stack = .;

    . = 0xfffc;
    _sim_end = .;

}
```





setup.s

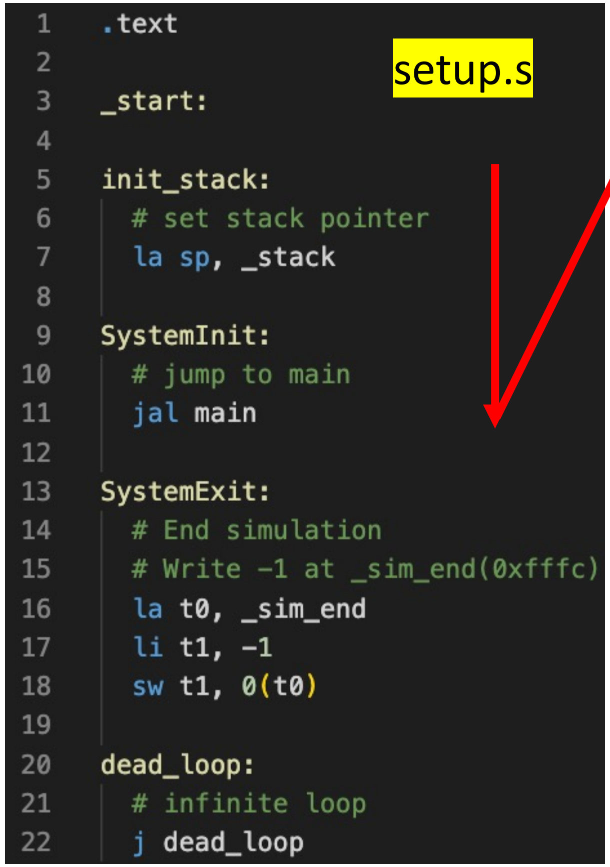
```
.text

_start:

init_stack:
  # set stack pointer
  la sp, _stack

SystemInit:
  # jump to main
  jal main

SystemExit:
  # End simulation
  # Write -1 at _sim_end(0xfffc)
  la t0, _sim_end
  li t1, -1
  sw t1, 0(t0)

dead_loop:
  # infinite loop
  j dead_loop
```



main.s

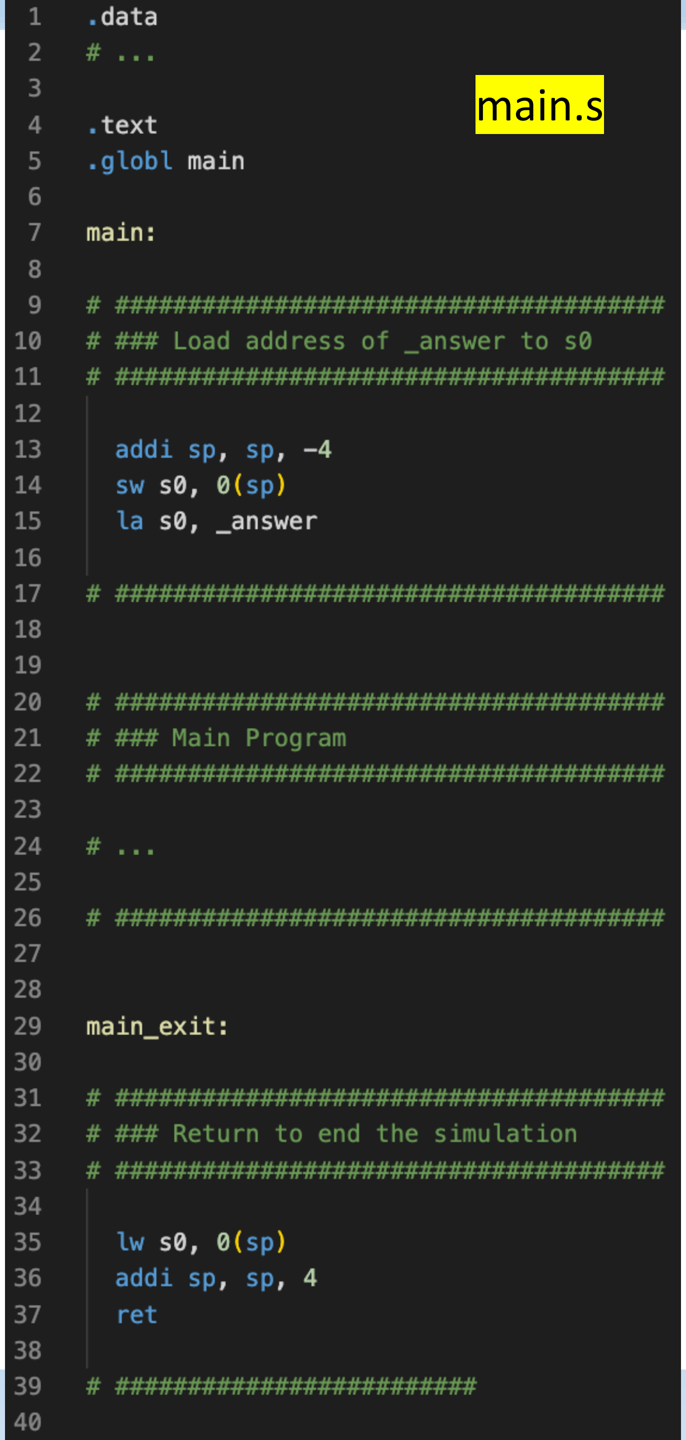
```
.data
# ...

.text
.globl main

main:

# ######################################
# ### Load address of _answer to s0
# ######################################

  addi sp, sp, -4
  sw s0, 0(sp)
  la s0, _answer

# ######################################


# ######################################
# ### Main Program
# ######################################

# ...

# ######################################


main_exit:

# ######################################
# ### Return to end the simulation
# ######################################

  lw s0, 0(sp)
  addi sp, sp, 4
  ret

# #########################

```

What does the program look like after linking the two files?



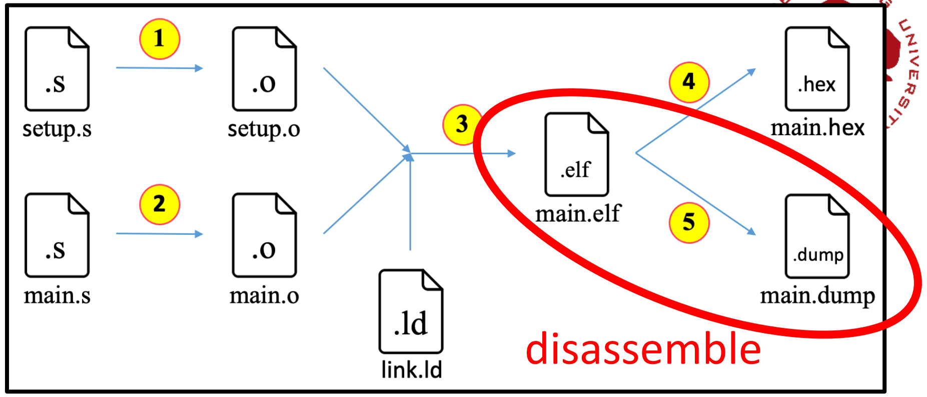
setup.s

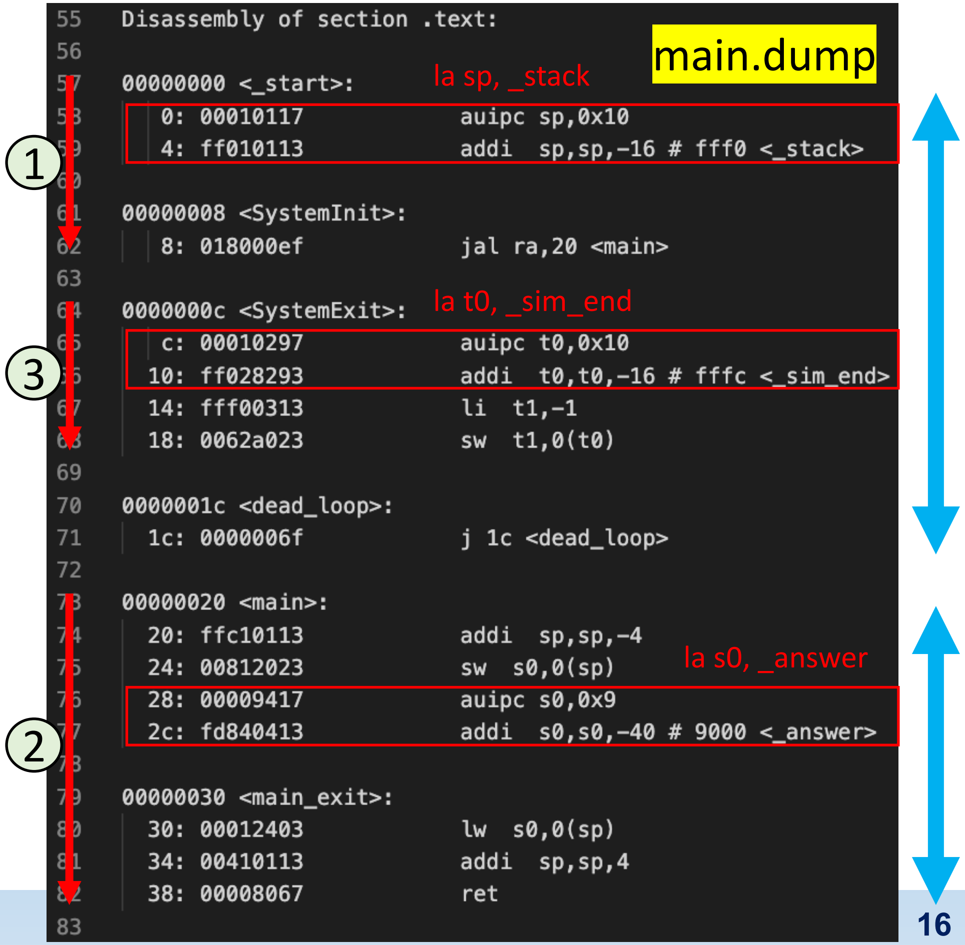
```
.text

_start:

init_stack:
    # set stack pointer
    la sp, _stack

SystemInit:
    # jump to main
    jal main

SystemExit:
    # End simulation
    # Write -1 at _sim_end(0xfffc)
    la t0, _sim_end
    li t1, -1
    sw t1, 0(t0)

dead_loop:
    # infinite loop
    j dead_loop
```



main.s

```
.data
# ...

.text
.globl main

main:

# ######################################
# ### Load address of _answer to s0
# ######################################

    addi sp, sp, -4
    sw s0, 0(sp)
    la s0, _answer

# ######################################


# ######################################
# ### Main Program
# ######################################

# ...

# ######################################


main_exit:

# ######################################
# ### Return to end the simulation
# ######################################

    lw s0, 0(sp)
    addi sp, sp, 4
    ret

# #########################
```

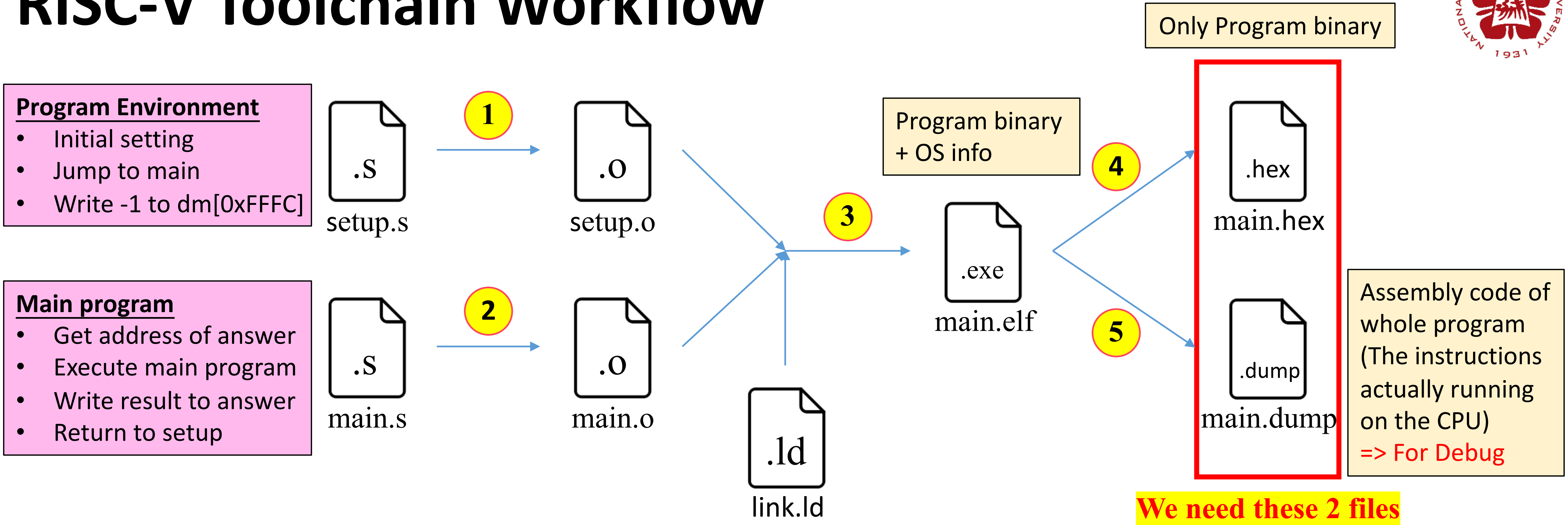




main.dump

```
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>

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

RISC-V Toolchain Workflow

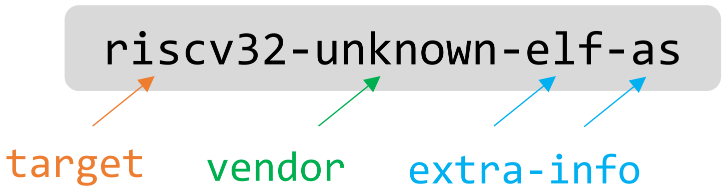


1. riscv32-unknown-elf-as -march=rv32i -mabi=ilp32 [setup.s] -o [setup.o]
2. riscv32-unknown-elf-as -march=rv32i -mabi=ilp32 [main.s] -o [main.o]
3. riscv32-unknown-elf-ld -b elf32-littleriscv -T [link.ld] [setup.o] [main.o] -o [main.elf]
4. riscv32-unknown-elf-objcopy -O verilog [main.elf] [main.hex]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

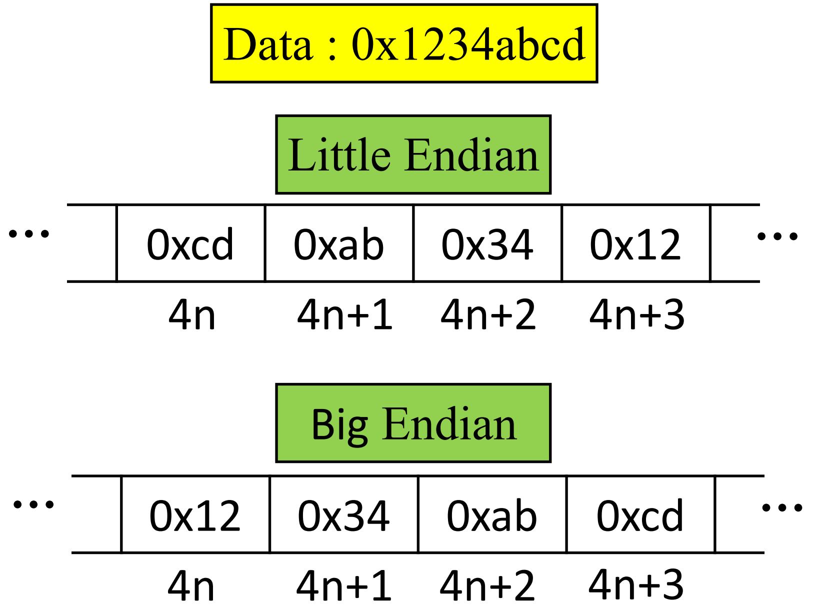
Target Triplet

Reference : link

- We can find a special command string `$ riscv32-unknown-elf` when using RISC-V Toolchain
- This is called "**Target Triplet**", whose structure is `machine-vendor-operatingsystem`
- Common Rule is `<target>[<endian>][-<vendor>]-<os>[-<extra-info>]`
    - <target> : Architecture (e.g., riscv32 , x86_64 , arm ,…)
    - [<endian>] : little / big-endian (not must)
    - [-<vendor>] : 供應商名稱 (not must)
    - -<os>: name of Operating System (sometimes can be ignored)
    - [-<extra-info>] : Usually descript ABI or Library
- Example :



```
riscv32-unknown-elf-as
```



