Computer Architecture HW1

TA: Claire Peng(彭亞綺)

Email: yachi@access.ee.ntu.edu.tw

Table of Contents

GEM5 Introduction

- What is gem5?
  - A modular, discrete-event driven computer system simulator.
  - Used for computer architecture research and education.
- Key feature
  - Simulation Modes: Full system (FS) and syscall emulation (SE).
  - Supported Architectures: Alpha, ARM, MIPS, Power, SPARC, **RISC-V**, x86-64.
- Resources
  - gem5 Documentation
  - Source code

GEM5 Installation - Prepare Image

- **Docker** is a platform used to develop, package, and run applications in isolated environments called containers
- **Containers** are consistent across different systems

Follow these Steps to install:

- Install Docker Desktop
- Download the image file (.tar.gz) and extract it to .tar (Link)
  - Size: ~8GB download, ~26GB after extraction.
- Load Image:

```
> docker load -input <path to extracted .tar file>
```

```
PS C:\Users\user> docker load --input "C:\Users\user\Downloads\ca_env_riscv64gc-multilib_path\ca_env_riscv64gc
-multilib_path.tar"
Loaded image: ca_env:riscv64gc-multilib_path
```

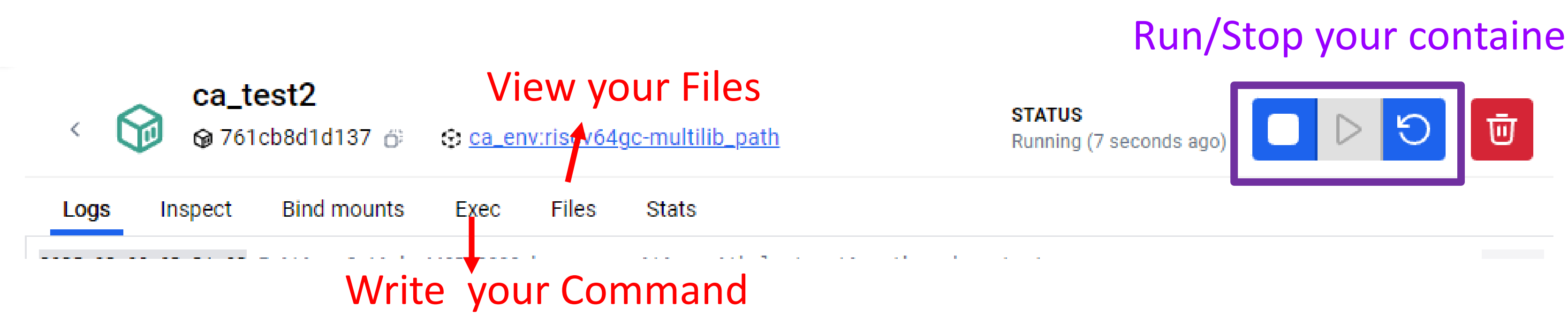
GEM5 Installation - Prepare Image

- **Run Container**
  - Create a folder name *workspace* on your local and copy the absolute path of this folder
- > docker run -it --name <container_name> -v <local_absolute_path>:/workspace ca_env:riscv64gc-multilib_path bash
  - Container name: Self-defined name (e.g., 1132_CA)
- This will map the local folder to the *workspace* folder in the Docker container, so they will update synchronously

GEM5 Installation - Docker Desktop Utility

- Use terminal (recommend)
  - 查詢containers資訊 (id/name) > docker ps -a
  - 重新連接container > docker start -i <id/name>
  - 斷開連接 > exit

- Through GUI
  - Containers > (select your container)



GEM5 Installation - Run Simulation

- Download files from NTUCOOL
- Put **simple-riscv-mod.py, *Makefile, example.s*** into *workspace*
- Run your assembly code: (Under *workspace* folder)
  - Change program name in makefile

```
15    # Program name
16    PROG := example
```

  - Compile the program into executable file `> make asm`
    - Equal to `> /opt/riscv/bin/riscv64-unknown-linux-gnu-g++ -march=rv32gc -mabi=ilp32 example.s –o example -static`
    - The new executable file with same name will generate in workspace
  - Run Simulation on Gem5 `> make gem5`
    - Equal to `>../gem5/build/RISCV/gem5.opt --debug-flags=Exec --debug-file=out_exec.txt simple-riscv-mod.py example $(GEM5_ARGS)`

GEM5 Installation - Run Simulation(cont)

