Computer Architecture HW2

TA: Claire Peng(彭亞綺)

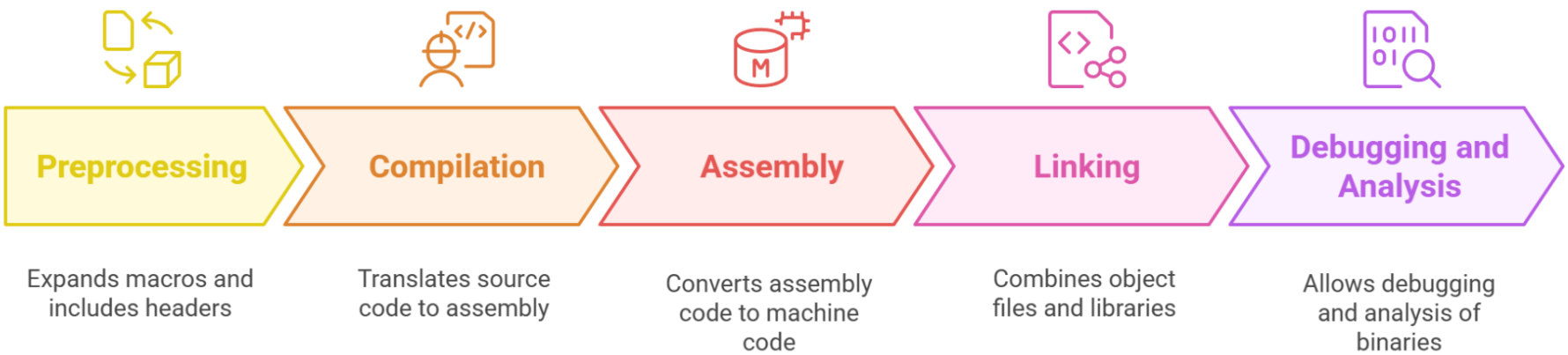
Email: yachi@access.ee.ntu.edu.tw

Table of Contents

- RISCV Toolchain
  - Introduction
  - Uasge
- Homework Description
  - Q1: Forwarding & hazard
  - Q2: Optimizaiton
  - Q3: Compare C++ and assembly
- Submission
- Grading policy

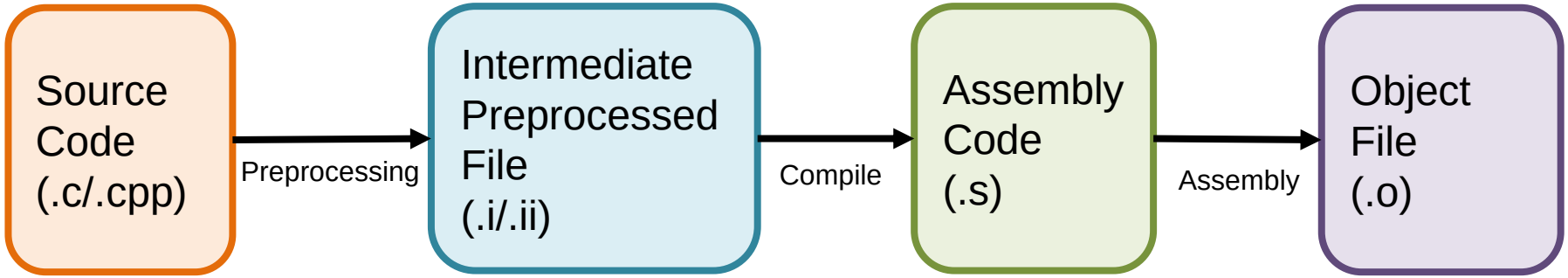
What is toolchain?

- We can use the toolchain to convert high-level languages (C/C++) into assembly and then compile it into executable binary code.
- The relevant toolchain has already been installed in the docker we provided. (riscv64-unknown-linux-gnu toolchain)



What is toolchain?

- **Preprocessing**
  - Expands macros, removes comments, and includes header files to produce a pure C or C++ source file.
- **Compilation**
  - Translate source code into target language.
  - Include optimizations, type checking, and code transformations.
- **Assembly**
  - Convert assembly code into machine code (binary instructions).

