

Computer Architecture

Lab 1: Single-Cycle CPU Implementation

Jaewoong Sim
Electrical and Computer Engineering
Seoul National University

Overview

- Goal: Implement a single-cycle CPU using Verilog HDL
 - Set up the Verilog coding environment
 - Learn how to use Verilog
 - Build a simple, **single-cycle** RISC-V processor based on **RV32I**
- You need to complete the following tasks one by one
 - **Task 0: Set up Verilog coding environment**
 - Task 1: Complete uarch for **arithmetic (R-Type)** instructions
 - Task 2: Implement uarch for **immediate arith. (I-Type)** instructions
 - Task 3: Implement uarch for **load (I-Type) & store (S-Type)** instructions
 - Task 4: Implement uarch for **branch (B-Type)** instructions
 - Task 5: Implement uarch for **jump (J/I-Type)** instructions
- We provide you with an *almost complete* R-Type implementation, which you need to build on
- We also provide you with test code for each task

RV32I Base Integer Instructions

RV32I Base Integer Instructions

	Inst	Name	FMT	Opcode	funct3	funct7	Description (C)	Note
Task 1	add	ADD	R	0110011	0x0	0x00	rd = rs1 + rs2	
	sub	SUB	R	0110011	0x0	0x20	rd = rs1 - rs2	
	xor	XOR	R	0110011	0x4	0x00	rd = rs1 ^ rs2	
	or	OR	R	0110011	0x6	0x00	rd = rs1 rs2	
	and	AND	R	0110011	0x7	0x00	rd = rs1 & rs2	
	sll	Shift Left Logical	R	0110011	0x1	0x00	rd = rs1 << rs2	
	srl	Shift Right Logical	R	0110011	0x5	0x00	rd = rs1 >> rs2	
	sra	Shift Right Arith*	R	0110011	0x5	0x20	rd = rs1 >> rs2	msb-extends
	slt	Set Less Than	R	0110011	0x2	0x00	rd = (rs1 < rs2)?1:0	
	sltu	Set Less Than (U)	R	0110011	0x3	0x00	rd = (rs1 < rs2)?1:0	zero-extends
Task 2	addi	ADD Immediate	I	0010011	0x0		rd = rs1 + imm	
	xori	XOR Immediate	I	0010011	0x4		rd = rs1 ^ imm	
	ori	OR Immediate	I	0010011	0x6		rd = rs1 imm	
	andi	AND Immediate	I	0010011	0x7		rd = rs1 & imm	
	slli	Shift Left Logical Imm	I	0010011	0x1	imm[5:11]=0x00	rd = rs1 << imm[0:4]	
	srlr	Shift Right Logical Imm	I	0010011	0x5	imm[5:11]=0x00	rd = rs1 >> imm[0:4]	
	srai	Shift Right Arith Imm	I	0010011	0x5	imm[5:11]=0x20	rd = rs1 >> imm[0:4]	msb-extends
	slti	Set Less Than Imm	I	0010011	0x2		rd = (rs1 < imm)?1:0	
	sltiu	Set Less Than Imm (U)	I	0010011	0x3		rd = (rs1 < imm)?1:0	zero-extends
Task 3	lb	Load Byte	I	0000011	0x0		rd = M[rs1+imm][0:7]	
	lh	Load Half	I	0000011	0x1		rd = M[rs1+imm][0:15]	
	lw	Load Word	I	0000011	0x2		rd = M[rs1+imm][0:31]	
	lbu	Load Byte (U)	I	0000011	0x4		rd = M[rs1+imm][0:7]	zero-extends
	lhu	Load Half (U)	I	0000011	0x5		rd = M[rs1+imm][0:15]	zero-extends
	sb	Store Byte	S	0100011	0x0		M[rs1+imm][0:7] = rs2[0:7]	
Task 4	sh	Store Half	S	0100011	0x1		M[rs1+imm][0:15] = rs2[0:15]	
	sw	Store Word	S	0100011	0x2		M[rs1+imm][0:31] = rs2[0:31]	
	beq	Branch ==	B	1100011	0x0		if(rs1 == rs2) PC += imm	
	bne	Branch !=	B	1100011	0x1		if(rs1 != rs2) PC += imm	
	blt	Branch <	B	1100011	0x4		if(rs1 < rs2) PC += imm	
	bge	Branch ≥	B	1100011	0x5		if(rs1 ≥ rs2) PC += imm	
Task 5	bltu	Branch < (U)	B	1100011	0x6		if(rs1 < rs2) PC += imm	zero-extends
	bgeu	Branch ≥ (U)	B	1100011	0x7		if(rs1 ≥ rs2) PC += imm	zero-extends
	jal	Jump And Link	J	1101111			rd = PC+4; PC += imm	
	jalr	Jump And Link Reg	I	1100111	0x0		rd = PC+4; PC = rs1 + imm	
	lui	Load Upper Imm	U	0110111			rd = imm << 12	
	auipc	Add Upper Imm to PC	U	0010111			rd = PC + (imm << 12)	
	ecall	Environment Call	I	1110011	0x0	imm=0x0	Transfer control to OS	
	ebreak	Environment Break	I	1110011	0x0	imm=0x1	Transfer control to debugger	