Data : 0x1234abcd

Little Endian

…	0xcd	0xab	0x34	0x12	…
	4n	4n+1	4n+2	4n+3	

Big Endian

…	0x12	0x34	0xab	0xcd	…
	4n	4n+1	4n+2	4n+3	

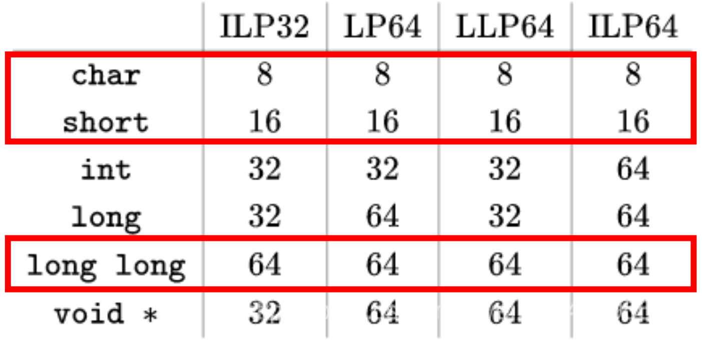
Command Explanation

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]
```

- **-march** : Set RISC-V ISA
- **-mabi** : Set RISC-V ABI

	int	long	pointer
ilp32/ilp32f/ilp32d	32-bit	32-bit	32-bit
lp64/lp64f/lp64d	32-bit	64-bit	64-bit

RISC-V 只支援這六種



	ILP32	LP64	LLP64	ILP64	
char	8	8	8	8	fixed
short	16	16	16	16	fixed
int	32	32	32	64	
long	32	64	32	64	
long long	64	64	64	64	fixed
void *	32	64	64	64	

LLP = long long & pointer

3.
```
$ riscv32-unknown-elf-ld -b elf32-littleriscv -T [link.ld] [setup.o] [main.o] -o [main.elf]
```

- **-b** : Specify target, elf32-littleriscv => 32 bits, little endian
- **-T** : Read linker script

The order is important

4.
```
$ riscv32-unknown-elf-objcopy -O verilog [main.elf] [main.hex]
```

- **-O** : Create an output file in format, verilog => the format that verilog simulator can understand

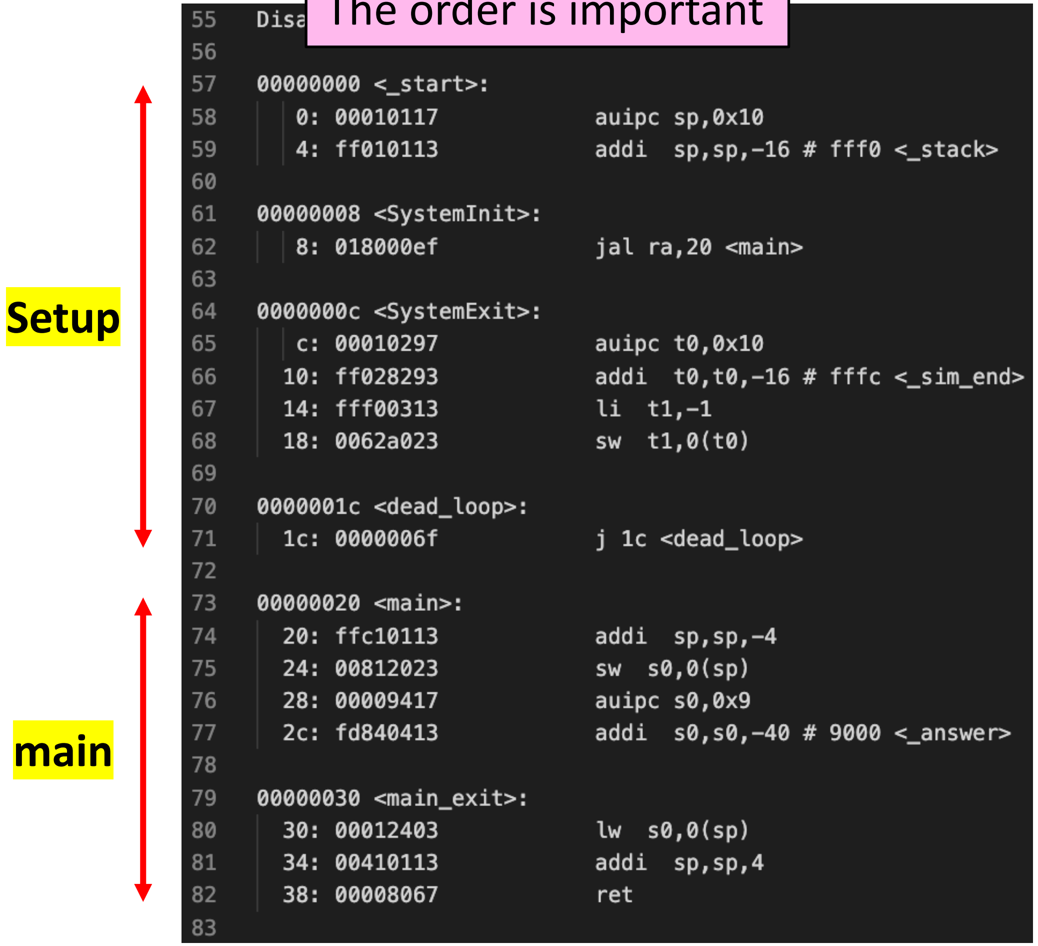
5.
```
$ riscv32-unknown-elf-objdump -xsd [main.elf] > [main.dump]
```

- **-x** : All headers, Display the contents of all headers
- **-s** : Full contents, Display the full contents of all sections requested
- **-d** : Disassemble, Display assembler contents of executable sections

Command Explanation

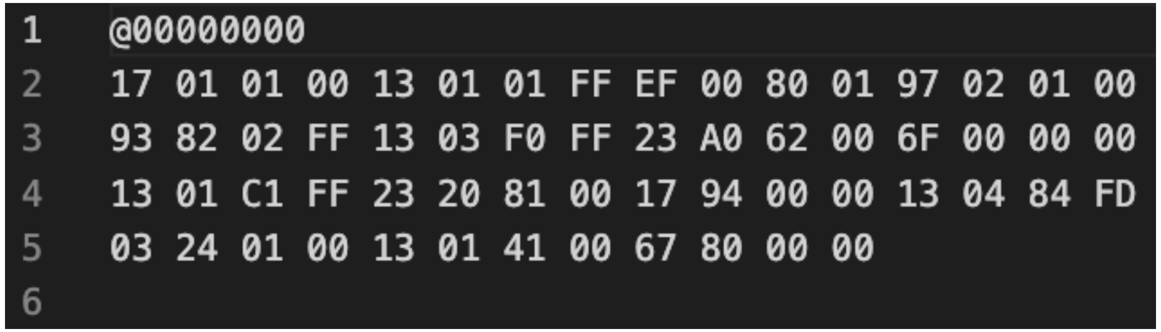
```
3.  $ riscv32-unknown-elf-ld -b elf32-littleriscv
    -T link.ld [setup.o] [main.o] -o [main.elf]
```

The order is important



```
Disa

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

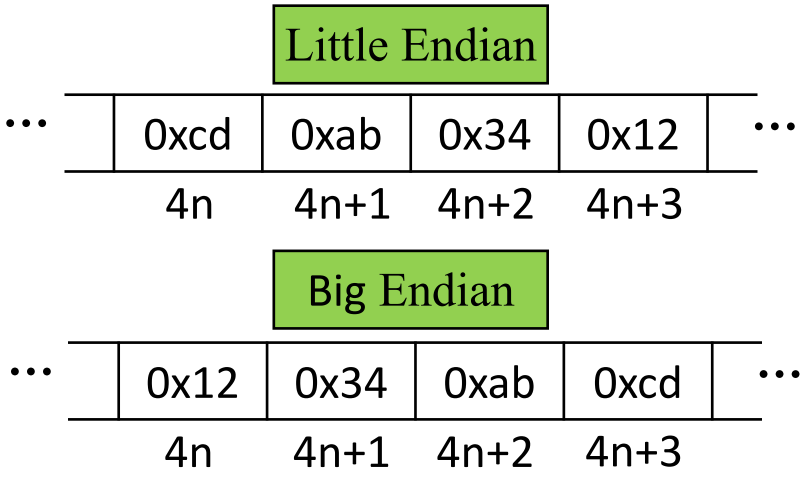
```
4.  $ riscv32-unknown-elf-objcopy –O verilog [main.elf] [main.hex]
```



```
@00000000
17 01 01 00 13 01 01 FF EF 00 80 01 97 02 01 00
93 82 02 FF 13 03 F0 FF 23 A0 62 00 6F 00 00 00
13 01 C1 FF 23 20 81 00 17 94 00 00 13 04 84 FD
03 24 01 00 13 01 41 00 67 80 00 00
```

- Start from address 0
- Byte address
- Little endian

Data : 0x1234abcd



Little Endian

…	0xcd	0xab	0x34	0x12	…
	4n	4n+1	4n+2	4n+3	

Big Endian

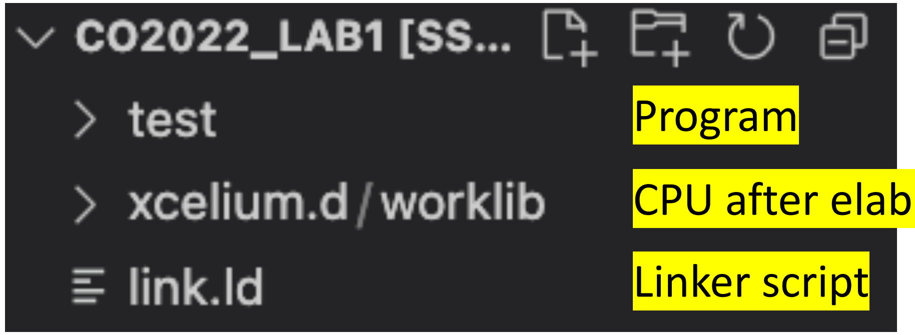
…	0x12	0x34	0xab	0xcd	…
	4n	4n+1	4n+2	4n+3	

Experiment

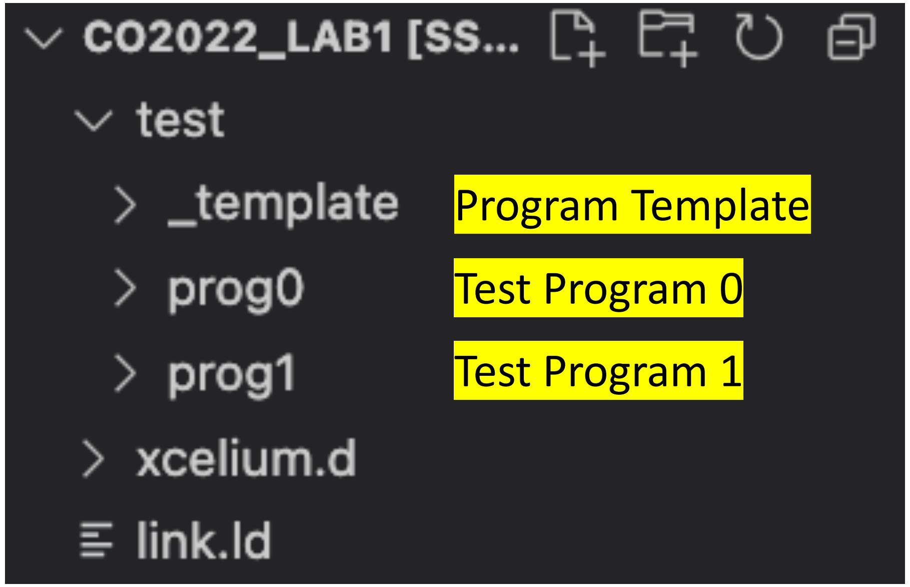
File Structure

Review 3 steps

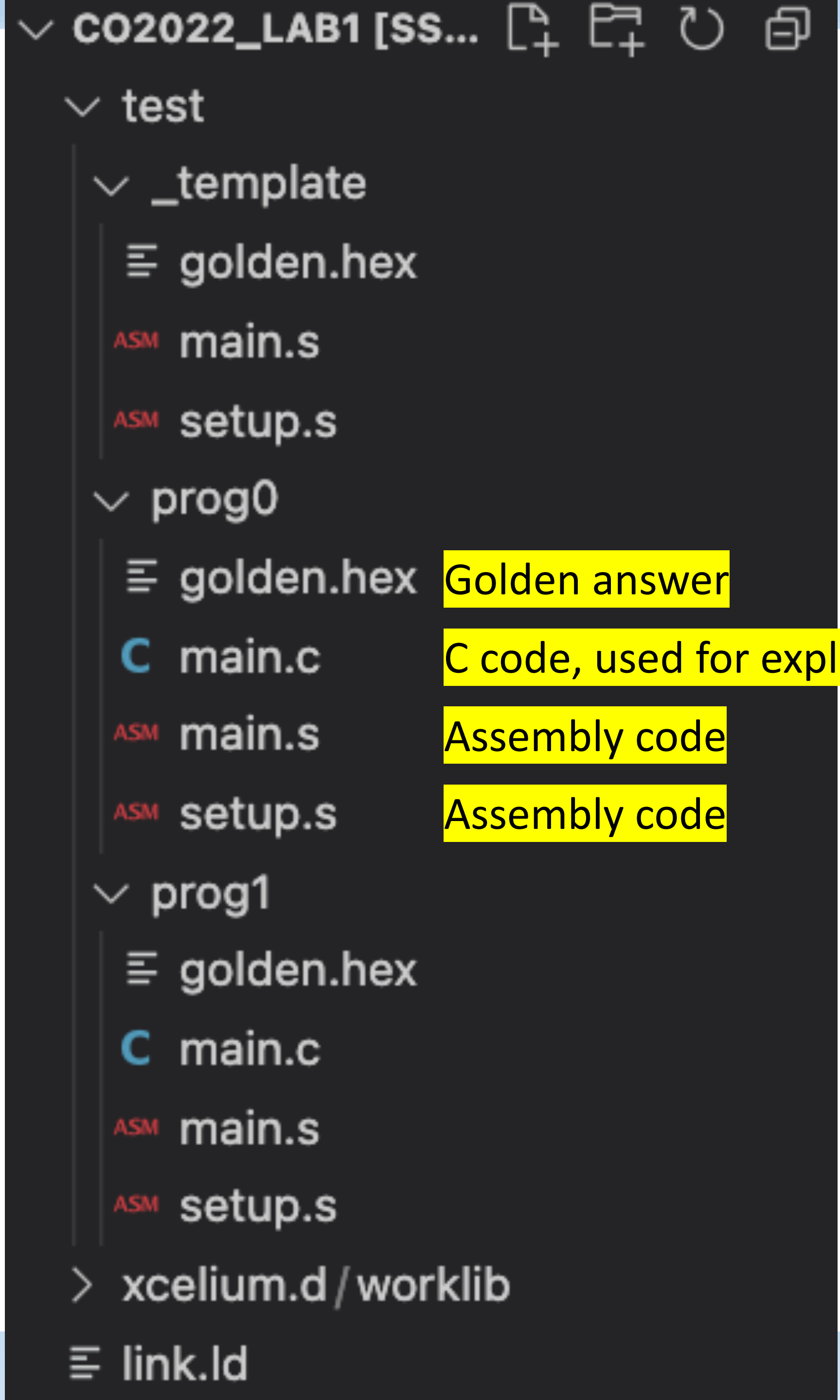
1. Compilation
2. Elaboration
3. Simulation



```
CO2022_LAB1 [SS...
  > test              Program
  > xcelium.d / worklib   CPU after elaborating
    link.ld           Linker script
```