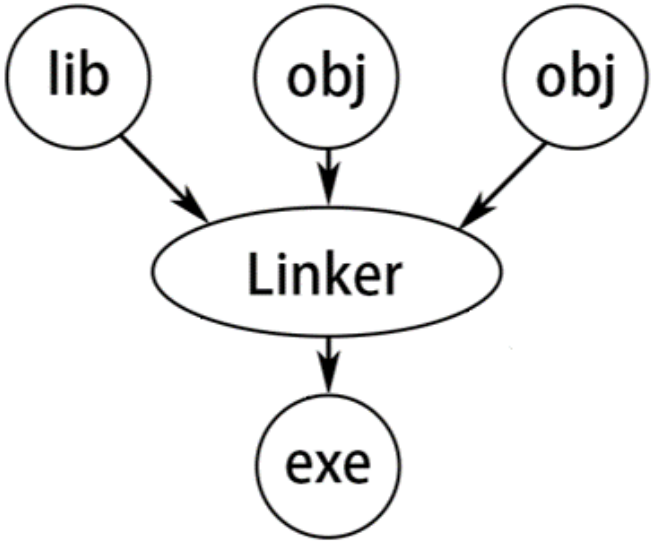


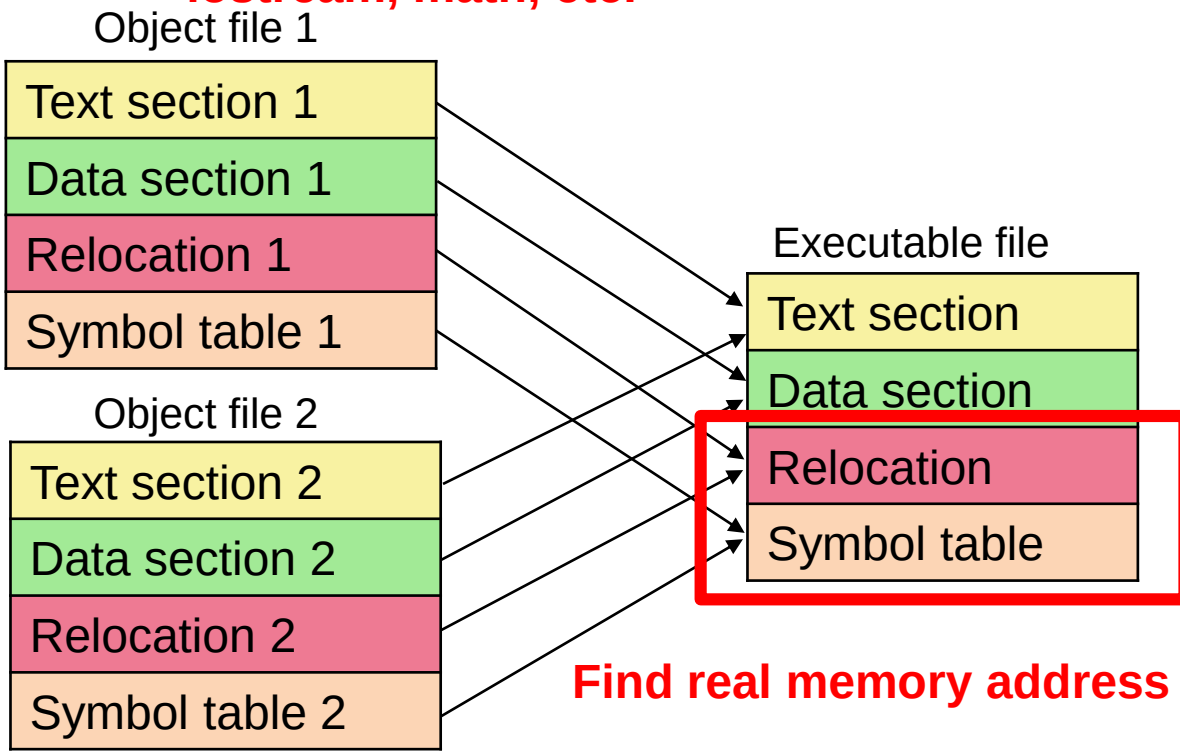
Machine code + symbol

What is toolchain?

- **Linker**
  - Link target file and libraries together and generate an executable file

iostream, math, etc.