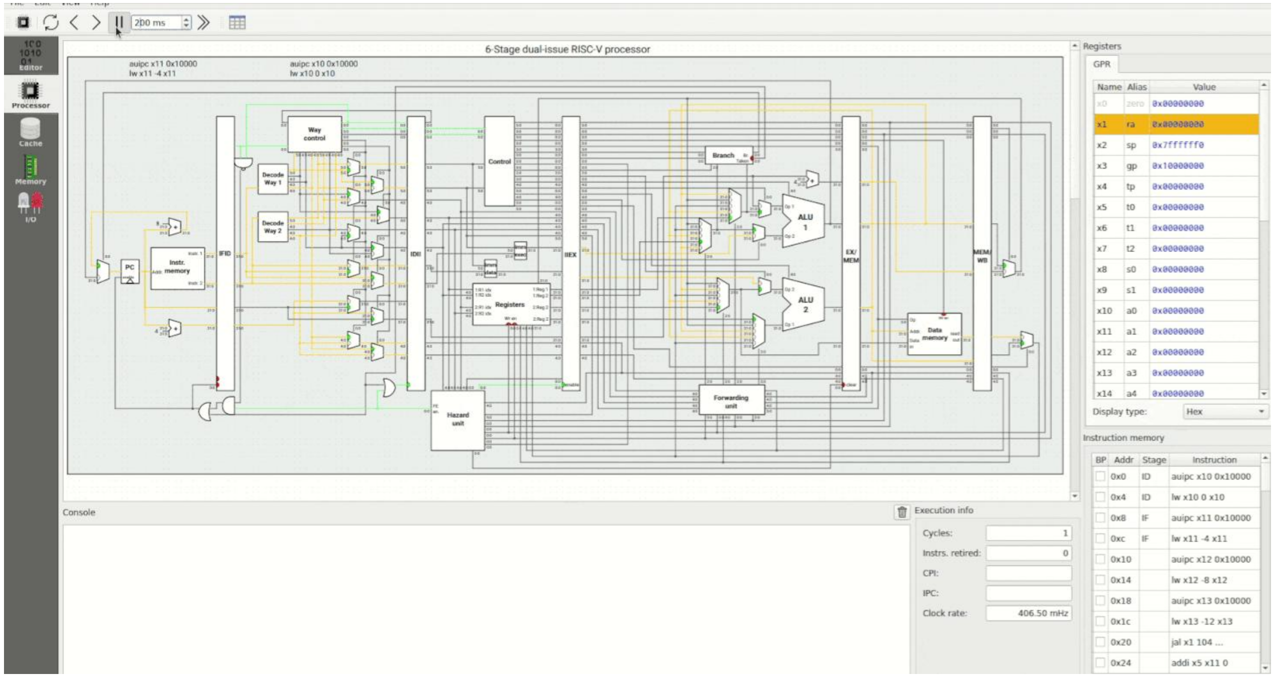
- If your gem5 installation is successful, you will see the following simulation messages in the terminal.
- You can find detailed information(simulation time/hit rate…) in workspace/m5_out/stats.txt

```
Beginning simulation!
src/sim/simulate.cc:199: info: Entering event queue @ 0.  Starting simulation...
src/sim/syscall_emul.cc:97: warn: ignoring syscall set_robust_list(...)
      (further warnings will be suppressed)
src/sim/syscall_emul.hh:1117: warn: readlink() called on '/proc/self/exe' may yield unexpected results in vari
ous settings.
      Returning '/workspace/example'
src/sim/mem_state.cc:448: info: Increasing stack size by one page.
src/sim/syscall_emul.cc:86: warn: ignoring syscall mprotect(...)
Exiting @ tick 8205739000 because exiting with last active thread context
Emulated example on gem5 with arguments:
```

We will use gem5 for evaluating performance in future assignments !

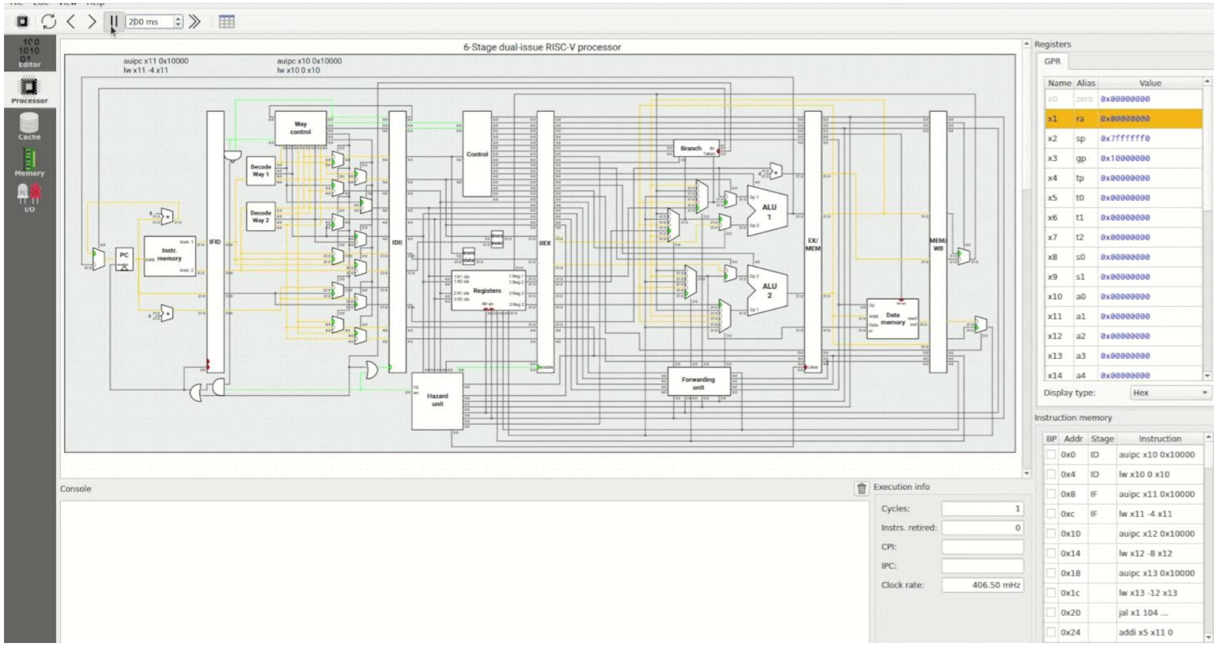
RIPES Introduction

- Ripes is a visual computer architecture simulator and assembly code editor built for the RISC-V instruction set architecture.
- We use this tool for debugging and testing functionality.