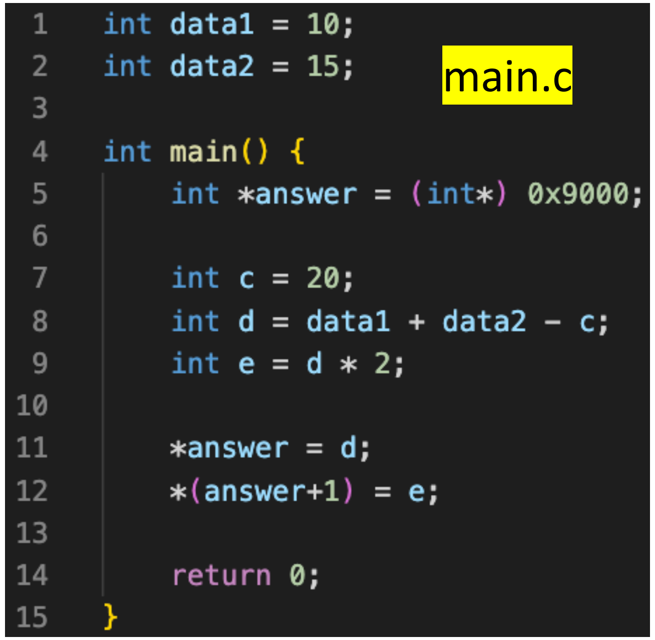


```
CO2022_LAB1 [SS...
  test
    > _template       Program Template
    > prog0           Test Program 0
    > prog1           Test Program 1
  > xcelium.d
    link.ld
```



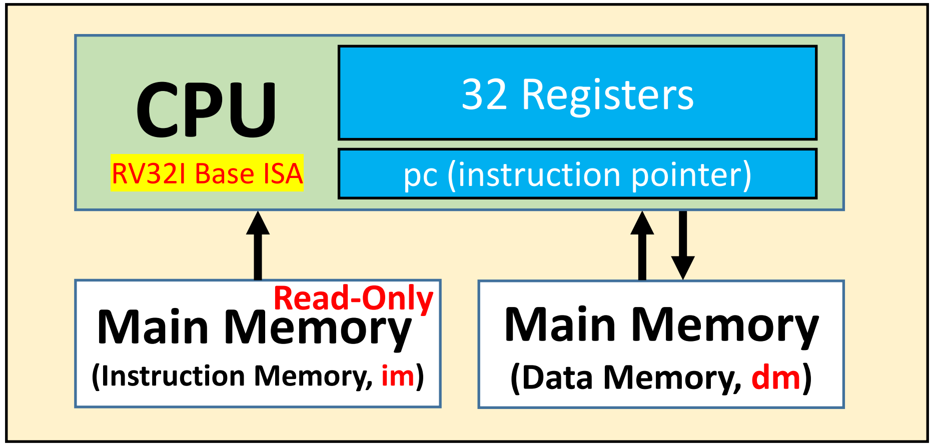
```
CO2022_LAB1 [SS...
  test
    _template
      golden.hex
      main.s
      setup.s
    prog0
      golden.hex      Golden answer
      main.c          C code, used for explain program
      main.s          Assembly code
      setup.s         Assembly code
    prog1
      golden.hex
      main.c
      main.s
      setup.s
  > xcelium.d / worklib
    link.ld
```

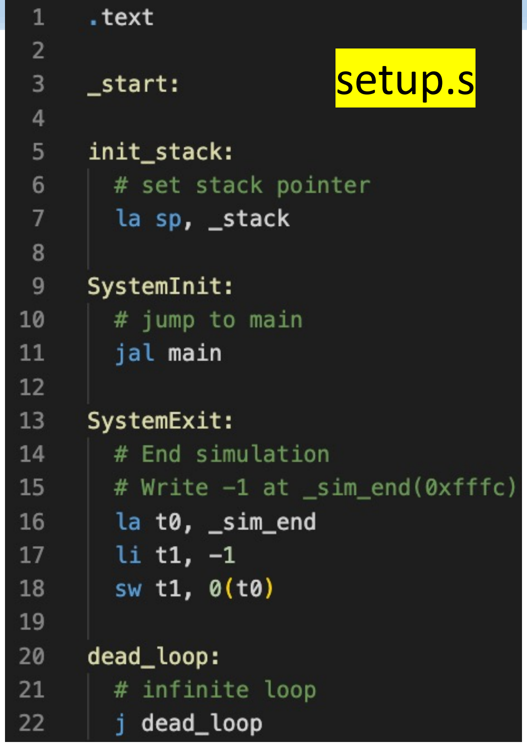
Test Program (prog0)



main.c

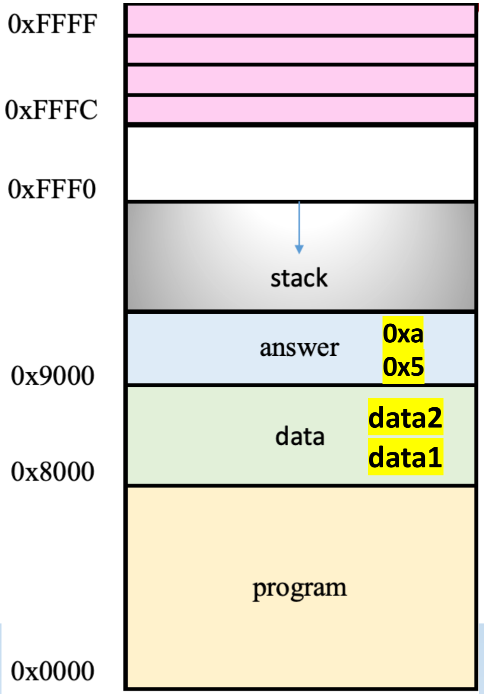
```c
int data1 = 10;
int data2 = 15;

int main() {
    int *answer = (int*) 0x9000;

    int c = 20;
    int d = data1 + data2 - c;
    int e = d * 2;

    *answer = d;
    *(answer+1) = e;

    return 0;
}
```

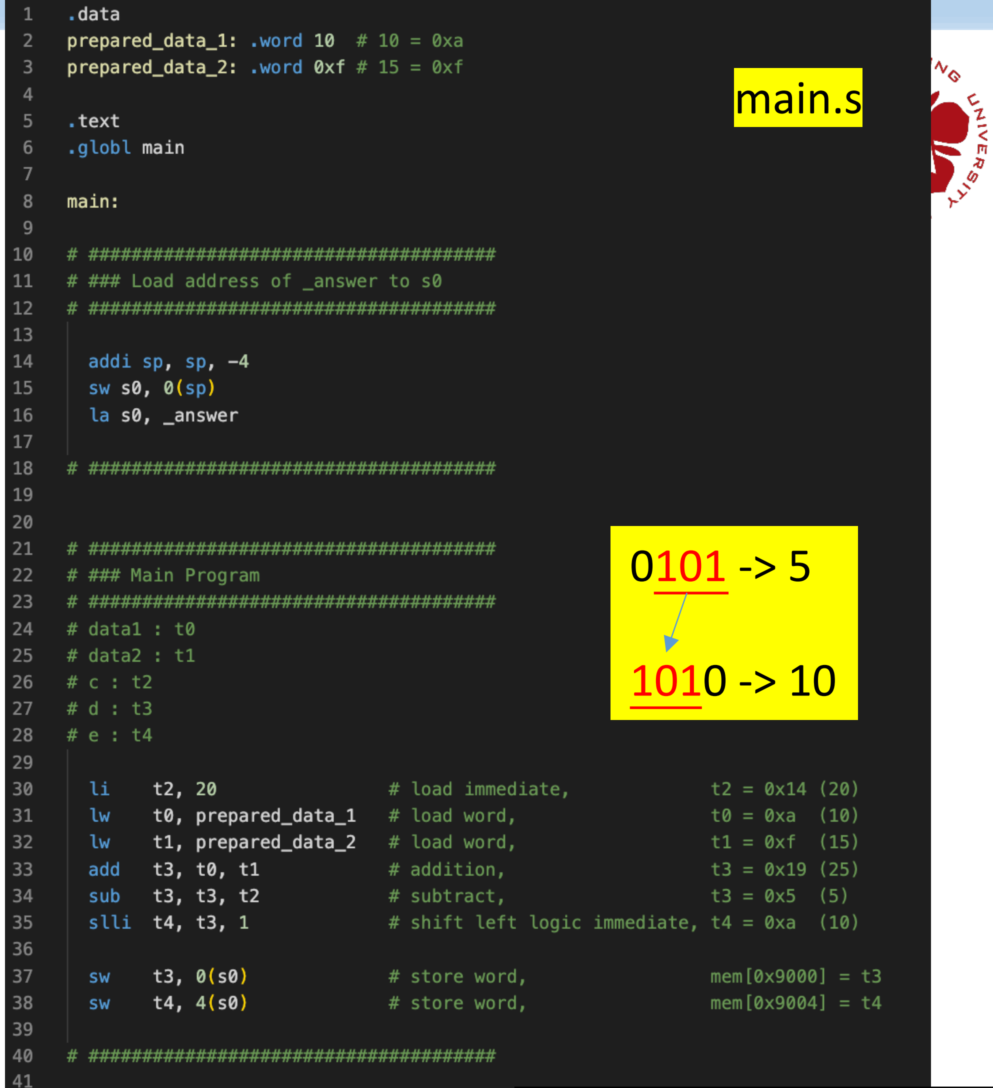




setup.s

```
.text

_start:

init_stack:
    # set stack pointer
    la sp, _stack

SystemInit:
    # jump to main
    jal main

SystemExit:
    # End simulation
    # Write -1 at _sim_end(0xfffc)
    la t0, _sim_end
    li t1, -1
    sw t1, 0(t0)

dead_loop:
    # infinite loop
    j dead_loop
```



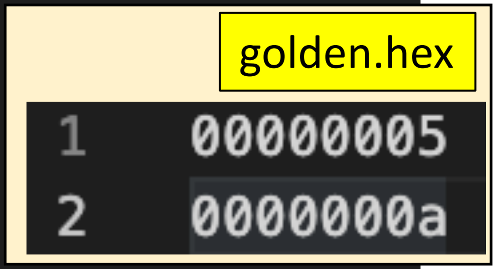


main.s

```
.data
prepared_data_1: .word 10  # 10 = 0xa
prepared_data_2: .word 0xf # 15 = 0xf

.text
.globl main

main:

# #####################################
# ### Load address of _answer to s0
# #####################################

  addi sp, sp, -4
  sw s0, 0(sp)
  la s0, _answer

# #####################################


# #####################################
# ### Main Program
# #####################################
# data1 : t0
# data2 : t1
# c : t2
# d : t3
# e : t4

  li    t2, 20               # load immediate,             t2 = 0x14 (20)
  lw    t0, prepared_data_1  # load word,                  t0 = 0xa  (10)
  lw    t1, prepared_data_2  # load word,                  t1 = 0xf  (15)
  add   t3, t0, t1           # addition,                   t3 = 0x19 (25)
  sub   t3, t3, t2           # subtract,                   t3 = 0x5  (5)
  slli  t4, t3, 1            # shift left logic immediate, t4 = 0xa  (10)

  sw    t3, 0(s0)            # store word,                 mem[0x9000] = t3
  sw    t4, 4(s0)            # store word,                 mem[0x9004] = t4

# #####################################


main_exit:

# #####################################
# ### Return to end the simulation
# #####################################

  lw s0, 0(sp)
  addi sp, sp, 4
  ret

# #####################################
```

0101 -> 5

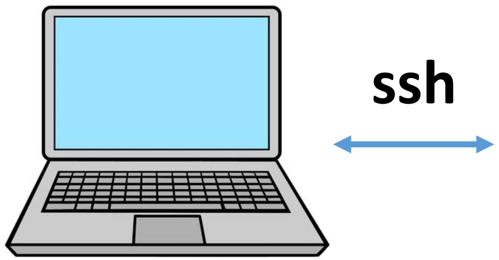
1010 -> 10

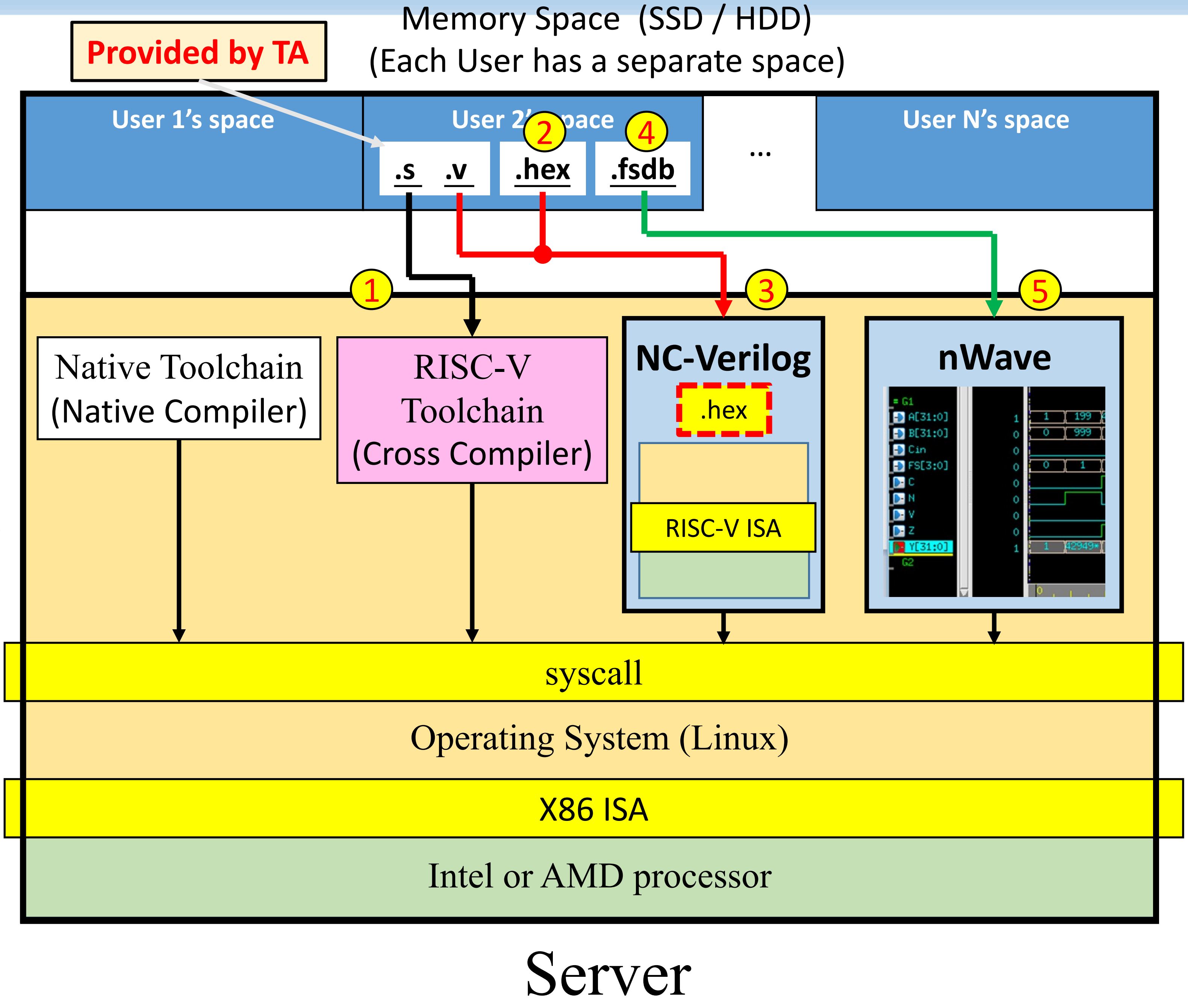


golden.hex

```
1    00000005
2    0000000a
```

Steps

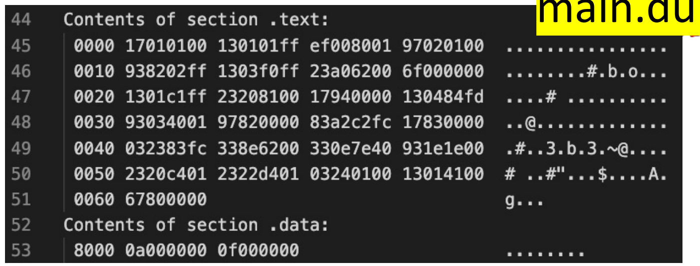




1. Cross-compile source files using RISC-V Toolchain => Generate .hex
2. Simulate with NC-Verilog => Generate .fsdb
3. Check waveform with nWave

