Introduction to Computer Architecture
Project 1

MIPS Binary Code Read

Hyungmin Cho
Department of Software
Sungkyunkwan University

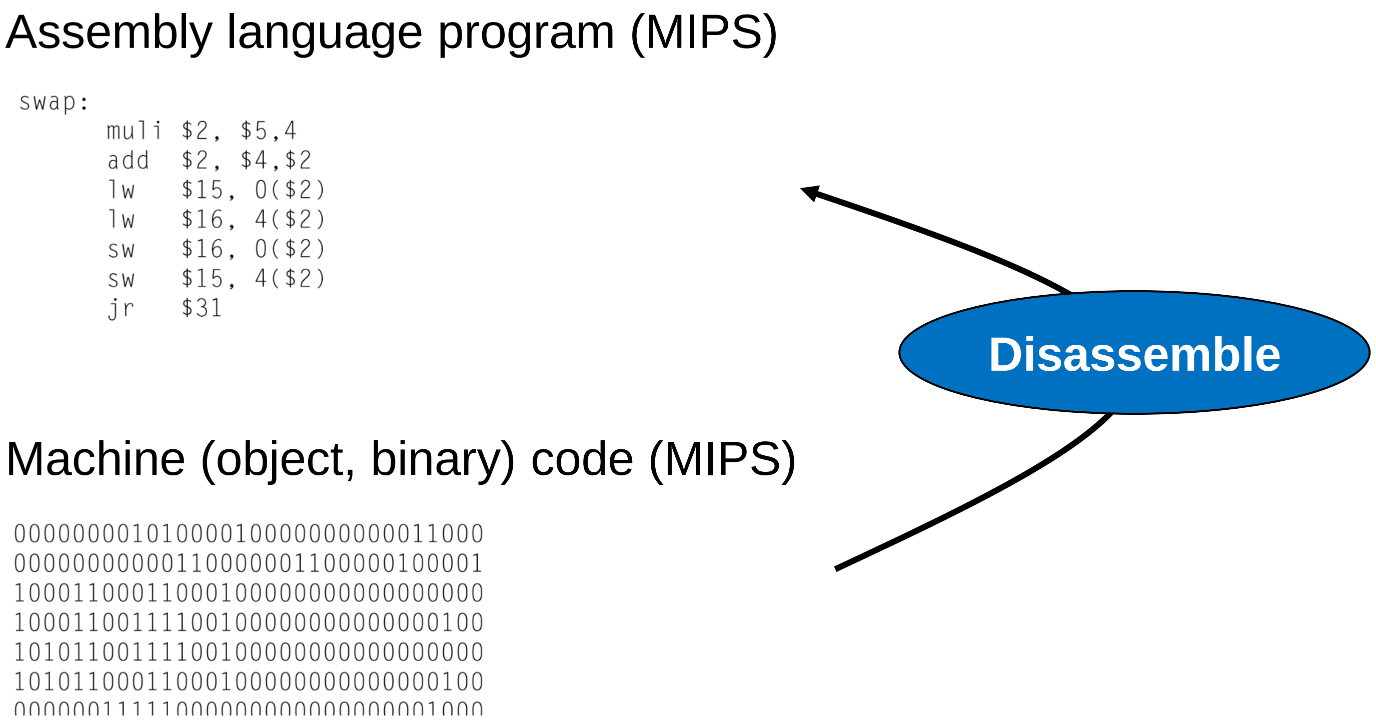
Project Schedule

- Project 1: Interpret MIPS binary code
- Project 2: Simulate a Single-cycle CPU
- Project 3: ?

- Every step depends on the previous one.