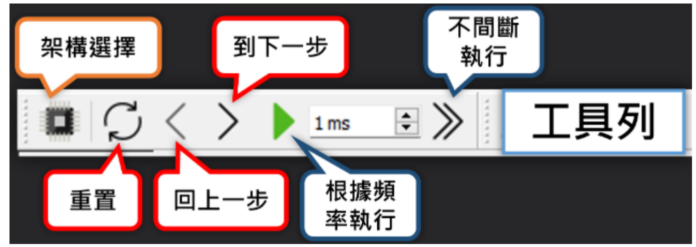
RIPES Installation

- Download here (v.2.2.6) according to your PC environment
- Unzip the content and execute Ripes.exe
- There are also browser version: https://ripes.me/
- Read more: Documentation

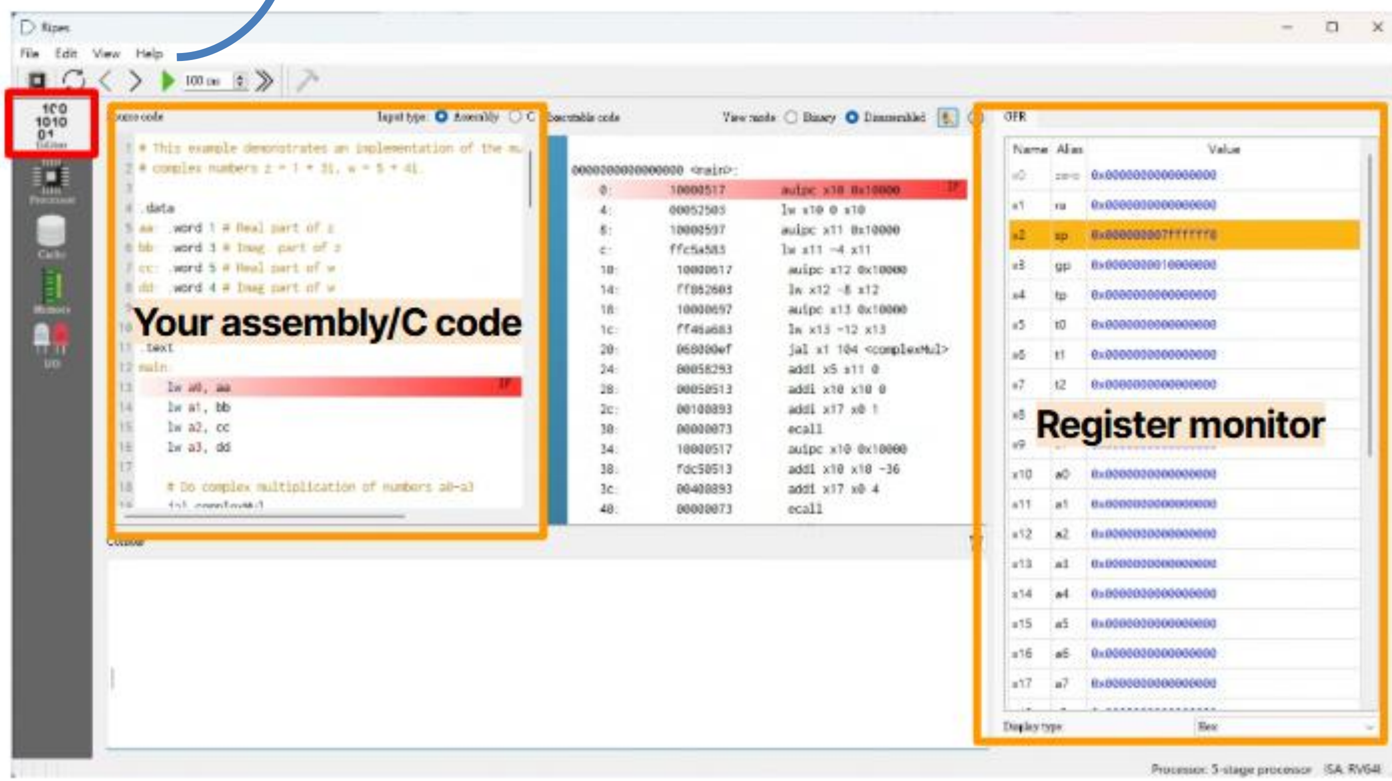


GUI of RIPES

- Editor tab

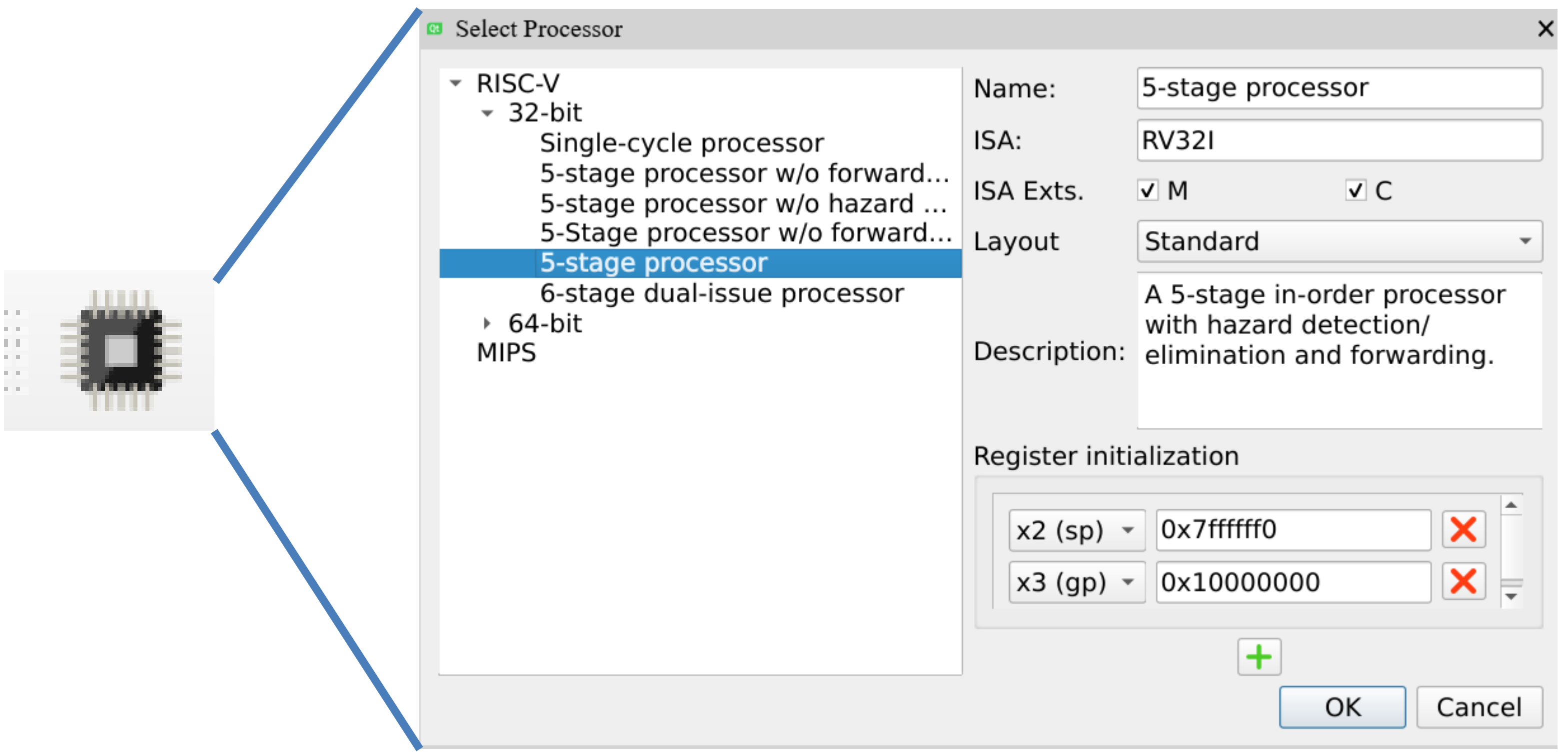


Editor tab



GUI of RIPES

- Select Processor
  - We use **32-bit** ISA through the whole semester