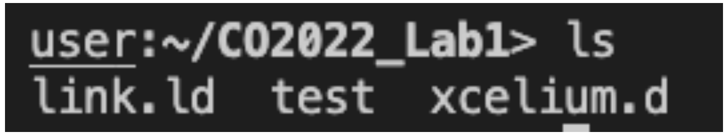
Step 1 - Use RISC-V Toolchain to generate “main.hex” & “main.dump”

main.dump



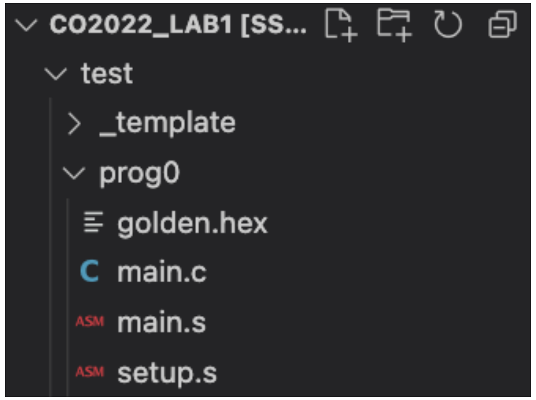
```
Contents of section .text:
 0000 17010100 130101ff ef008001 97020100  ................
 0010 938202ff 1303f0ff 23a06200 6f000000  ........#.b.o...
 0020 1301c1ff 23208100 17940000 130484fd  ....# ..........
 0030 93034001 97820000 83a2c2fc 17830000  ..@.............
 0040 032383fc 338e6200 330e7e40 931e1e00  .#..3.b.3.~@....
 0050 2320c401 2322d401 03240100 13014100  # ..#"...$....A.
 0060 67800000                             g...
Contents of section .data:
 8000 0a000000 0f000000                    ........
```

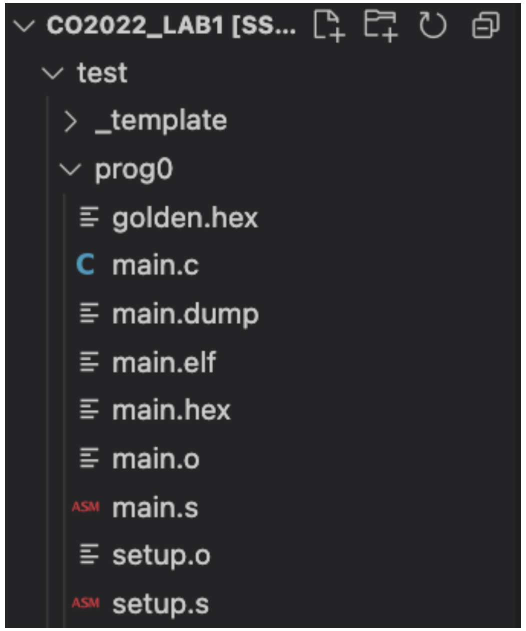
1. Use “cd(change directory)” to change the path to “CO2022_Lab1” folder

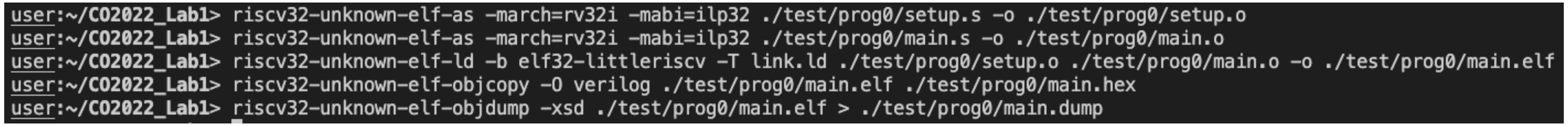


```
user:~/CO2022_Lab1> ls
link.ld  test  xcelium.d
```

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

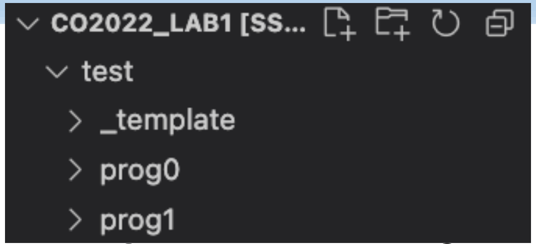






```
user:~/CO2022_Lab1> riscv32-unknown-elf-as -march=rv32i -mabi=ilp32 ./test/prog0/setup.s -o ./test/prog0/setup.o
user:~/CO2022_Lab1> riscv32-unknown-elf-as -march=rv32i -mabi=ilp32 ./test/prog0/main.s -o ./test/prog0/main.o
user:~/CO2022_Lab1> riscv32-unknown-elf-ld -b elf32-littleriscv -T link.ld ./test/prog0/setup.o ./test/prog0/main.o -o ./test/prog0/main.elf
user:~/CO2022_Lab1> riscv32-unknown-elf-objcopy -O verilog ./test/prog0/main.elf ./test/prog0/main.hex
user:~/CO2022_Lab1> riscv32-unknown-elf-objdump -xsd ./test/prog0/main.elf > ./test/prog0/main.dump
```

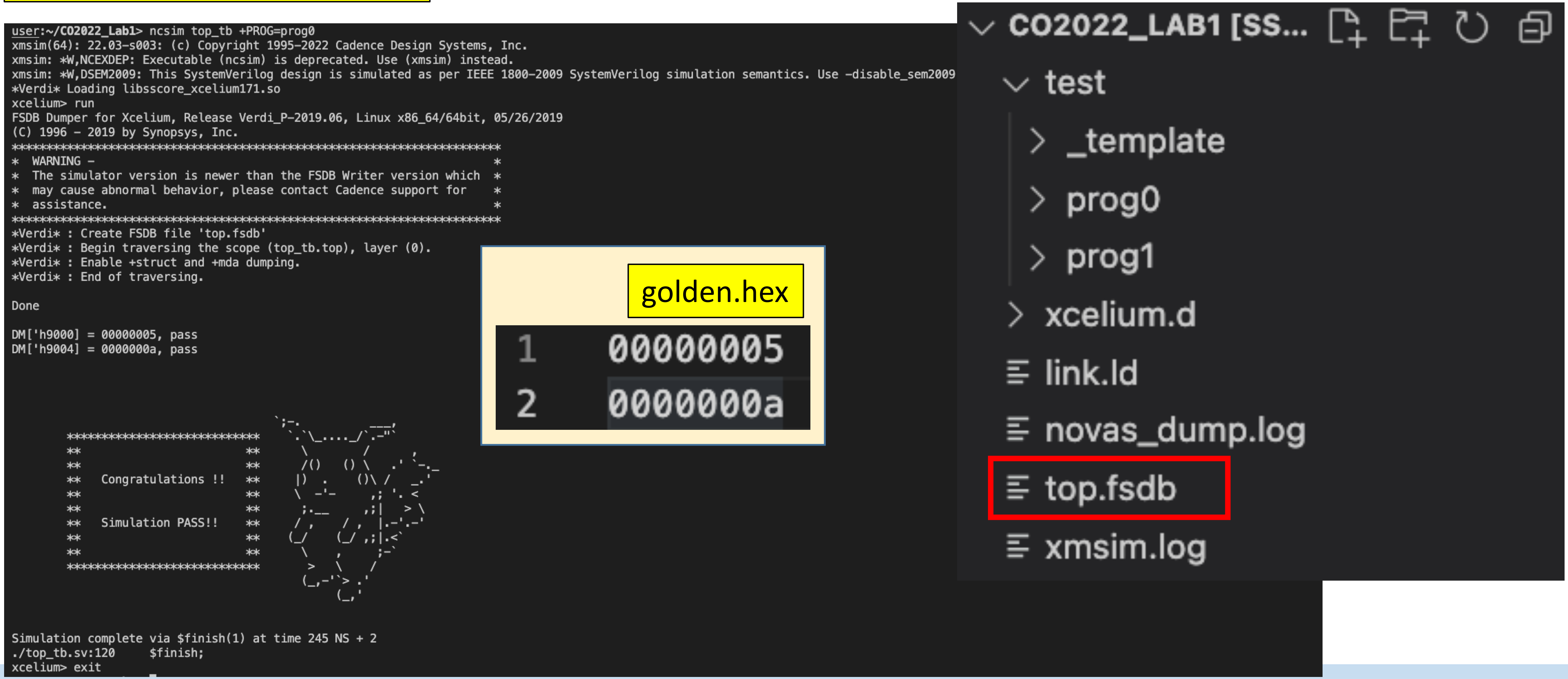
Step 2



2. **Use NC-Verilog to simulate a CPU running the program and producing waveform file**

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

Specify a directory, it will find the main.hex & golden.hex under the directory



```
user:~/CO2022_Lab1> ncsim top_tb +PROG=prog0
xmsim(64): 22.03-s003: (c) Copyright 1995-2022 Cadence Design Systems, Inc.
xmsim: *W,NCEXDEP: Executable (ncsim) is deprecated. Use (xmsim) instead.
xmsim: *W,DSEM2009: This SystemVerilog design is simulated as per IEEE 1800-2009 SystemVerilog simulation semantics. Use -disable_sem2009
*Verdi* Loading libsscore_xcelium171.so
xcelium> run
FSDB Dumper for Xcelium, Release Verdi_P-2019.06, Linux x86_64/64bit, 05/26/2019
(C) 1996 - 2019 by Synopsys, Inc.
*********************************************************************
*  WARNING -                                                        *
*  The simulator version is newer than the FSDB Writer version which *
*  may cause abnormal behavior, please contact Cadence support for   *
*  assistance.                                                      *
*********************************************************************
*Verdi* : Create FSDB file 'top.fsdb'
*Verdi* : Begin traversing the scope (top_tb.top), layer (0).
*Verdi* : Enable +struct and +mda dumping.
*Verdi* : End of traversing.

Done

DM['h9000] = 00000005, pass
DM['h9004] = 0000000a, pass


                                       `;-.          ___,
        ****************************     `.`\_...._/`.-"`
        **                        **       \        /      ,
        **                        **       /()   () \    .' `-._
        **   Congratulations !!   **      |)  .    ()\  /   _.'
        **                        **      \  -'-     ,; '. <
        **                        **       ;.__     ,;|   > \
        **   Simulation PASS!!    **      / ,    / ,  |.-'.-'
        **                        **     (_/    (_/ ,;|.<`
        **                        **       \    ,     ;-`
        ****************************        >   \    /
                                           (_,-'`> .'
                                                (_,'

Simulation complete via $finish(1) at time 245 NS + 2
./top_tb.sv:120     $finish;
xcelium> exit
```

golden.hex

```
00000005
0000000a
```

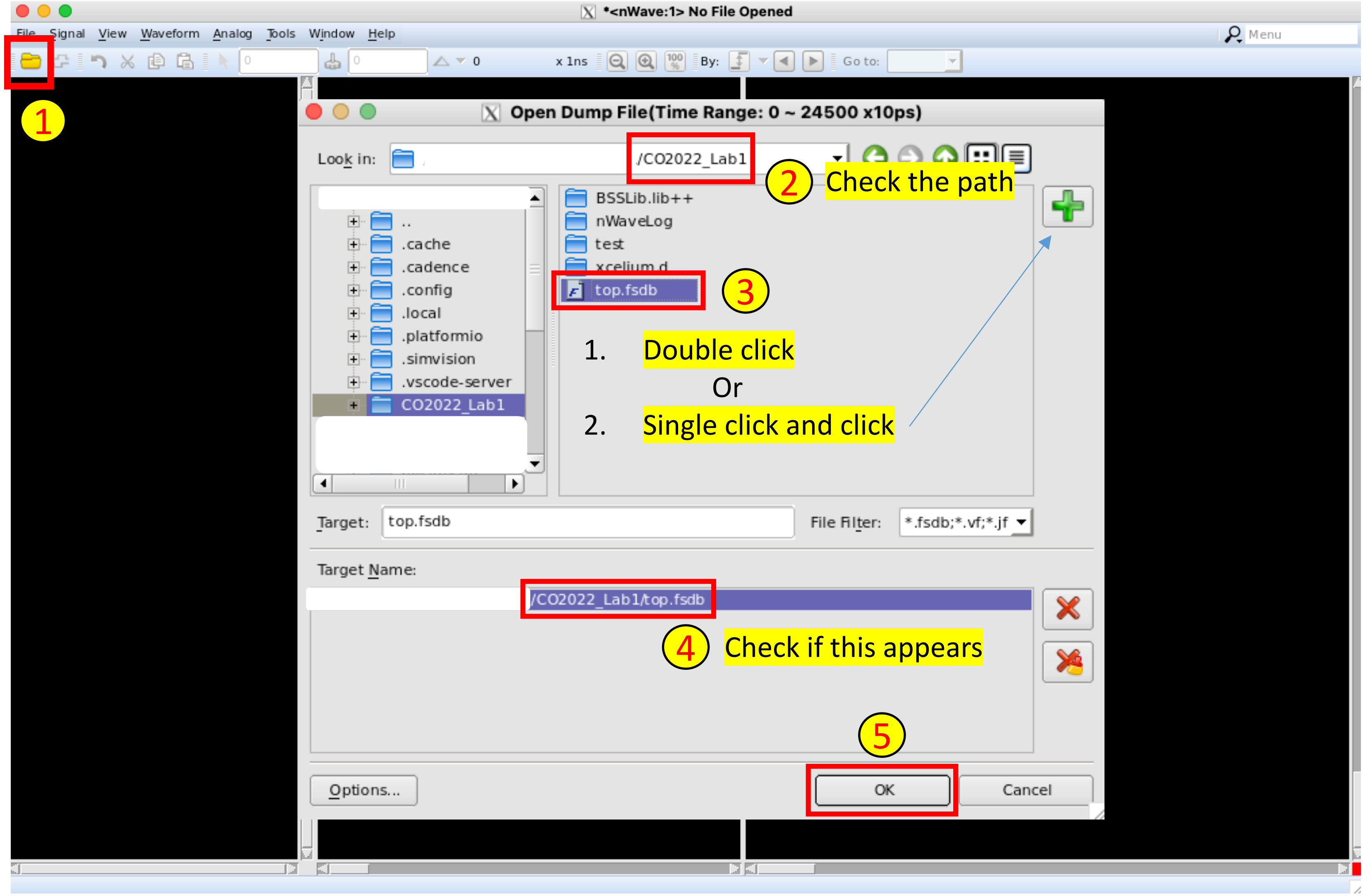
Step 3 - Use nWave to Check waveform : Open Waveform File (1/7)

$ nWave

or

$ nWave &

& :
means to execute it in the background. Meanwhile, you can do other things through the terminal