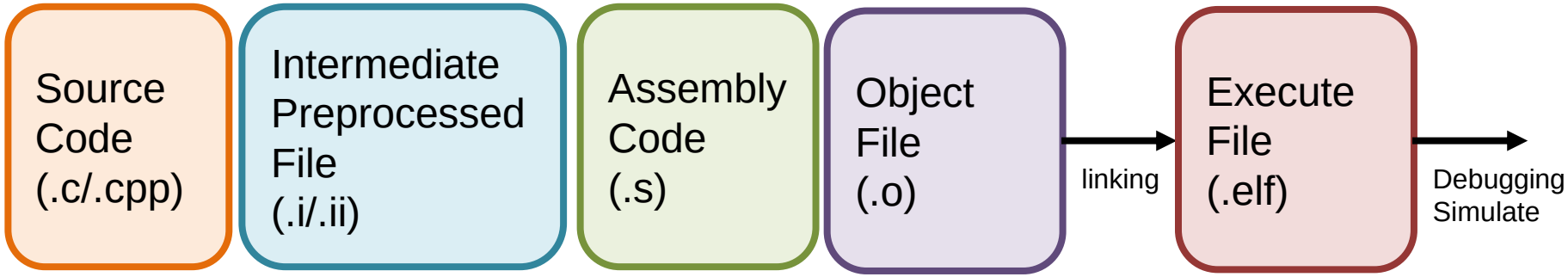




Find real memory address

What is toolchain?

- **Debugging and analyzing**
  - Additional tools like `objdump` (for disassembly), `gdb` (for debugging), and spike or gem5 (for simulation) help analyze and test the generated RISC-V binary.



Executable and Linkable Format

RISCV Toolchain Arguments

- **riscv64-unknown-linux-gnu-g++**
  - `-o <output_file>`: Specify output file name
  - `-E`: Preprocess only. Outputs the preprocessed source file (.ii for C++ or .i for C).
  - `-S`: Generate assembly code. Translates source code into assembly (.s) without assembling it into machine code.
  - `-c`: Compile only. Generates an object file (.o) without linking.
  - `-march`: Specify the target RISC-V core architecture(rv32i,rv64i,rv32gc) and additional instruction set extensions(m,a,f,d,c,g)
  - `-mabi`: Specify the ABI (Application Binary Interface)
    - Defines the calling convention, register usage, and memory alignment rules that compiled code follows
    - E.g. ilp32 refer to 32-bit Integer, Long, Pointer (for rv32)
  - `-O0, -O1, -O2, -O3, -Os`: Optimization levels during compilation

RISCV Toolchain Usage

- We have included the commands we may use in the makefile.
  - `make g++`: Compiles C++ code to an executable file.
  - `make gcc`: Compiles C code to an executable file.
  - `make asm`: Assembles assembly code to an executable file.
  - `make dump_asm`: Disassembles the executable file to assembly code.
  - `make gem5`: Emulate the program on gem5
  - `make clean`: Clean up generated files
  - `make hw2_compile_ccode`: Compile hw2_ccode.cpp and also generate disassembly file
  - `make hw2_compile_assembly`: Co-compile hw2_assembly.cpp and LIS.s and also generate disassembly file

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

Remember to change the program name for different target !

Homework Question 1

- **Q1. (Using Ripes) Imply simulation on hw1_2.s with different processors and make discussion.**
  - **Q1-1.** What is the cycle count when using **5-stage processor** and **5-stage processor w/o forwarding unit**? (5%)
  - **Q1-2.** How does forwarding improve efficiency? Provide a small example from your code.(10%)
  - **Q1-3.** Did your program run correctly under **5-stage processor w/o hazard detection**? What kind of hazard might occur in this setting? Try to fix the program and compare the cycle count with 5-stage processor.(25%)
  - Store your final version into **hw2_1.s**

```
# 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
nop
nop
nop
ecall
```

When using **5-stage processor w/o hazard detection** you need to insert three nop before system call to prevent error

Homework Question 2

- **Q2. Run your Homework 1 assembly code (hw1_2.s) on gem5. Optimize the simulation time and list the changes in simulation time (ticks) before and after the modifications. Describe the methods you used to optimize the simulation time. (30%)**
  - Command you might use: `make asm`, `make g++`
  - Observe detailed simulation information in *m5out/stats.txt* and *out_exec.txt.*
  - Store your final version into **hw2_2.s**

```
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 ex
    li a7, 93
    ecall
```

(PROG := hw2_2)