  - Choose 5-stage processor for default

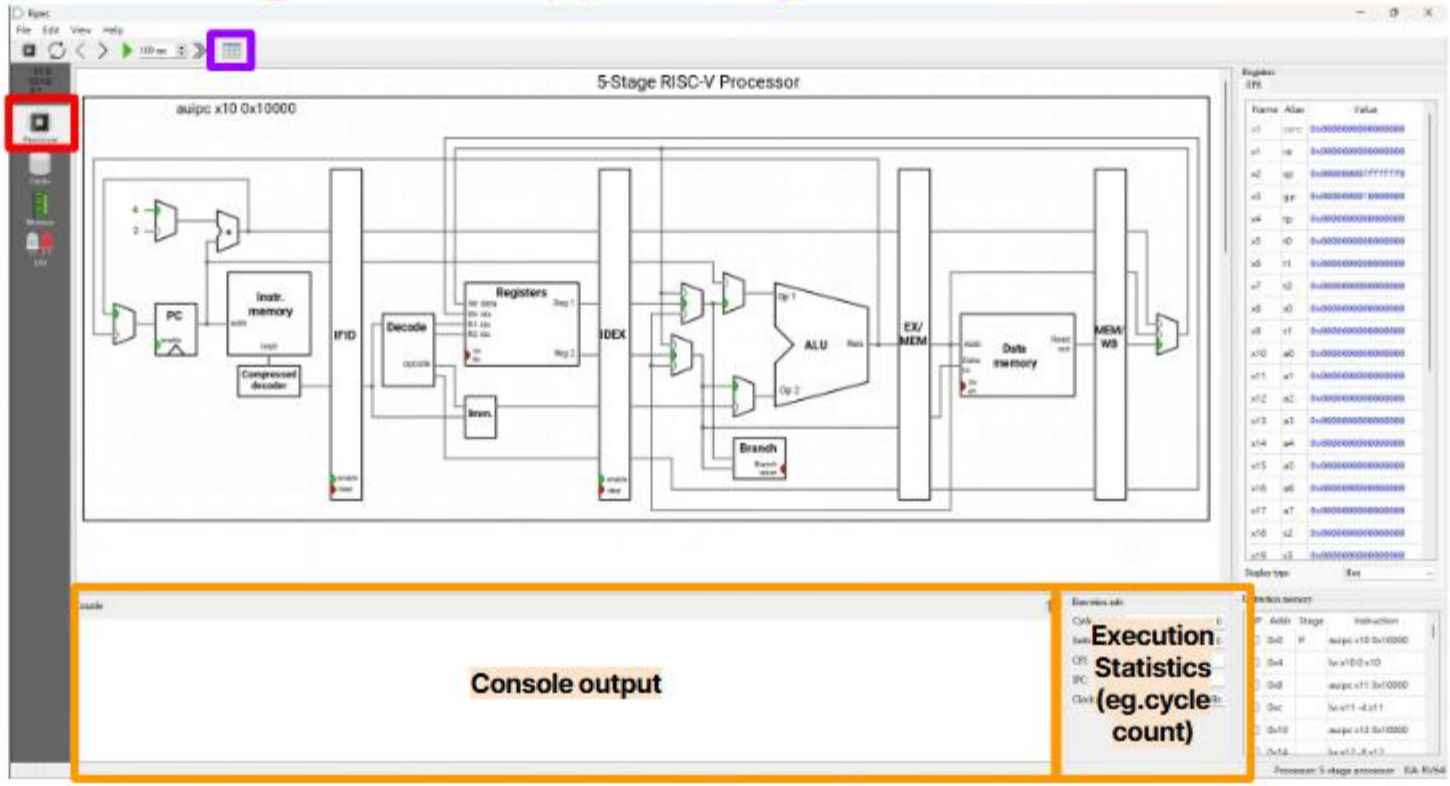


GUI of RIPES

- Processor tab

Stage table: view pipeline stage

Processor tab



Problem 1 - Recursive Function

- Implement recursive function fact, for any integer n>=0

$$fact(n) = \begin{cases} 4, & \text{if } n = 0, \\ 4fact(\frac{n-1}{2}) + 8n + 3, & \text{if } n = 1, 3, 5\ldots, \\ 4fact(\frac{n}{2}) + 8n + 3, & \text{if } n = 2, 4, 6\ldots, \end{cases}$$