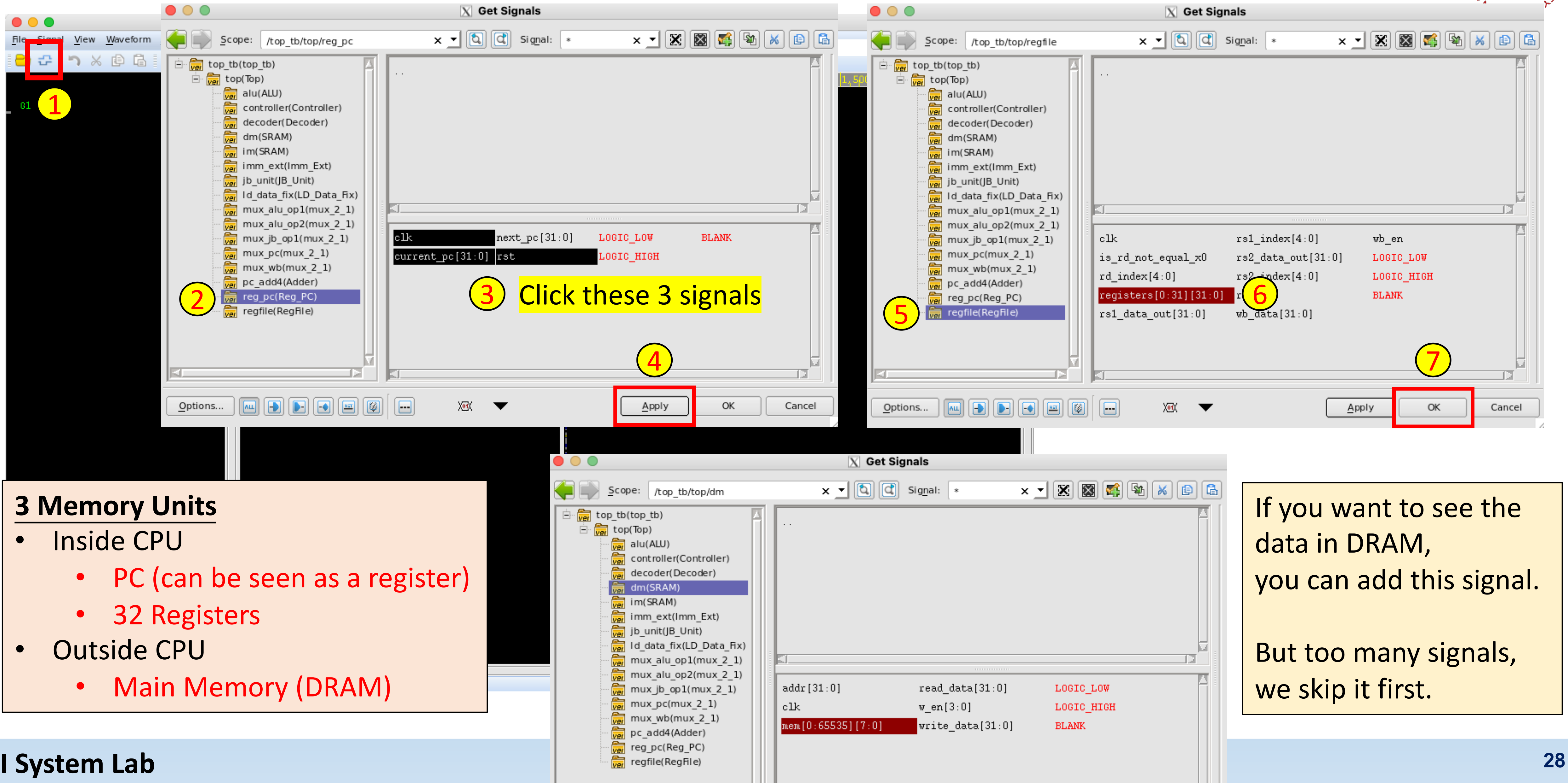


1. Double click

Or

2. Single click and click

Step 3 - Use nWave to Check waveform : Add Signal (2/7)



Scope: /top_tb/top/reg_pc Signal: *

Scope: /top_tb/top/regfile Signal: *

Scope: /top_tb/top/dm Signal: *

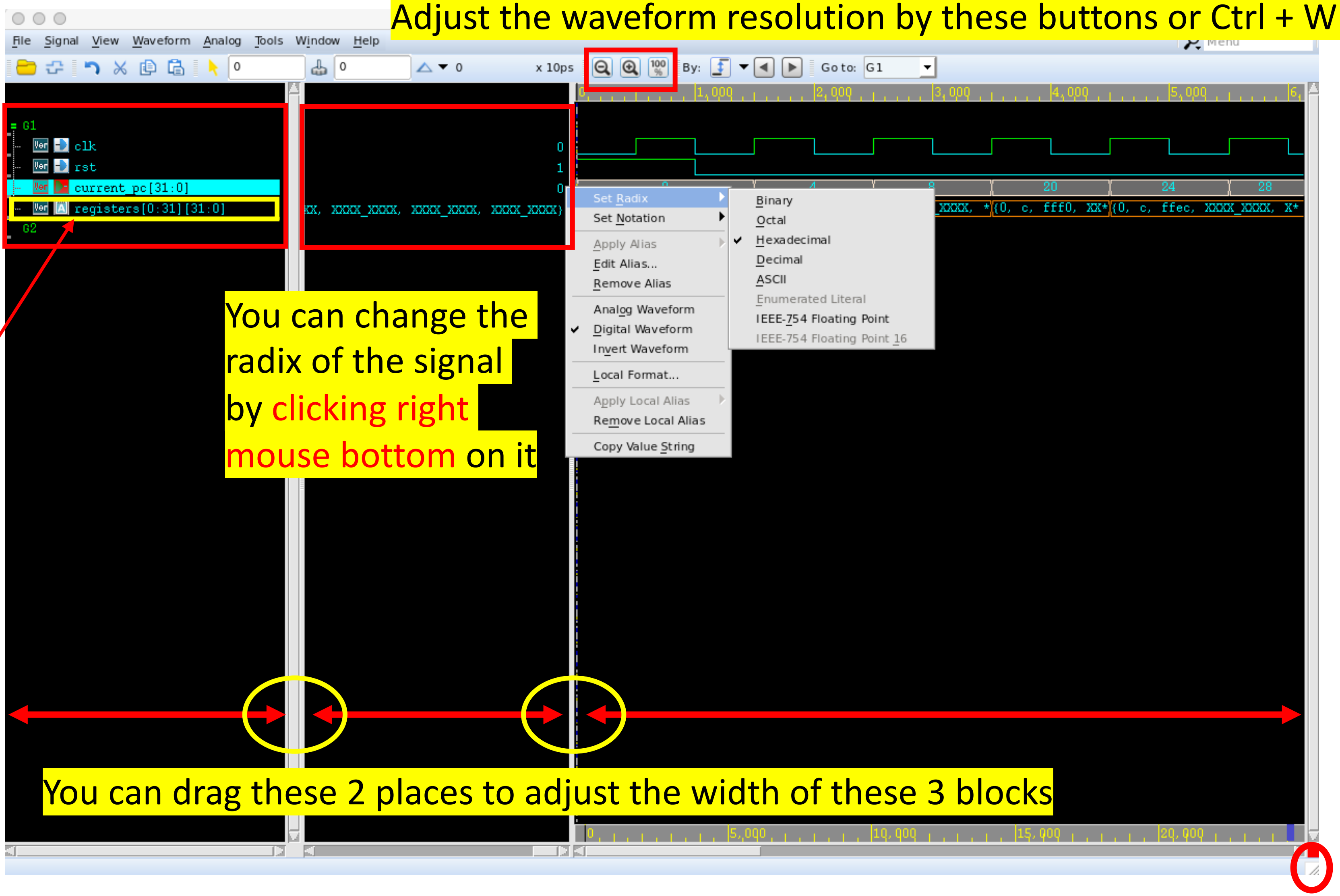
3 Memory Units

- Inside CPU
  - PC (can be seen as a register)
  - 32 Registers
- Outside CPU
  - Main Memory (DRAM)

If you want to see the data in DRAM,
you can add this signal.

But too many signals,
we skip it first.

Step 3 - Use nWave to Check waveform : Tips (3/7)



Adjust the waveform resolution by these buttons or Ctrl + Wheel

You can drag the signals by pressing and holding the left mouse button to move their order

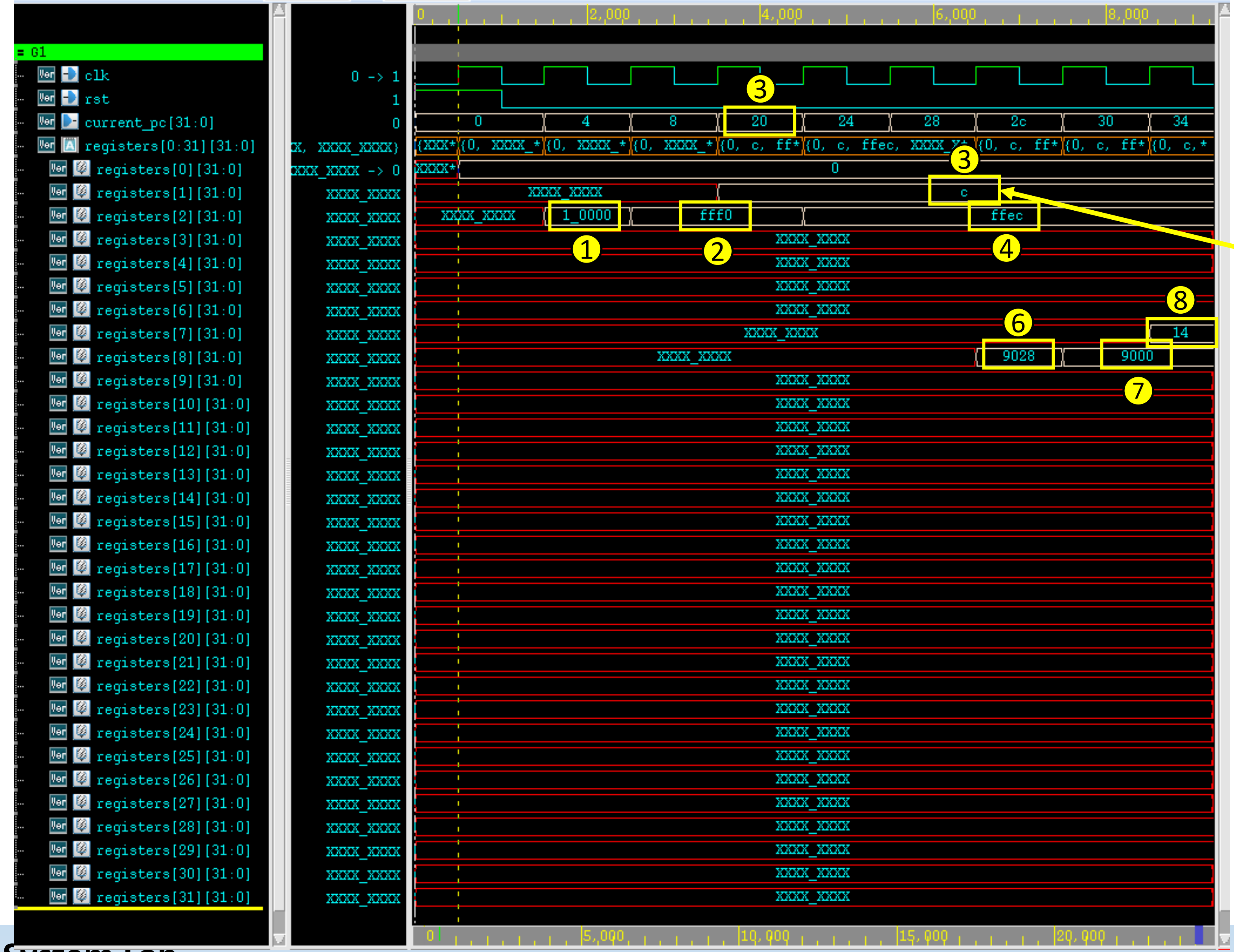
You can change the radix of the signal by clicking right mouse bottom on it

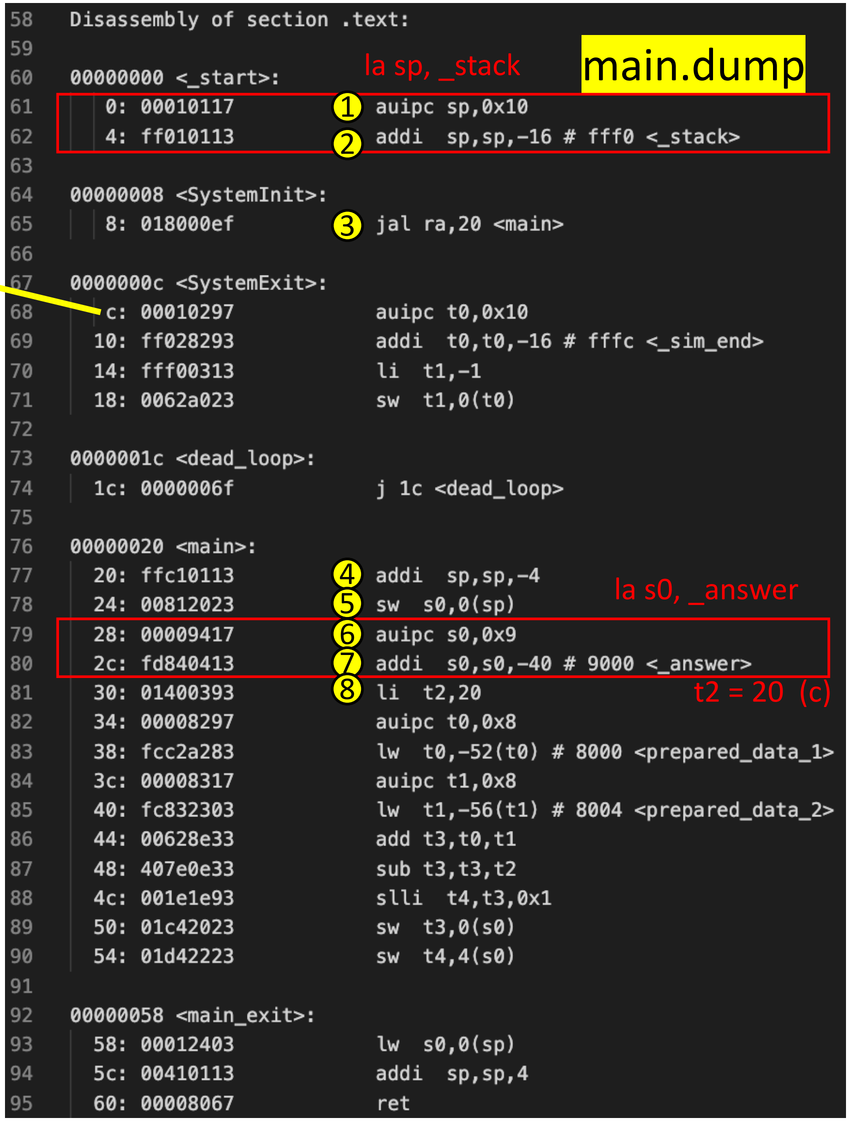
Double click left mouse button can expand 32 registers