💡 Modify the exit system call id into **93** when simulating on gem5

Homework Question 2

- **Q2. Run your Homework 1 assembly code (hw1_2.s) on gem5. Optimize the simulation time and list the changes in simulation time (ticks) before and after the modifications. Describe the methods you used to optimize the simulation time. (30%)**
  - Get points according to your simulation time
  - **Don't** use any optimization option (O2,O3…) in toolchain
  - **Don't** modify processor configuration in gem5_config.py
  - Store your final version into **hw2_2.s**

Performance	Score
Function fail	0
Function right + report	10
Function right + report + (tick < 8.18E9)	20
Function right + report + (tick < 8.162E9)	30

Homework Question 3

- **Q3. Implement the same function using both C++ and assembly.**
  - **Q3-1.** Record the instruction count and execution time of two version (10%)
  - **Q3-2.** Discuss your observations. What are the advantages and disadvantages of these two implementations?(20%)

```
int main() {
    int arr[] = {34, 11, 91, 23, 46, 78, 98, 50, 54, 77, 27, 86, 91, 39, 95,
    int n = sizeof(arr) / sizeof(arr[0]);
    int dp[n] = {0};
    int result;

    result = LIS(n, arr, dp);

    char buffer[1024];
    int len = snprintf(buffer, sizeof(buffer), "Your answer = %d\n", result);

    if (len > 0) {
        write(STDOUT_FILENO, buffer, len);
    }

    return 0;
}
```

LIS function

hw2_ccode.cpp

```
int LIS(int n, int* arr, int* dp) {
    // TODO: Implement your code here

    return -1;  // Return the length of the longest incr
}
```

hw2_assembly.cpp

```
extern "C" {
    int LIS(int n, int* arr, int* dp); // Implement
}
```

Homework Question 3

- **Q3. Implement the same function using both C++ and assembly.**
  - **Q3-1.** Record the instruction count and execution time of two version (10%)
  - **Q3-2.** Discuss your observations. What are the advantages and disadvantages of these two implementations?(20%)

hw2_ccode.cpp

```cpp
; function !
ompile_ccode"

#include <unistd.h>
#include <cstdio>

int LIS(int n, int* arr, int* dp) {
    // TODO: Implement your code here

    return -1;  // Return the length of the longest increasing subsequence
}

int main() {
    int arr[] = {34, 11, 91, 23, 46, 78, 98, 50, 54, 77, 27, 86, 91, 39, 95, 41, 57, 45, 55, 28, 68,
    int n = sizeof(arr) / sizeof(arr[0]);
    int dp[n] = {0};
    int result;

    result = LIS(n, arr, dp);

    char buffer[1024];
    int len = snprintf(buffer, sizeof(buffer), "Your answer = %d\n", result); // Gloden answer = 10

    if (len > 0) {
        write(STDOUT_FILENO, buffer, len);
    }

    return 0;
}
```

Step 1.
Implement LIS function in *hw2_ccode.cpp*

Step 2.
Use `make hw2_compile_ccode` to get the executable file *hw2_ccode* and disassembly *hw2_ccode.asm*

Step 3.
Simulate on gem5
(PROG := hw2_ccode)

Homework Question 3

- **Q3. Implement the same function using both C++ and assembly.**
  - **Q3-1.** Record the instruction count and execution time of two version (10%)
  - **Q3-2.** Discuss your observations. What are the advantages and disadvantages of these two implementations?(20%)

hw2_assembly.c

```
ng "make hw2_compile_assembly"
#include <unistd.h>
#include <cstdio>

extern "C" {
    int LIS(int n, int* arr, int* dp); // Implement this function in LIS.s
}

int main() {
    int arr[] = {34, 11, 91, 23, 46, 78, 98, 50, 54, 77, 27, 86, 91, 39, 95, 41, 57, 45, 55, 28, 68,
    int n = sizeof(arr) / sizeof(arr[0]);
    int dp[n] = {0};
```

LIS.s

```
.global LIS

LIS:
    # TODO #
    # copy your code (only LIS section) from hw1_2.s #
```

Step 4.
copy your code (only LIS section) from hw1_2.s into LIS.s

Step 5.
Use `make hw2_compile_assembly` to get the executable file *hw2_assembly* and disassembly *hw2_assembly.asm*

Step 6.
Simulate on gem5
(PROG := hw2_assembly)