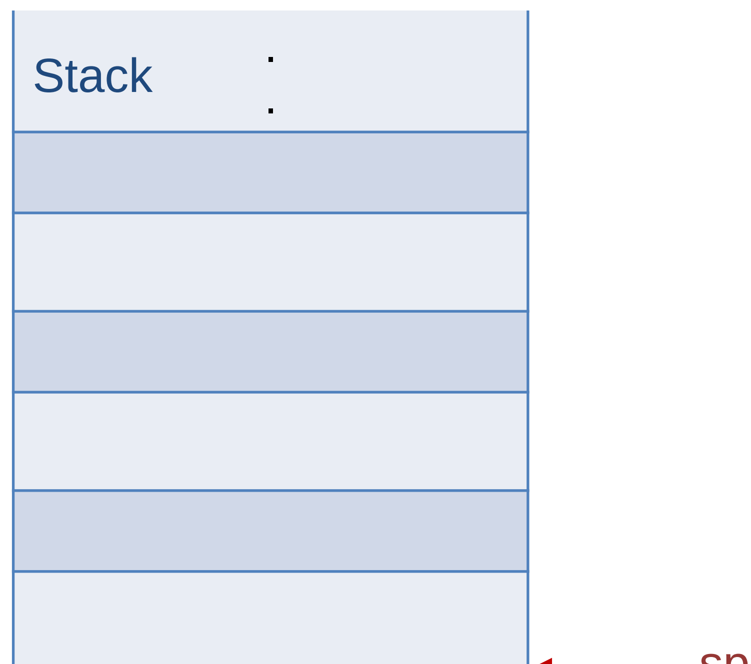
- Example
  - fact(7)=599
  - fact(10)=655

Recursive Example

- In factorial.s   Fact(n) = n* Fact(n-1) , Fact(0)=1 , n=2
- Use stack to store current n and return address

```
fact:
    addi sp, sp, -16     # Allocate stack frame
    sw   ra, 8(sp)       # Save return address
    sw   a0, 0(sp)       # Save argument (n)

    addi t0, a0, -1      # t0 = n - 1
    bge  t0, zero, nfact # if n - 1 >= 0 → recursive case

    # Base case: n <= 0 → return 1
    addi a0, zero, 1
    addi sp, sp, 16      # Free stack frame
    jr x1                # Return

nfact:
    addi a0, a0, -1
    jal  ra, fact        # Recursive call: fact(n-1)

    addi t1, a0, 0       # t1 = fact(n - 1)
    lw   a0, 0(sp)       # Restore original argument (n)
    lw   ra, 8(sp)       # Restore return address
    addi sp, sp, 16      # Free stack frame

    mul a0, a0, t1       # n * fact(n-1)
    ret                  # Return result
```

ra	Address return to main
a0(n)	2

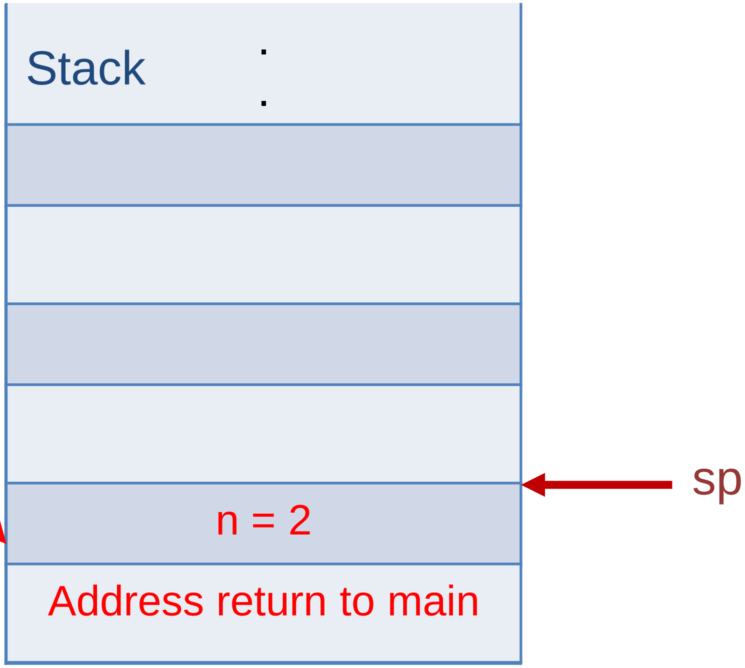


Recursive Example

- Store current n and return address
- Go to n-1

```
fact:
    addi sp, sp, -16     # Allocate stack frame
    sw   ra, 8(sp)       # Save return address
    sw   a0, 0(sp)       # Save argument (n)

    addi t0, a0, -1      # t0 = n - 1
    bge  t0, zero, nfact # if n - 1 >= 0 → recursive case

    # Base case: n <= 0 → return 1
    addi a0, zero, 1
    addi sp, sp, 16      # Free stack frame
    jr x1                # Return

nfact:
    addi a0, a0, -1
    jal  ra, fact        # Recursive call: fact(n-1)

    addi t1, a0, 0       # t1 = fact(n - 1)
    lw   a0, 0(sp)       # Restore original argument (n)
    lw   ra, 8(sp)       # Restore return address
    addi sp, sp, 16      # Free stack frame

    mul a0, a0, t1       # n * fact(n-1)
    ret                  # Return result
```