Honor Code

DO NOT CHEAT!

**Everyone who involved in violation
will get zero scores**

Environment Setup

Setting Up the Verilog Coding Environment

- We recommend you use lightweight, open-source tools as shown below
 - It is okay to use other tools such as Xilinx Vivado if you want or have them
- Icarus Verilog
 - Open-Source Verilog compiler and simulator
- GTKWave
 - Waveform viewer
 - Use when we debug the Verilog code

Setting Up the Verilog Coding Environment

Windows

Windows Subsystem for Linux 2 (WSL2) for Windows 10

- WSL2 is supported on version 1903, 1909, 2004, 20H2, or higher
➔ Update Windows 10 accordingly
- Run Windows PowerShell as administrator (right click)
- Copy the command below
 - `dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart`
 - `dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart`
- Download WSL 2 kernel update package (click the link below)

https://wslstorestorage.blob.core.windows.net/wslblob/wsl_update_x64.msi

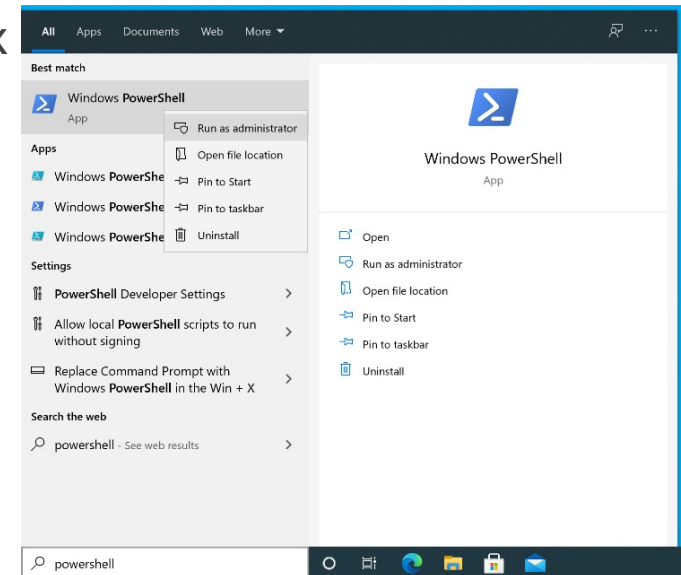
- Set WSL2 as a default version & Download Linux