Homework Question 3

- **Q3. Implement the same function using both C++ and assembly.**
  - **Q3-1.** Record the instruction count and execution time of two version (10%)
  - **Q3-2.** Discuss your observations. What are the advantages and disadvantages of these two implementations?(20%)

💡 If you function working correctly, you can see the answer of LIS in your terminal(the golden answer of provided test case is 10)

```
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/hw2_ccode'
src/sim/mem_state.cc:448: info: Increasing stack size by one page.
src/sim/syscall_emul.cc:86: warn: ignoring syscall mprotect(...)
Your answer = 10
Exiting @ tick 10252563000 because exiting with last active thread context
Emulated hw2_ccode on gem5 with arguments:
root@4fe5cad0b477:/workspace#
```

Use this for execution time (tick)

Appendix: out_exec.txt

- In gem5, using `-debug_flag=Exec` option enables execution trace logging, which is useful for understanding instruction execution.
- You can search for **@main** to find the program entry point.

```
8143946000: system.cpu: T0 : 0x10550 @main     : addi sp, sp, -1264          : IntAlu :  D=0x000000007ffff9f0
8143995000: system.cpu: T0 : 0x10554 @main+4    : sw ra, 1260(sp)            : MemWrite :  D=0x0000000000010714 A=0x7ffffedc
8144065000: system.cpu: T0 : 0x10558 @main+8    : sw s0, 1256(sp)            : MemWrite :  D=0x0000000000077b68 A=0x7ffffed8
8144135000: system.cpu: T0 : 0x1055c @main+12    : sw s1, 1252(sp)            : MemWrite :  D=0x0000000000000002 A=0x7ffffed4
8144205000: system.cpu: T0 : 0x10560 @main+16    : sw s2, 1248(sp)            : MemWrite :  D=0x0000000000000001 A=0x7ffffed0
8144275000: system.cpu: T0 : 0x10564 @main+20    : sw s3, 1244(sp)            : MemWrite :  D=0x000000007fffff94 A=0x7ffffecc
8144345000: system.cpu: T0 : 0x10568 @main+24    : sw s4, 1240(sp)            : MemWrite :  D=0x000000007fffff9c A=0x7ffffec8
8144415000: system.cpu: T0 : 0x1056c @main+28    : sw s5, 1236(sp)            : MemWrite :  D=0x0000000000000002 A=0x7ffffec4
8144754000: system.cpu: T0 : 0x10570 @main+32    : sw s6, 1232(sp)            : MemWrite :  D=0x0000000000010376 A=0x7ffffec0
8144824000: system.cpu: T0 : 0x10574 @main+36    : sw s7, 1228(sp)            : MemWrite :  D=0x0000000000010140 A=0x7ffffebc
8144894000: system.cpu: T0 : 0x10578 @main+40    : sw s8, 1224(sp)            : MemWrite :  D=0x000000000000003a A=0x7ffffeb8
8144964000: system.cpu: T0 : 0x1057c @main+44    : sw s9, 1220(sp)            : MemWrite :  D=0x000000000007d4c0 A=0x7ffffeb4
8145034000: system.cpu: T0 : 0x10580 @main+48    : addi s0, sp, 1264          : IntAlu :  D=0x000000007ffffee0
```

Simulated time: CPU instance: Thread ID : PC@symbolic location : executed instruction: execution info ….

Submission

- Deadline: **4/27(sun.) 23:59**
- Upload <student_id>_hw2.zip and Report.pdf to NTUCOOL (ex. b11901001_hw2.zip)
- <student_id>_hw2.zip
  - `<student_id>_hw2/`
    - `hw2_1.s` (refer to Q1)
    - `hw2_2.s` (refer to Q2)
    - `hw2_ccode.cpp` (refer to Q3)
    - `LIS.s` (refer to Q3)

- Use provided report template to write report
- Wrong file name or format would get **10%** penalty each.

Grading Policy

- Evaluate score based on report
  - Q1(40%)+Q2(30%)+Q3(30%)=100%
- If the result of your **code does not match the report**, your score for that question will be **reduced by 50%.**
- **-10%** for any wrong file name or format for submission
- No late submission
- No plagiarize
- If your have any problem on hw2, ask question through NTUCOOL 討論區 or send email to yachi@access.ee.ntu.edu.tw with the subject starting with '[Computer Architecture]'