ra	Address return to Fact(2)
a0(n)	1

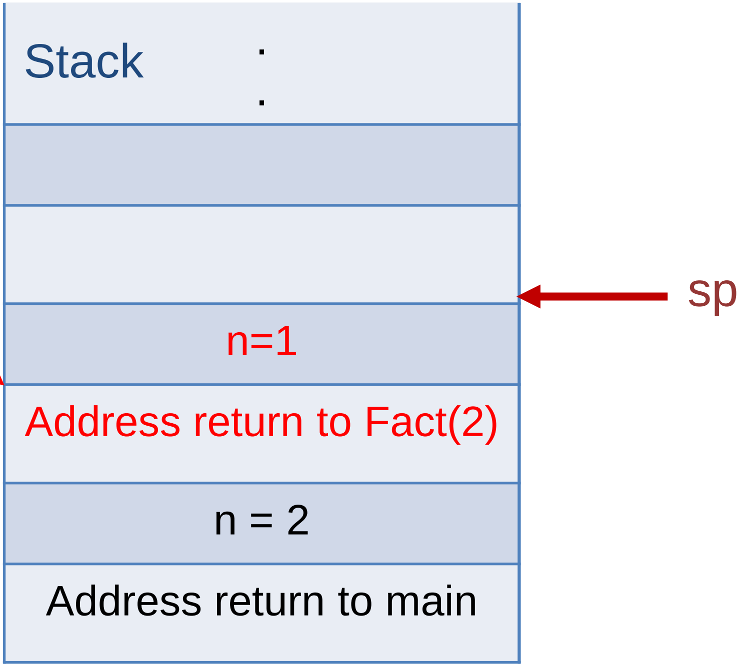


Recursive Example

- Store current n and return address
- Go to n-1

```
fact:
    addi sp, sp, -16     # Allocate stack frame
    sw   ra, 8(sp)       # Save return address
    sw   a0, 0(sp)       # Save argument (n)

    addi t0, a0, -1      # t0 = n - 1
    bge  t0, zero, nfact # if n - 1 >= 0 → recursive case

    # Base case: n <= 0 → return 1
    addi a0, zero, 1
    addi sp, sp, 16      # Free stack frame
    jr x1                # Return

nfact:
    addi a0, a0, -1
    jal  ra, fact        # Recursive call: fact(n-1)

    addi t1, a0, 0       # t1 = fact(n - 1)
    lw   a0, 0(sp)       # Restore original argument (n)
    lw   ra, 8(sp)       # Restore return address
    addi sp, sp, 16      # Free stack frame

    mul a0, a0, t1       # n * fact(n-1)
    ret                  # Return result
```

ra	Address return to Fact(1)
a0(n)	0

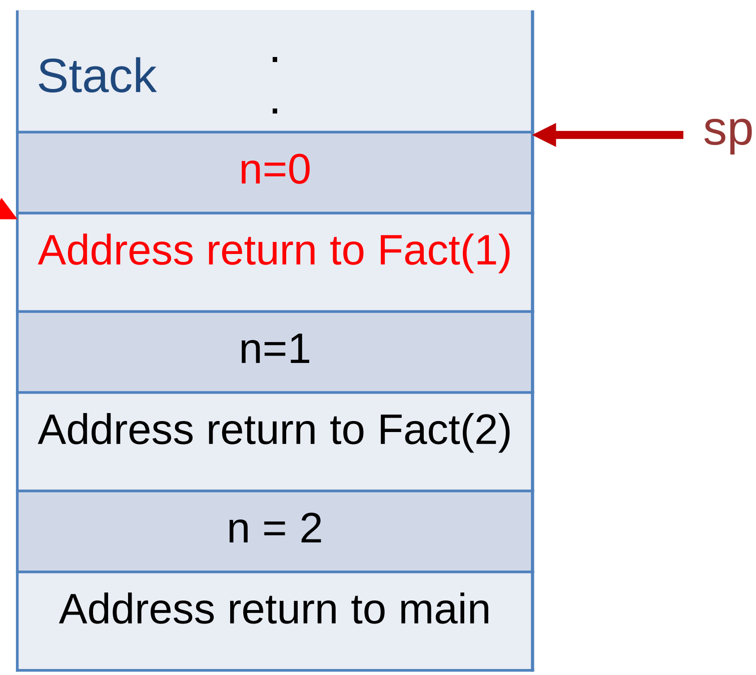


Recursive Example

- Store current n and return address
- Meet base case (n<=0), free stack frame

```
fact:
    addi sp, sp, -16      # Allocate stack frame
    sw   ra, 8(sp)        # Save return address
    sw   a0, 0(sp)        # Save argument (n)

    addi t0, a0, -1       # t0 = n - 1
    bge  t0, zero, nfact  # if n - 1 >= 0 → recursive case

    # Base case: n <= 0 → return 1
    addi a0, zero, 1
    addi sp, sp, 16       # Free stack frame
    jr x1                 # Return

nfact:
    addi a0, a0, -1
    jal  ra, fact         # Recursive call: fact(n-1)

    addi t1, a0, 0        # t1 = fact(n - 1)
    lw   a0, 0(sp)        # Restore original argument (n)
    lw   ra, 8(sp)        # Restore return address
    addi sp, sp, 16       # Free stack frame

    mul a0, a0, t1        # n * fact(n-1)
    ret                   # Return result
```

ra	Address return to Fact(1)
a0	1 (base case)