➤ `wsl --set-default-version 2`

- Download 'Ubuntu 20.04 LTS' in Microsoft Store

- Open Ubuntu 20.04 LTS
& Enter your UNIX username

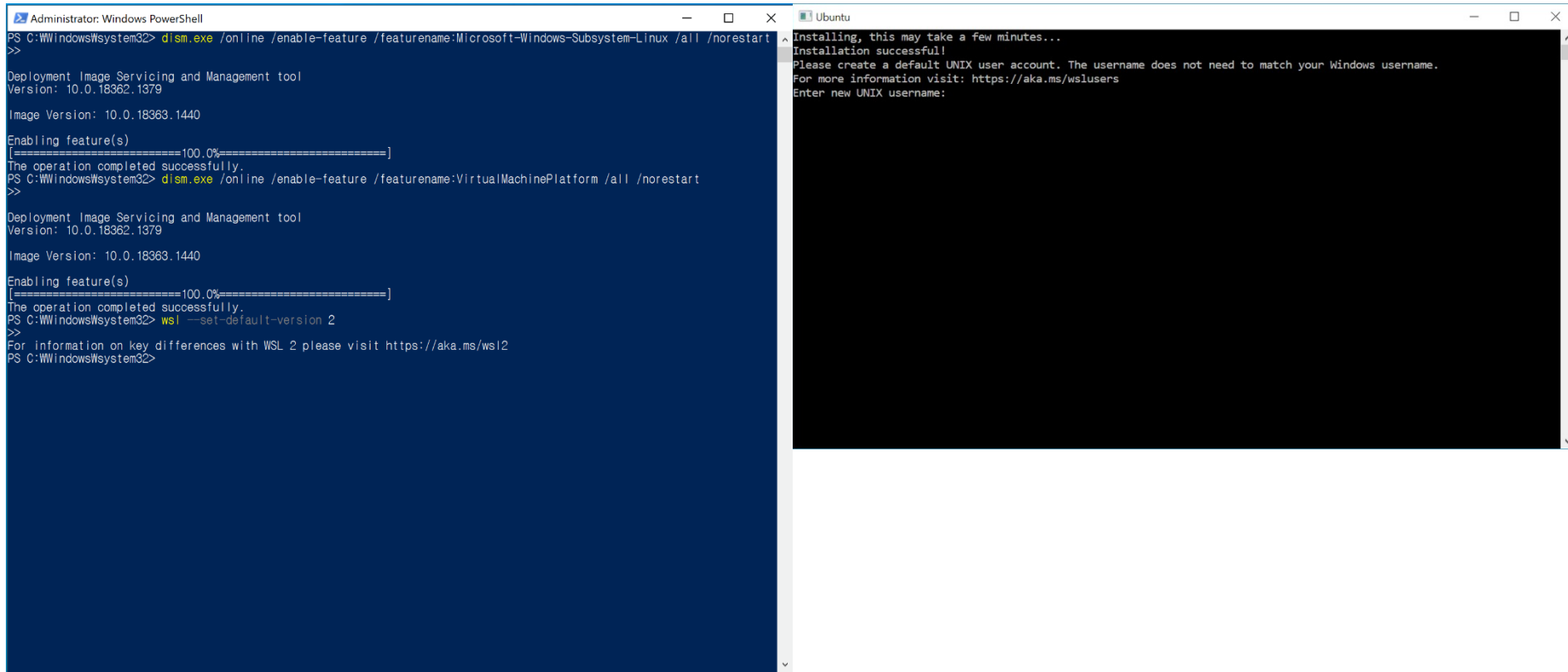
- `$ sudo apt update`
- `$ sudo apt install iverilog gtkwave git`



Setting Up the Verilog Coding Environment

Windows

- WSL screenshots



```
Administrator: Windows PowerShell
PS C:\Windows\System32> dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart
>>

Deployment Image Servicing and Management tool
Version: 10.0.18362.1379

Image Version: 10.0.18363.1440

Enabling feature(s)
[=====100.0%=====]
The operation completed successfully.
PS C:\Windows\System32> dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart
>>

Deployment Image Servicing and Management tool
Version: 10.0.18362.1379

Image Version: 10.0.18363.1440

Enabling feature(s)
[=====100.0%=====]
The operation completed successfully.
PS C:\Windows\System32> wsl --set-default-version 2
>>
For information on key differences with WSL 2 please visit https://aka.ms/wsl2
PS C:\Windows\System32>
```

```
Ubuntu
Installing, this may take a few minutes...
Installation successful!
Please create a default UNIX user account. The username does not need to match your Windows username.
For more information visit: https://aka.ms/wslusers
Enter new UNIX username:
```


Setting Up the Verilog Coding Environment

Windows (WSL)

X11 Forwarding

- To enabling X11 forwarding, we can use **Xming**
 - Download and Install Xming
 - <https://sourceforge.net/projects/xming/>
 - Refer this website: <https://gyusday.tistory.com/13>
 - After installation, edit your configuration file (~/.bashrc or ~/.zshrc)
 - ▶ 1) Check your shell: `$ echo $SHELL`
 - ▶ 2) Open your configuration file: `$ vim ~/.bashrc` (or `$ vim ~/.zshrc`)
 - ▶ 3) Write & Save: `export DISPLAY="localhost:0.0"`
 - ▶ 4) Apply your change: `source ~/.bashrc`
 - ▶ If you have any problem, refer this website
 - <https://gyusday.tistory.com/12>

Setting Up the Verilog Coding Environment

Linux/Mac OS

Linux (Ubuntu 20.04/18.04 LTS)