Project 1 Goal

- Your program reads a binary file filled with MIPS machine code, and **print the assembly representation of the code**



Assembly language program (MIPS)

```
swap:
      muli $2, $5,4
      add  $2, $4,$2
      lw   $15, 0($2)
      lw   $16, 4($2)
      sw   $16, 0($2)
      sw   $15, 4($2)
      jr   $31
```

Machine (object, binary) code (MIPS)

```
00000000101000010000000000011000
00000000000110000001100000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

Program Interface

- Executable file name
  - The name of the program should be “**`mips-sim`**”
  - If you’re using a language that needs an interpreter (e.g., python), you need to provide a shell script (example on page 14).

- Input
  - The input is a binary file that has MIPS machine codes
  - The input file name is given by the first command-line argument
    - You can assume that the length of the input file name is no greater than 255

- Output
  - Prints the disassembled instruction
  - Each line prints in the following format

```
inst <instruction number>: <32-bit binary code in hex format> <disassembled instruction>
```

Disassembled Instruction Format

- Instruction name in lowercase
  - ❖ `add, sub, sw, jal, …`

- Registers are all represented in numbers
  - ❖ `$0, $1, $20`, …
  - ❖ Do not to use their name (`$s0, $t2`, …)

- Immediate and address values are represented in **signed decimal**
  - ❖ `sw $16,` **`20`**`($29)`
  - ❖ `addi $29, $29,` **`-16`**

Instructions to support

- add, addu, and, div, divu, jalr, jr, mfhi, mflo, mthi, mtlo, mult, multu, nor, or, sll, sllv, slt, sltu, sra, srav, srl, srlv, sub, subu, syscall, xor, addi, addiu, andi, beq, bne, lb, lbu, lh, lhu, lui, lw, ori, sb, slti, sltiu, sh, sw, xori, j, jal

- If there is an instruction that can’t be interpreted, print “unknown instruction”

Things to Consider

- Endianness!
  - Input file (e.g., **test.bin**) uses the big endian format
  - Your computer (x86) uses the little endian format

Execution Results

```
$ ./mips-sim /home/swe3005/2022s/proj1/test1.bin
inst 0: 00220020 add $0, $1, $2
inst 1: 8d420020 lw $2, 32($10)
inst 2: 22300008 addi $16, $17, 8
inst 3: 14400004 bne $2, $0, 4
inst 4: 00000000 sll $0, $0, 0
inst 5: 03e00008 jr $31
inst 6: 00000000 sll $0, $0, 0
inst 7: a7c40008 sh $4, 8($30)
inst 8: 00135940 sll $11, $19, 5
inst 9: 0000000d unknown instruction
          ⋮

$
```

- Your program should print the results to `stdout`
  - i.e., just use normal print functions that prints to the console (e.g., print, printf, …)
- DO NOT save the output to an arbitrary file.

Test Input Files

- You can obtain test input files from the following location of the department servers (`swui.skku.edu`, `swye.skku.edu`, `swji.skku.edu`)
  - ❖ `~swe3005/2022s/proj1/test1.bin`
  - ❖ `~swe3005/2022s/proj1/test2.bin`
  - ❖ …
  - ❖ `~swe3005/2022s/proj1/test8.bin`
- If you want to check the contents of the binary file, you may use the **xxd** program

```
$ xxd /home/swe3005/2022s/proj1/test1.bin
00000000: 0022 0020 8d42 0020 2230 0008 1440 0004
00000010: 0000 0000 03e0 0008 0000 0000 a7c4 0008
00000020: 0013 5940 0000 000d
```

Test Result

- The expected results files are in the following location
  - ~swe3005/2022s/proj1/test1.txt
  - ~swe3005/2022s/proj1/test2.txt
  - …
  - ~swe3005/2022s/proj1/test8.txt

```
$ cat /home/swe3005/2022s/proj1/test1.bin
inst 0: 00220020 add $0, $1, $2
inst 1: 8d420020 lw $2, 32($10)
inst 2: 22300008 addi $16, $17, 8
inst 3: 14400004 bne $2, $0, 4
inst 4: 00000000 sll $0, $0, 0
inst 5: 03e00008 jr $31
inst 6: 00000000 sll $0, $0, 0
inst 7: a7c40008 sh $4, 8($30)
inst 8: 00135940 sll $11, $19, 5
inst 9: 0000000d unknown instruction
```

Test Result

- Your output should EXACTLY MATCH with the reference output.
  - Any difference (e.g., extra character) is considered as a wrong answer

- You can make sure your output is correct using the `diff` command