Recursive Example

- Store fact(n-1) into t1 and store fact(n)=n*fact(n-1) into a0

```
fact:
    addi sp, sp, -16     # Allocate stack frame
    sw   ra, 8(sp)       # Save return address
    sw   a0, 0(sp)       # Save argument (n)

    addi t0, a0, -1      # t0 = n - 1
    bge  t0, zero, nfact # if n - 1 >= 0 → recursive case

    # Base case: n <= 0 → return 1
    addi a0, zero, 1
    addi sp, sp, 16      # Free stack frame
    jr x1                # Return

nfact:
    addi a0, a0, -1
    jal  ra, fact        # Recursive call: fact(n-1)

    addi t1, a0, 0       # t1 = fact(n - 1)
    lw   a0, 0(sp)       # Restore original argument (n)
    lw   ra, 8(sp)       # Restore return address
    addi sp, sp, 16      # Free stack frame

    mul a0, a0, t1       # n * fact(n-1)
    ret                  # Return result
```

t1	1 (= fact(0))
ra	Address return to Fact(2)
a0	1 (n=1)

Stack . .
n=0
Address return to Fact(1)
n=1
Address return to Fact(2)
n = 2
Address return to main

sp

Recursive Example

- Store fact(n-1) into t1 and store fact(n)=n*fact(n-1) into a0

```
fact:
    addi sp, sp, -16     # Allocate stack frame
    sw   ra, 8(sp)       # Save return address
    sw   a0, 0(sp)       # Save argument (n)

    addi t0, a0, -1      # t0 = n - 1
    bge  t0, zero, nfact # if n - 1 >= 0 → recursive case

    # Base case: n <= 0 → return 1
    addi a0, zero, 1
    addi sp, sp, 16      # Free stack frame
    jr x1                # Return

nfact:
    addi a0, a0, -1
    jal  ra, fact        # Recursive call: fact(n-1)

    addi t1, a0, 0       # t1 = fact(n - 1)
    lw   a0, 0(sp)       # Restore original argument (n)
    lw   ra, 8(sp)       # Restore return address
    addi sp, sp, 16      # Free stack frame

    mul a0, a0, t1       # n * fact(n-1)
    ret                  # Return result
```

t1	1 (= fact(0))
ra	Address return to Fact(2)
a0	1 (1*fact(0))

Stack . .
n=0
Address return to Fact(1)
n=1
Address return to Fact(2)
n = 2
Address return to main

sp

Recursive Example

- Store fact(n-1) into t1 and store fact(n)=n*fact(n-1) into a0

```
fact:
    addi sp, sp, -16     # Allocate stack frame
    sw   ra, 8(sp)       # Save return address
    sw   a0, 0(sp)       # Save argument (n)

    addi t0, a0, -1      # t0 = n - 1
    bge  t0, zero, nfact # if n - 1 >= 0 → recursive case

    # Base case: n <= 0 → return 1
    addi a0, zero, 1
    addi sp, sp, 16      # Free stack frame
    jr x1                # Return

nfact:
    addi a0, a0, -1
    jal  ra, fact        # Recursive call: fact(n-1)

    addi t1, a0, 0       # t1 = fact(n - 1)
    lw   a0, 0(sp)       # Restore original argument (n)
    lw   ra, 8(sp)       # Restore return address
    addi sp, sp, 16      # Free stack frame

    mul a0, a0, t1       # n * fact(n-1)
    ret                  # Return result
```

t1	1 (= fact(1))
ra	Address return to main
a0	2 (n=2)

Stack
. .
n=0
Address return to Fact(1)
n=1
Address return to Fact(2)
n = 2
Address return to main

sp

Recursive Example

- Store fact(n-1) into t1 and store fact(n)=n*fact(n-1) into a0

t1	1 (= fact(1))
ra	Address return to main
a0	2 (2*fact(1))

```
fact:
    addi sp, sp, -16      # Allocate stack frame
    sw   ra, 8(sp)        # Save return address
    sw   a0, 0(sp)        # Save argument (n)

    addi t0, a0, -1       # t0 = n - 1
    bge  t0, zero, nfact  # if n - 1 >= 0 → recursive case

    # Base case: n <= 0 → return 1
    addi a0, zero, 1
    addi sp, sp, 16       # Free stack frame
    jr x1                 # Return

nfact:
    addi a0, a0, -1
    jal  ra, fact         # Recursive call: fact(n-1)

    addi t1, a0, 0        # t1 = fact(n - 1)
    lw   a0, 0(sp)        # Restore original argument (n)
    lw   ra, 8(sp)        # Restore return address
    addi sp, sp, 16       # Free stack frame

    mul a0, a0, t1        # n * fact(n-1)
    ret                   # Return result
```

Stack
. .
n=0
Address return to Fact(1)
n=1
Address return to Fact(2)
n = 2
Address return to main

sp

Template of Problem 1

- Input: n in **a0**
- Output: fact(n) in **a0**
- Write you assembly code under fact label only.
- You can change the input number to test functionality. We will use hidden test cases to evaluate your function
- In hidden case: n<500
- Tips: Read factorial.s in RIPES example first.

```

.data
input: .word 7

.text
.global main

# This is 1132 CA Homework 1
# Implement fact(x) = 4*F(floor(n-1)/2) + 8n + 3 , where F(0)=4
# Input: n in a0(x10)
# Output: fact(n) in a0(x10)
# DO NOT MOTIFY "main" function

main:
    # Load input into a0
    lw a0, input

    # Jump to fact
    jal fact

    # You should use ret or jalr x1 to jump back here after function complete
    # Exit program
    # System id for exit is 10 in Ripes, 93 in GEM5 !
    li a7, 10
    ecall

fact:
    # TODO #
```

Problem 2 - Longest Increasing Subsequence

- Given an integer array nums, return the length of the longest strictly increasing subsequence.