You can drag these 2 places to adjust the width of these 3 blocks

You can drag here to resize the entire window

Step 3 - Use nWave to Check waveform : prog0 (4/7)





main.dump

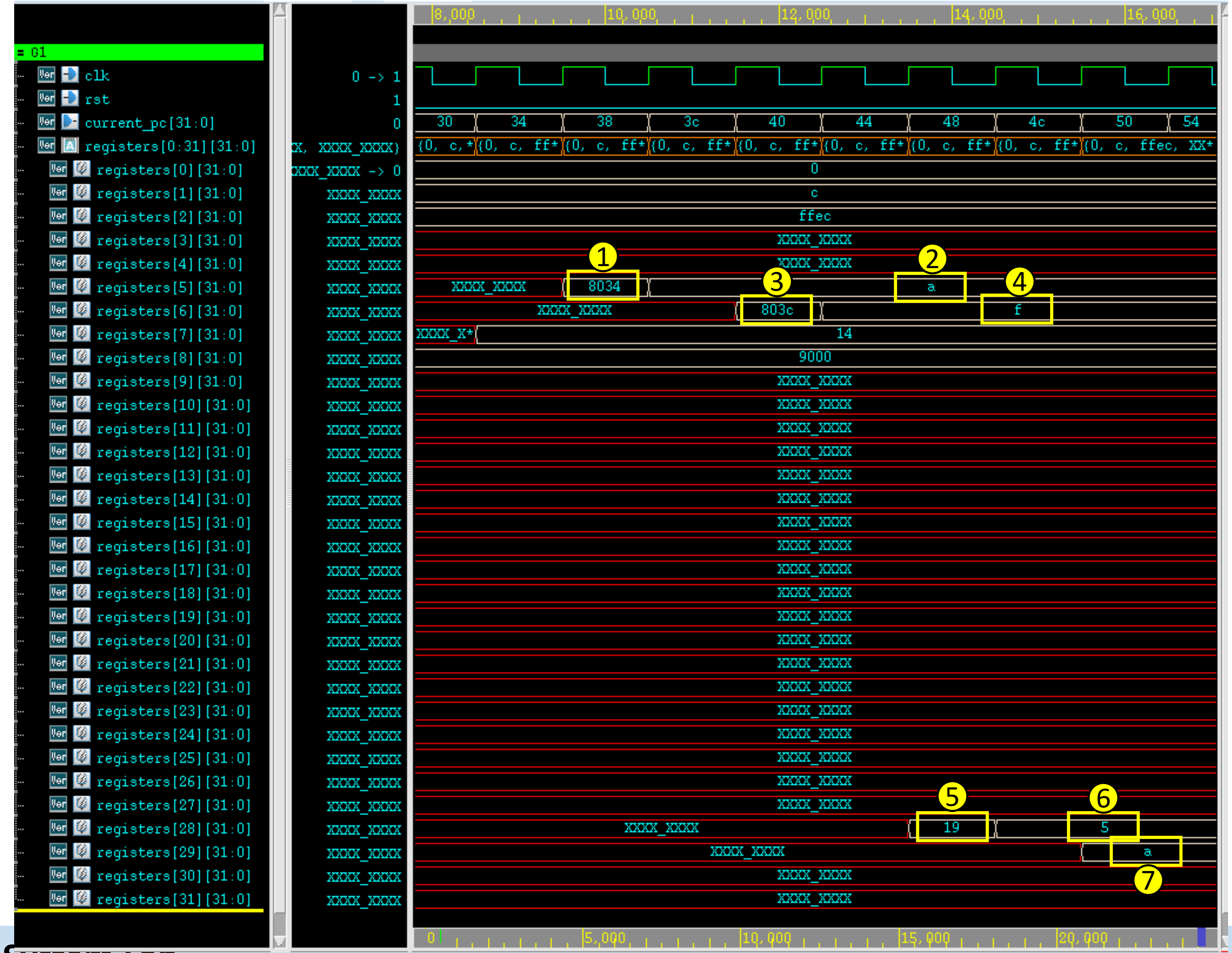
la sp, _stack

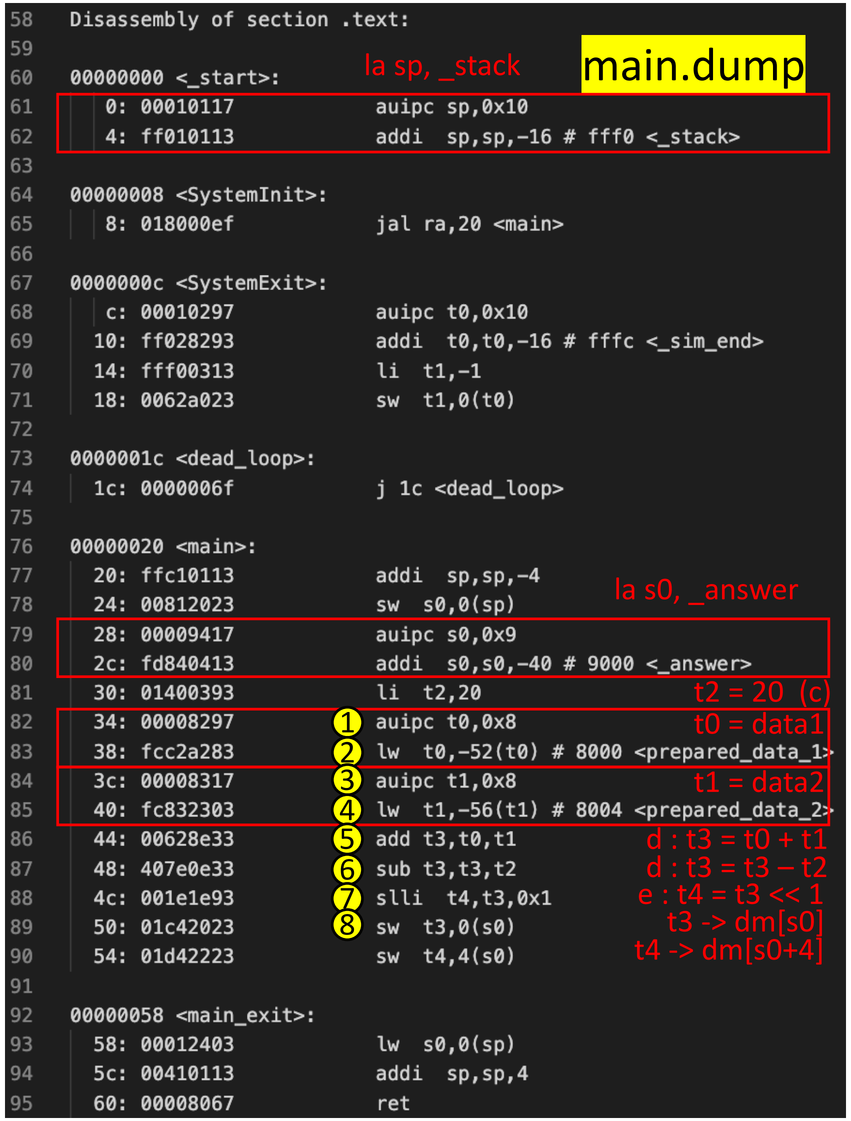
la s0, _answer

t2 = 20 (c)

```
Disassembly of section .text:

00000000 <_start>:
    0: 00010117         ① auipc sp,0x10
    4: ff010113         ② addi  sp,sp,-16 # fff0 <_stack>

00000008 <SystemInit>:
    8: 018000ef         ③ 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         ④ addi  sp,sp,-4
   24: 00812023         ⑤ sw  s0,0(sp)
   28: 00009417         ⑥ auipc s0,0x9
   2c: fd840413         ⑦ addi  s0,s0,-40 # 9000 <_answer>
   30: 01400393         ⑧ li  t2,20
   34: 00008297           auipc t0,0x8
   38: fcc2a283           lw  t0,-52(t0) # 8000 <prepared_data_1>
   3c: 00008317           auipc t1,0x8
   40: fc832303           lw  t1,-56(t1) # 8004 <prepared_data_2>
   44: 00628e33           add t3,t0,t1
   48: 407e0e33           sub t3,t3,t2
   4c: 001e1e93           slli  t4,t3,0x1
   50: 01c42023           sw  t3,0(s0)
   54: 01d42223           sw  t4,4(s0)

00000058 <main_exit>:
   58: 00012403           lw  s0,0(sp)
   5c: 00410113           addi  sp,sp,4
   60: 00008067           ret
```

Step 3 - Use nWave to Check waveform : prog0 (5/7)





main.dump

```
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>
  30: 01400393          li  t2,20
  34: 00008297        ① auipc t0,0x8
  38: fcc2a283        ② lw  t0,-52(t0) # 8000 <prepared_data_1>
  3c: 00008317        ③ auipc t1,0x8
  40: fc832303        ④ lw  t1,-56(t1) # 8004 <prepared_data_2>
  44: 00628e33        ⑤ add t3,t0,t1
  48: 407e0e33        ⑥ sub t3,t3,t2
  4c: 001e1e93        ⑦ slli  t4,t3,0x1
  50: 01c42023        ⑧ sw  t3,0(s0)
  54: 01d42223          sw  t4,4(s0)

00000058 <main_exit>:
  58: 00012403          lw  s0,0(sp)
  5c: 00410113          addi  sp,sp,4
  60: 00008067          ret
```

la sp, _stack

la s0, _answer

t2 = 20 (c)

t0 = data1

t1 = data2

d : t3 = t0 + t1

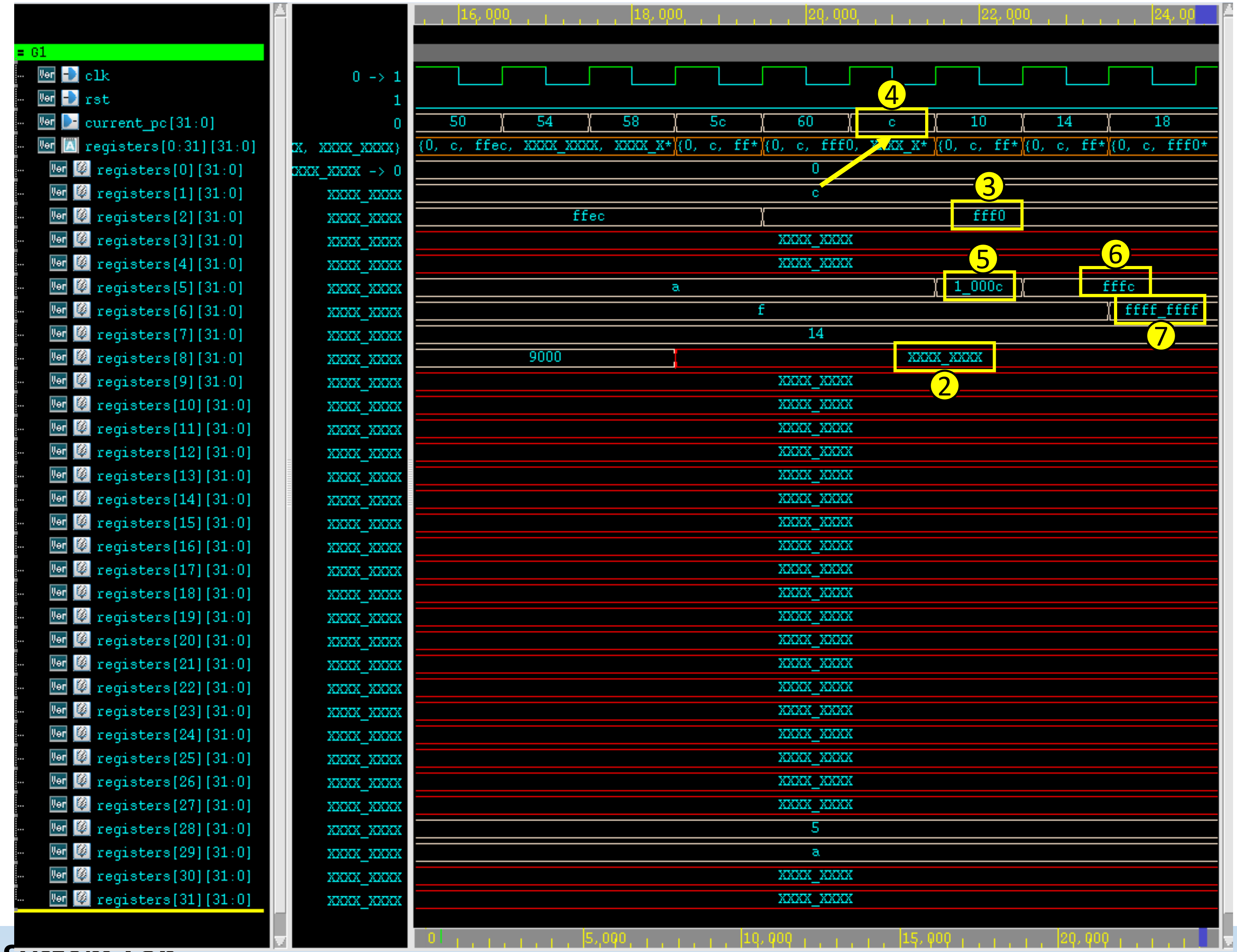
d : t3 = t3 – t2

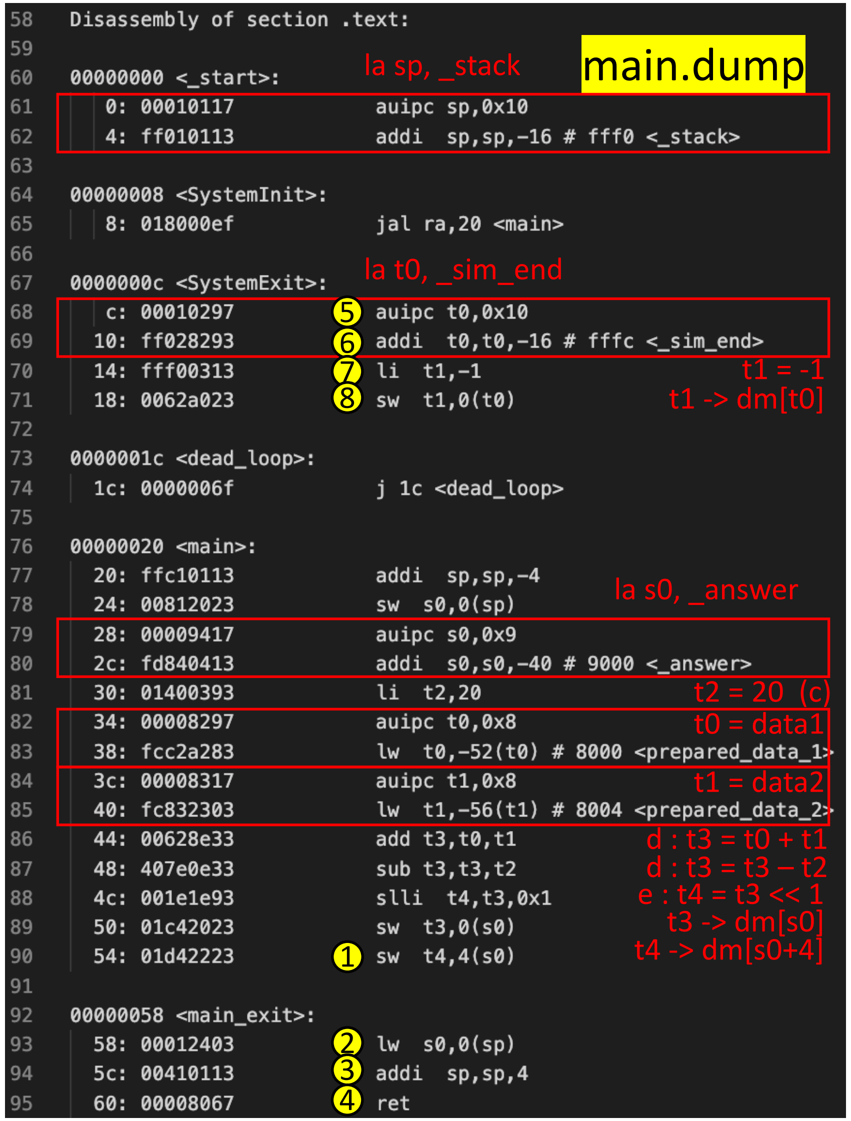
e : t4 = t3 << 1

t3 -> dm[s0]

t4 -> dm[s0+4]

Step 3 - Use nWave to Check waveform : prog0 (6/7)





main.dump

```
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        ⑤ auipc t0,0x10
   10: ff028293        ⑥ addi  t0,t0,-16 # fffc <_sim_end>
   14: fff00313        ⑦ li  t1,-1
   18: 0062a023        ⑧ 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>
   30: 01400393          li  t2,20
   34: 00008297          auipc t0,0x8
   38: fcc2a283          lw  t0,-52(t0) # 8000 <prepared_data_1>
   3c: 00008317          auipc t1,0x8
   40: fc832303          lw  t1,-56(t1) # 8004 <prepared_data_2>
   44: 00628e33          add t3,t0,t1
   48: 407e0e33          sub t3,t3,t2
   4c: 001e1e93          slli  t4,t3,0x1
   50: 01c42023          sw  t3,0(s0)
   54: 01d42223        ① sw  t4,4(s0)

00000058 <main_exit>:
   58: 00012403        ② lw  s0,0(sp)
   5c: 00410113        ③ addi  sp,sp,4
   60: 00008067        ④ ret
```

la sp, _stack

la t0, _sim_end

t1 = -1

t1 -> dm[t0]

la s0, _answer

t2 = 20 (c)

t0 = data1

t1 = data2

d : t3 = t0 + t1

d : t3 = t3 – t2

e : t4 = t3 << 1

t3 -> dm[s0]