```
$ ./mips-sim /home/swe3005/2022s/proj1/test1.bin > my_output1.txt

$ diff my_output1.txt /home/swe3005/2022s/proj1/test1.txt

$
```

This will save your output to my_output1.txt

Nothing will be printed if two files match.
Otherwise, it will show the differences.

Project Rule – IMPORTANT!

- You can use any language you'd like to use

- Your submission must be compliable and executable on the department server
  - Caution: some students complained their code is okay on their own PC but fails on the server. In most cases, such differences were caused by a bug in their code. Their bug didn't show up on their own PC, but it somehow showed up on the server. Remember, your submission is scored based on how it behaves on the department server. Make sure to test your program on the server if you created your program locally on your own PC.

- You need to provide a `Makefile` to compile your code
  - Do not need if you're using a script language (e.g., python)
  - The name of the executable should be **`mips-sim`**
  - If your build fails, your project score is **zero**.

- If you're using a script language, you need to provide a shell script that can accept an argument, and the name of the script file should be **`mips-sim`**

Makefile Example

- C

Makefile

```
CC=gcc
CCFLAGS=

#add C source files here
SRCS=main.c

TARGET=mips-sim

OBJS := $(patsubst %.c,%.o,$(SRCS))

all: $(TARGET)

%.o:%.c
          $(CC) $(CCFLAGS) $< -c -o $@

$(TARGET): $(OBJS)
          $(CC) $(CCFLAGS) $^ -o $@

.PHONY=clean

clean:
          rm -f $(OBJS) $(TARGET)
```

- C++

Makefile

```
CXX=g++
CXXFLAGS=

#add C++ source files here
SRCS=main.cc

TARGET=mips-sim

OBJS := $(patsubst %.cc,%.o,$(SRCS))

all: $(TARGET)

%.o:%.cc
          $(CXX) $(CXXFLAGS) $< -c -o $@

$(TARGET): $(OBJS)
          $(CXX) $(CXXFLAGS) $^ -o $@

.PHONY=clean

clean:
          rm -f $(OBJS) $(TARGET)
```

Script Example

- Python (if your python file is mips-sim.py)

`mips-sim` ← Don’t forget to give the excute permission: `chmod +x mips-sim`

```
python3 mips-sim.py $1
```

- Also, be aware of the python version on the server
  - python: python 2.7.17
  - python3: python 3.6.9

Project Environment

- We will use the department's In-Ui-Ye-Ji cluster
  - `swui.skku.edu`
  - `swye.skku.edu`
  - `swji.skku.edu`
  - ssh port: 1398

- First time users :
  - ID: your student ID (e.g., 2020123456)
  - Use the default password (unless you already changed your password…)
    - "pw"+Student_ID (last 8 digits)
    - e.g., The initial password for 2020123456 is `pw20123456`
  - MUST change your password after the first login (Use **`yppasswd`** command)

Submission

- Clear the build directory
  - Do not leave any executable or object file in the submission
  - `make clean`

- Use the `submit` program
  - `~swe3005/bin/submit project_id path_to_submit`
  - If you want to submit the ‘project_1’ directory…
    - **`~swe3005/bin/submit proj1 project_1`**

```
Submitted Files for proj1:
File Name                                 File Size       Time
--------------------------------------------------------------------------------
proj1-2021123456-Sep.05.17.22.388048074        268490                  Thu Sep  5 17:22:49 2021
```

- Verify the submission
  - **`~swe3005/bin/check-submission proj1`**

Project 1 Due Date

- 2022 Apr. 8th, 23:59:59

- **No late submission**