- Example:
  - Input: nums = [4, 5, 1, 8, 3, 6, 9, 2]
  - Output: 4
  - Explanation: The longest increasing subsequence is [1,3,6,9], therefore the length is 4.

Problem 2 – LIS Algorithm

- We provide a reference solution using dynamic programming

Algorithm 1 Longest Increasing Subsequence using Dynamic Programming

Require: An array A of length n

Ensure: Length of the Longest Increasing Subsequence (LIS)

Initialize an array dp of size n with all values set to 1
for $i \leftarrow 1$ to $n-1$ **do**
    for $j \leftarrow 0$ to $i-1$ **do**
        if $A[j] < A[i]$ **then**
            $dp[i] \leftarrow \max(dp[i], dp[j]+1)$
        end if
    end for
end for
return $\max(dp)$

Template of Problem 2

- Input:
  - Length in a0
  - Sequence in a1
  - Dp array in a2
- Output: Length of LIS in a0(x10)
- Write your assembly code under LIS label only.
- You can change the input to test functionality. We will use hidden test cases to evaluate your function
- For hidden case: sequence length < =50

```
.data
nums:    .word 4, 5, 1, 8, 3, 6, 9 ,2    # input sequence
n:       .word 8                         # sequence length
dp:      .word 0, 0, 0, 0, 0, 0, 0, 0, 0   # dp array


.text
.globl main

# This is 1132 CA Homework 1 Problem 2
# Implement Longest Increasing Subsequence Algorithm
# Input:
#       sequence length (n) store in a0
#       address of sequence store in a1
#       address of dp array with length n store in a2 (we can decide to use or not)
# Output: Length of Longest Increasing Subsequenc in a0(x10)

# DO NOT MODIFY "main" FUNCTION !!!

main:

    lw a0, n          # a0 = n
    la a1, nums       # a1 = &nums[0]
    la a2, dp         # a2 = &dp[0]

    jal LIS         # Jump to LIS algorithm

    # You should use ret or jalr x1 to jump back after algorithm complete
    # Exit program
    # System id for exit is 10 in Ripes, 93 in GEM5
    li a7, 10
    ecall

LIS:
    # TODO #
```

Common Instruction

Instruction Usage	Meaning	Remarks
add rd, rs1, rs2	rd = rs1 + rs2	add
sub rd, rs1, rs2	rd = rs1 - rs2	subtract
addi rd, rs1, imm	rd = rs1 + imm	add constant
subi rd, rs1, imm	rd = rs1 - imm	subtract constant
sll rd, rs1, rs2	rd = rs1 $\ll$ rs2	Logical left shift by register
srl rd, rs1, rs2	rd = rs1 $\gg$ rs2	Logical right shift by register
slli rd, rs1, imm	rd = rs1 $\ll$ imm	Logical left shift by immediate
srli rd, rs1, imm	rd = rs1 $\gg$ imm	Logical right shift by immediate
lw rd, offset(rs1)	rd = MEM[rs1 + offset]	Loads a 32-bit word
sw rs2, offset(rs1)	MEM[rs1 + offset] = rs2	Stores a 32-bit word
beq rs1, rs2, offset	if (rs1 = rs2) pc += offset	Branches if rs1 = rs2
bne rs1, rs2, offset	if (rs1 $\neq$ rs2) pc += offset	Branches if rs1 $\neq$ rs2
blt rs1, rs2, offset	if (rs1 $<$ rs2) pc += offset	Branches if rs1 $<$ rs2
bge rs1, rs2, offset	if (rs1 $\geq$ rs2) pc += offset	Branches if rs1 $\geq$ rs2
jal rd, offset	rd = pc + 4; pc += offset	Store next address in rd and jumps to the PC-relative offset. If rd=x1, it can be omitted.
jalr rd, rs1, offset	rd = pc + 4; pc = rs1 + offset	Jumps to the address computed from rs1 + offset and stores the return address in rd

More Instruction

Pseudo Instruction	RISC-V	Meaning
nop	addi x0, x0, 0	No operation
not rd, rs	xori rd, rs, -1	Invert all bits
neg rd, rs	sub rd, x0, rs	Generative Opposite number
j offset	jal x0, offset	Jump but not link
jal offset	jal x1, offset	x1 can be omitted
jr rs	jalr x0, rs, 0	x1 can be omitted
jalr rs	jalr x1, rs, 0	x1 can be omitted
ret	jalr x0, x1, 0	Return by x1
mv rd, rs	addi rd, x0, rs	Copy number
li rd, imm	addi rd, x0, imm	Load immediate value
la rd, label	Expands to auipc +addi	Load address of label into rd

Submission

- Deadline: **3/30(sun.) 23:59**
- Upload <student_id>_hw1.zip to NTUCOOL
  - E.g. b11901001_hw1.zip)
- `<student_id>_hw1.zip`
  - `<student_id>_hw1/`
    - `hw1_1.s`
    - `hw1_2.s`
- Wrong file name or format would get **10%** penalty.

Grading Policy

- Each problem is evaluated based on:
  - Passing provided test case: **10%**
  - Passing hidden test cases: **40%**
- We will use normal hidden test data to avoid directly outputting results. You don't need to consider the problems of overflow or out of memory.
- **-10%** for any wrong file name or format for submission
- **No late submission**
- **No plagiarize**
- If you have any problem with homework 1, ask questions through the **NTUCOOL 討論區** or send email to yachi@access.ee.ntu.edu.tw with the subject starting with '[Computer Architecture]'