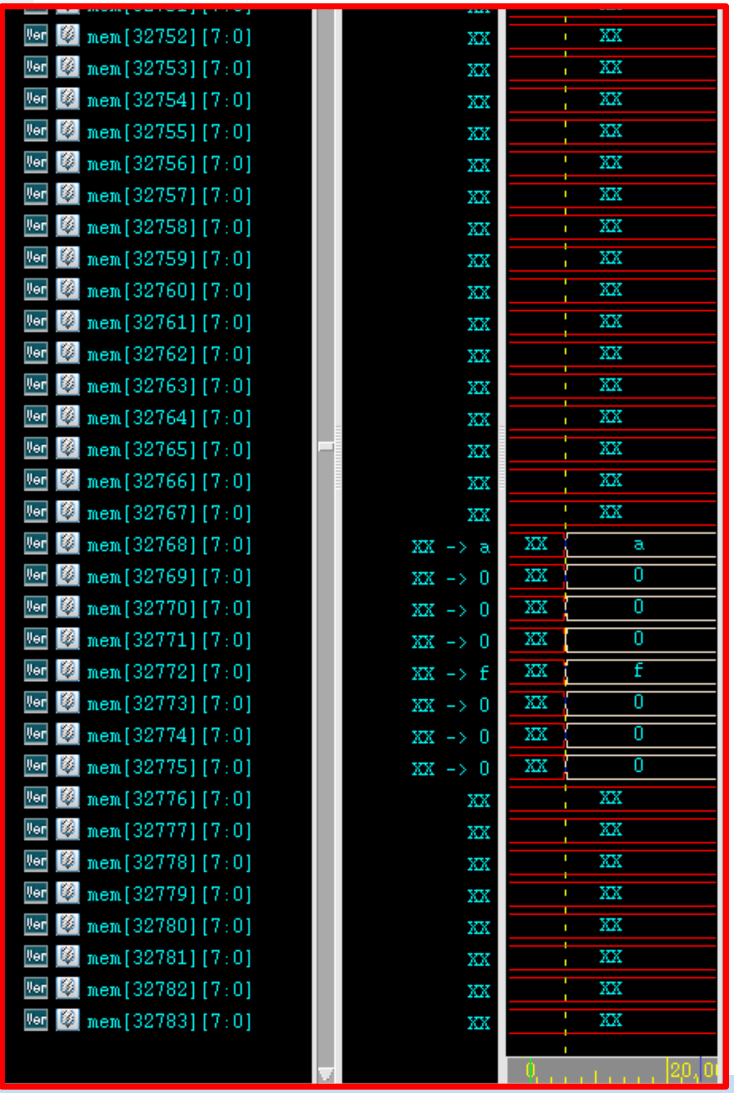
t4 -> dm[s0+4]

Step 3 - Use nWave to Check waveform : DRAM (7/7)

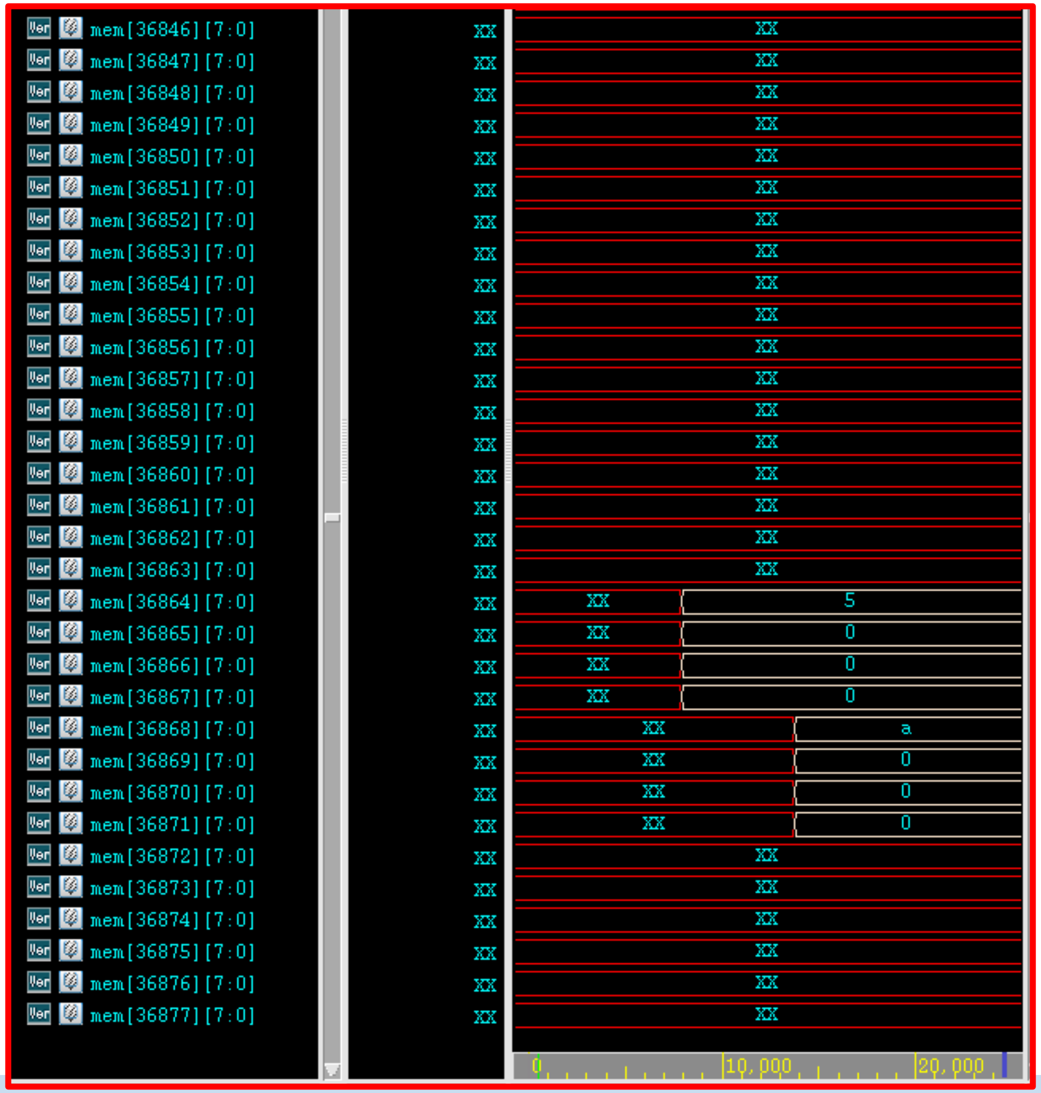
$0x8000_{(hex)} = 8 * 16^3 = 32768_{decimal}$

$0x9000_{(hex)} = 9 * 16^3 = 36864_{decimal}$

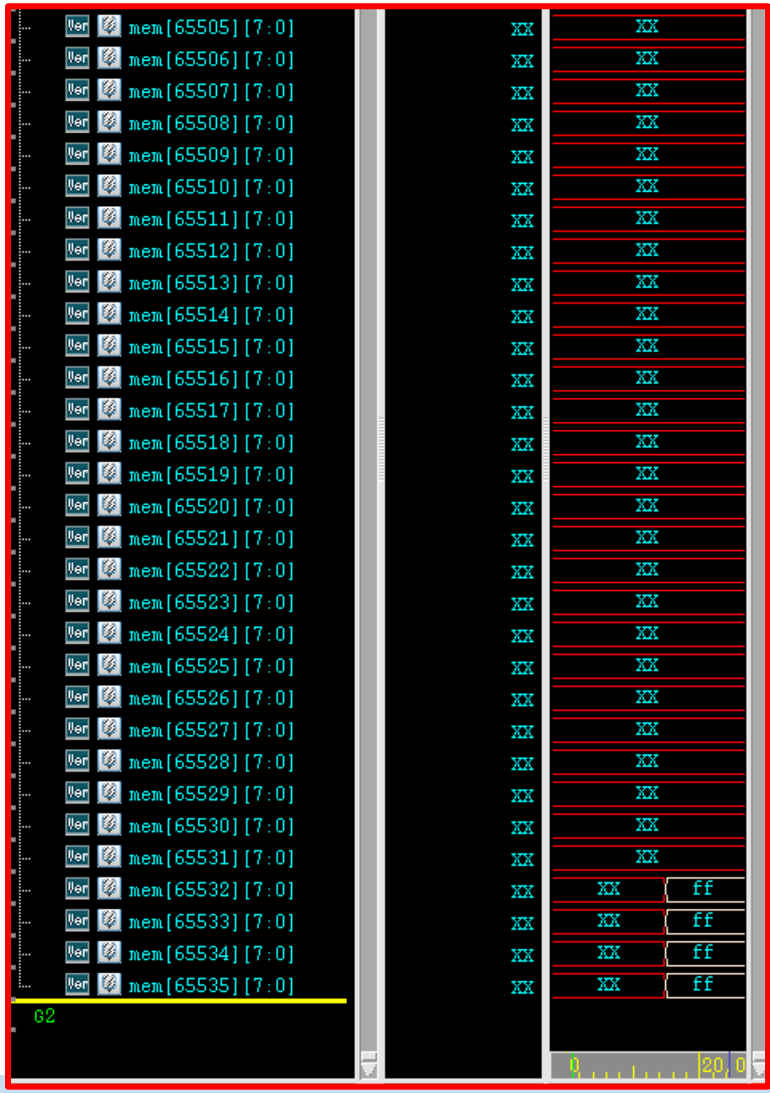
$0xfffc_{(hex)} = 65532_{decimal}$



```
mem[32752][7:0]   XX        XX
mem[32753][7:0]   XX        XX
mem[32754][7:0]   XX        XX
mem[32755][7:0]   XX        XX
mem[32756][7:0]   XX        XX
mem[32757][7:0]   XX        XX
mem[32758][7:0]   XX        XX
mem[32759][7:0]   XX        XX
mem[32760][7:0]   XX        XX
mem[32761][7:0]   XX        XX
mem[32762][7:0]   XX        XX
mem[32763][7:0]   XX        XX
mem[32764][7:0]   XX        XX
mem[32765][7:0]   XX        XX
mem[32766][7:0]   XX        XX
mem[32767][7:0]   XX        XX
mem[32768][7:0]   XX -> a   XX   a
mem[32769][7:0]   XX -> 0   XX   0
mem[32770][7:0]   XX -> 0   XX   0
mem[32771][7:0]   XX -> 0   XX   0
mem[32772][7:0]   XX -> f   XX   f
mem[32773][7:0]   XX -> 0   XX   0
mem[32774][7:0]   XX -> 0   XX   0
mem[32775][7:0]   XX -> 0   XX   0
mem[32776][7:0]   XX        XX
mem[32777][7:0]   XX        XX
mem[32778][7:0]   XX        XX
mem[32779][7:0]   XX        XX
mem[32780][7:0]   XX        XX
mem[32781][7:0]   XX        XX
mem[32782][7:0]   XX        XX
mem[32783][7:0]   XX        XX
                            0         20,0
```

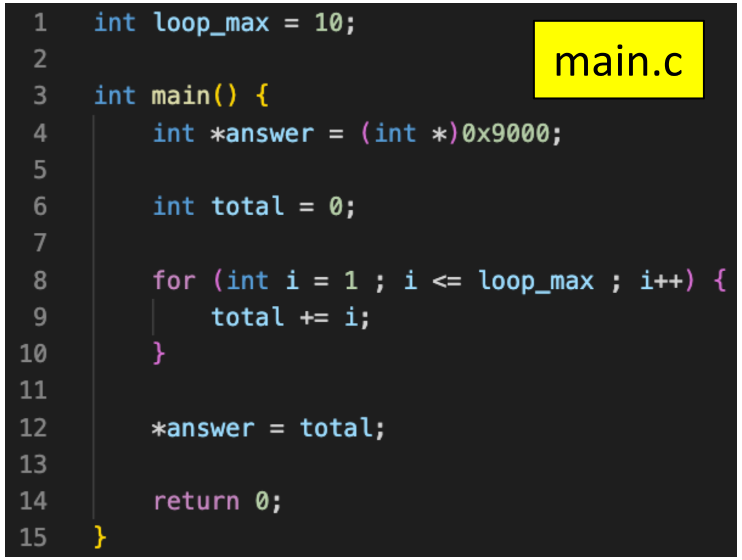


```
mem[36846][7:0]   XX        XX
mem[36847][7:0]   XX        XX
mem[36848][7:0]   XX        XX
mem[36849][7:0]   XX        XX
mem[36850][7:0]   XX        XX
mem[36851][7:0]   XX        XX
mem[36852][7:0]   XX        XX
mem[36853][7:0]   XX        XX
mem[36854][7:0]   XX        XX
mem[36855][7:0]   XX        XX
mem[36856][7:0]   XX        XX
mem[36857][7:0]   XX        XX
mem[36858][7:0]   XX        XX
mem[36859][7:0]   XX        XX
mem[36860][7:0]   XX        XX
mem[36861][7:0]   XX        XX
mem[36862][7:0]   XX        XX
mem[36863][7:0]   XX        XX
mem[36864][7:0]   XX        XX        5
mem[36865][7:0]   XX        XX        0
mem[36866][7:0]   XX        XX        0
mem[36867][7:0]   XX        XX        0
mem[36868][7:0]   XX           XX        a
mem[36869][7:0]   XX           XX        0
mem[36870][7:0]   XX           XX        0
mem[36871][7:0]   XX           XX        0
mem[36872][7:0]   XX        XX
mem[36873][7:0]   XX        XX
mem[36874][7:0]   XX        XX
mem[36875][7:0]   XX        XX
mem[36876][7:0]   XX        XX
mem[36877][7:0]   XX        XX
                            0         10,000         20,000
```



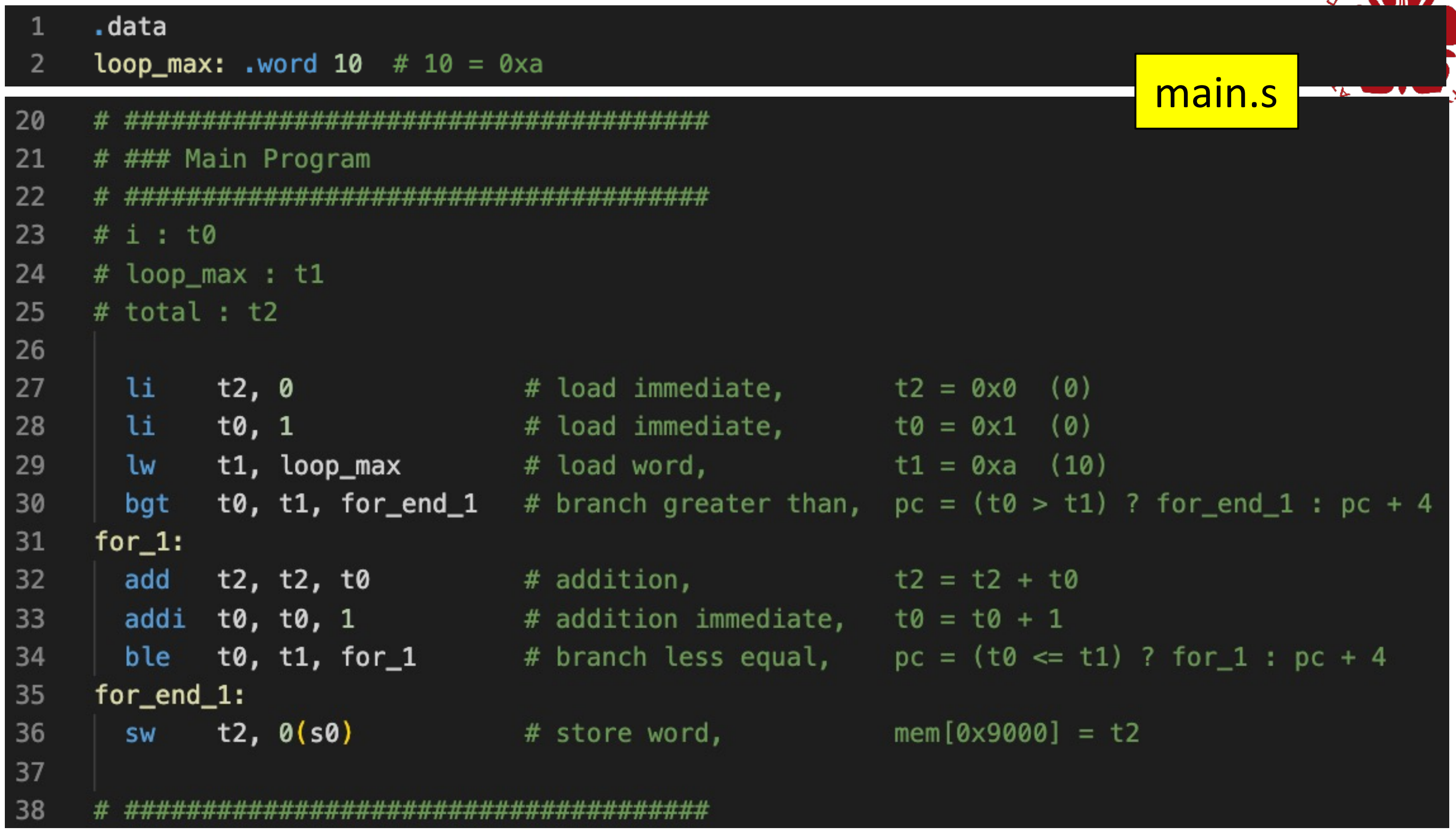
```
mem[65505][7:0]   XX        XX
mem[65506][7:0]   XX        XX
mem[65507][7:0]   XX        XX
mem[65508][7:0]   XX        XX
mem[65509][7:0]   XX        XX
mem[65510][7:0]   XX        XX
mem[65511][7:0]   XX        XX
mem[65512][7:0]   XX        XX
mem[65513][7:0]   XX        XX
mem[65514][7:0]   XX        XX
mem[65515][7:0]   XX        XX
mem[65516][7:0]   XX        XX
mem[65517][7:0]   XX        XX
mem[65518][7:0]   XX        XX
mem[65519][7:0]   XX        XX
mem[65520][7:0]   XX        XX
mem[65521][7:0]   XX        XX
mem[65522][7:0]   XX        XX
mem[65523][7:0]   XX        XX
mem[65524][7:0]   XX        XX
mem[65525][7:0]   XX        XX
mem[65526][7:0]   XX        XX
mem[65527][7:0]   XX        XX
mem[65528][7:0]   XX        XX
mem[65529][7:0]   XX        XX
mem[65530][7:0]   XX        XX
mem[65531][7:0]   XX        XX
mem[65532][7:0]   XX        XX   ff
mem[65533][7:0]   XX        XX   ff
mem[65534][7:0]   XX        XX   ff
mem[65535][7:0]   XX        XX   ff
G2
                            0        20,0
```

Test Program (prog1)



main.c

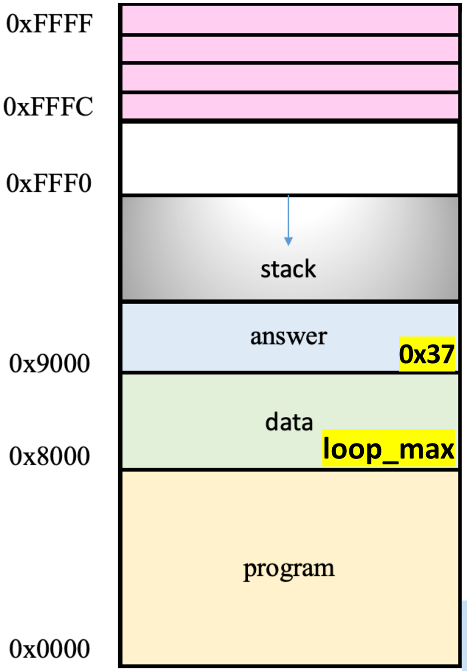
```c
int loop_max = 10;

int main() {
    int *answer = (int *)0x9000;

    int total = 0;

    for (int i = 1 ; i <= loop_max ; i++) {
        total += i;
    }

    *answer = total;

    return 0;
}
```



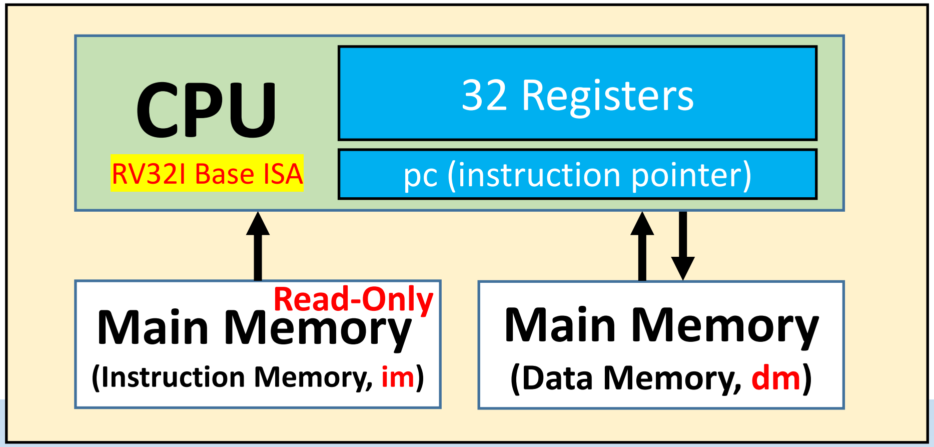
main.s

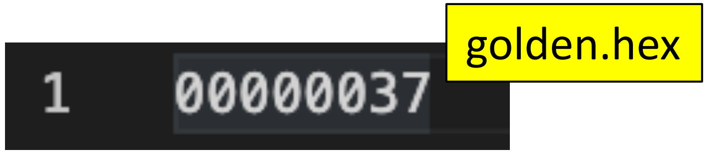
```
.data
loop_max: .word 10  # 10 = 0xa
```

```
# ######################################
# ### Main Program
# ######################################
# i : t0
# loop_max : t1
# total : t2

    li     t2, 0            # load immediate,       t2 = 0x0  (0)
    li     t0, 1            # load immediate,       t0 = 0x1  (0)
    lw     t1, loop_max     # load word,            t1 = 0xa  (10)
    bgt    t0, t1, for_end_1  # branch greater than,  pc = (t0 > t1) ? for_end_1 : pc + 4
for_1:
    add    t2, t2, t0       # addition,             t2 = t2 + t0
    addi   t0, t0, 1        # addition immediate,   t0 = t0 + 1
    ble    t0, t1, for_1    # branch less equal,    pc = (t0 <= t1) ? for_1 : pc + 4
for_end_1:
    sw     t2, 0(s0)        # store word,           mem[0x9000] = t2

# ######################################
```



$$55_{decimal} = 3 * 16 + 7 = 37_{hex}$$



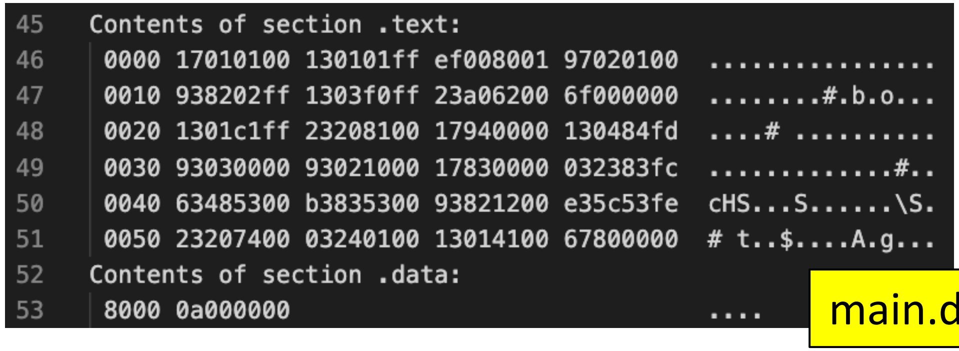


golden.hex

```
00000037
```

Test Program (prog1) - Do yourself !

1. Use RISC-V Toolchain to generate “main.hex” & “main.dump”

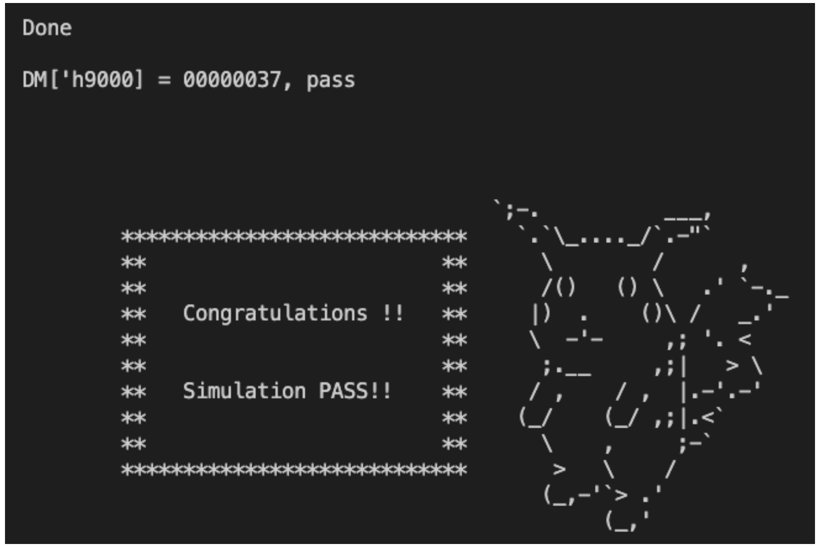


```
Contents of section .text:
 0000 17010100 130101ff ef008001 97020100  ................
 0010 938202ff 1303f0ff 23a06200 6f000000  ........#.b.o...
 0020 1301c1ff 23208100 17940000 130484fd  ....# ..........
 0030 93030000 93021000 17830000 032383fc  .............#..
 0040 63485300 b3835300 93821200 e35c53fe  cHS...S......\S.
 0050 23207400 03240100 13014100 67800000  # t..$....A.g...
Contents of section .data:
 8000 0a000000                             ....
```

main.dump

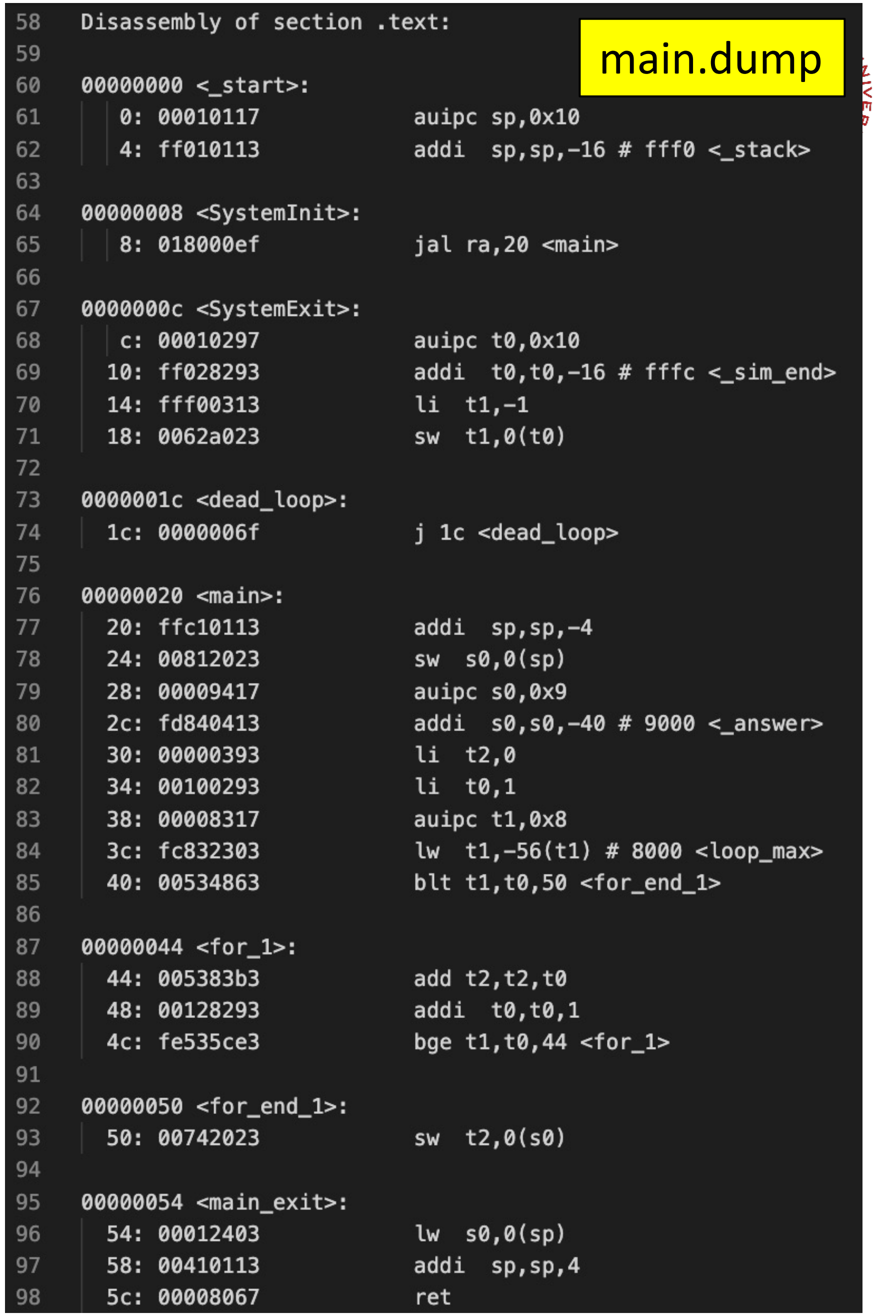
2. Use NC-Verilog to simulate a CPU running the program and producing waveform file

Check the print result



```
Done

DM['h9000'] = 00000037, pass



                                      `;-.          ___,
        ****************************    `.`\_...._/`.-"`
        **                        **      \        /      ,
        **                        **      /()   () \    .' `-._
        **   Congratulations !!   **     |)  .    ()\  /   _.'
        **                        **     \  -'-     ,; '. <
        **                        **      ;.__     ,;|   > \
        **   Simulation PASS!!    **     / ,    / ,  |.-'.-'
        **                        **    (_/    (_/ ,;|.<`
        **                        **      \    ,     ;-`
        ****************************       >   \    /
                                          (_,-'`> .'
                                               (_,'
```

3. Use nWave with dump file to check waveform



main.dump

```
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>
  30:  00000393          li  t2,0
  34:  00100293          li  t0,1
  38:  00008317          auipc t1,0x8
  3c:  fc832303          lw  t1,-56(t1) # 8000 <loop_max>
  40:  00534863          blt t1,t0,50 <for_end_1>

00000044 <for_1>:
  44:  005383b3          add t2,t2,t0
  48:  00128293          addi  t0,t0,1
  4c:  fe535ce3          bge t1,t0,44 <for_1>

00000050 <for_end_1>:
  50:  00742023          sw  t2,0(s0)

00000054 <main_exit>:
  54:  00012403          lw  s0,0(sp)
  58:  00410113          addi  sp,sp,4
  5c:  00008067          ret
```