- `$ sudo apt-get install iverilog gtkwave git`

Mac OS

- You can use “homebrew” to install icarus verilog
 - Install Homebrew,
 - ▶ `/bin/bash -c “$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)”`
 - Install icarus Verilog & GTKWave
 - ▶ `brew install icarus-verilog`
 - ▶ `brew install gtkwave`
 - X11 forwarding: Install xquartz (<https://www.cyberciti.biz/faq/apple-osx-mountain-lion-mavericks-install-xquartz-server/>)

Test Your Framework

- Copy and paste the code on the right and save it as “hello.v”

```
module hello;

reg clk;

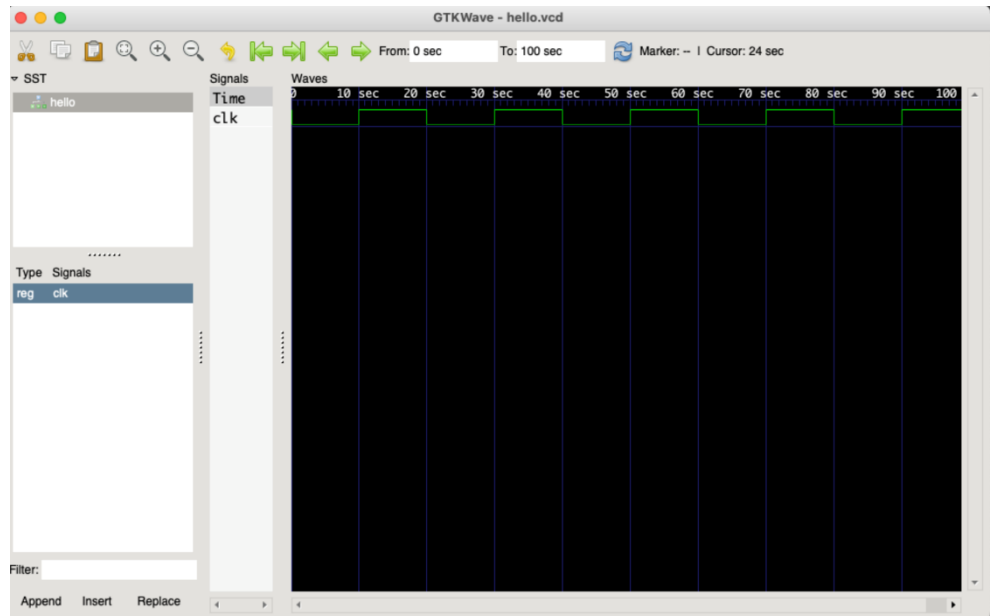
initial begin
    clk = 1'b0;
    $monitor("@ %2d Hello World ", clk, $time);
    #100 $finish;
end

always begin
    #10 clk = ~clk;
end

// dump the state of the design
// VCD (Value Change Dump) is a standard dump
// format defined in Verilog.
initial begin
    $dumpfile("hello.vcd");
    $dumpvars(0, hello);
end

endmodule
```

- Compile “hello.v” and run it
 - iverilog -o hello hello.v
 - vvp hello
 - gtkwave hello.vcd



How to Compile, Run, Test Your Code

Linux & macOS & WSL

- How to compile & run your code
 - run “make” inside your project directory
 - this will generate a “simple_cpu” binary file
 - “./simple_cpu” will execute the code

```
(base) joonhohwangbo ~/Desktop/ece322/lab1/student_version master ls
Makefile data src test.py
(base) joonhohwangbo ~/Desktop/ece322/lab1/student_version master make
iverilog -I src/modules/ -s riscv_tb -o simple_cpu src/modules/ALU.v src/modules/ALU_control.v
src/modules/defines.v src/modules/register_file.v src/modules/control.v src/modules/data_memory.v
src/modules/mux_2x1.v src/modules/mux_4x1.v src/modules/imm_generator.v src/modules/adder.v
src/modules/branch_control.v src/riscv_tb.v src/simple_cpu.v
(base) joonhohwangbo ~/Desktop/ece322/lab1/student_version master ls
Makefile data simple_cpu src test.py
(base) joonhohwangbo ~/Desktop/ece322/lab1/student_version master ./simple_cpu
```

Linux & macOS & WSL

- **Recompile** your code using “make” if you change your code

Linux & macOS & WSL

- We provide some python code for automatically running and comparing your verilog code output with the correct resulting registers in “test.py”
- Run “python3 test.py”

```
(whale) joonhohwangbo > ~/Desktop/ece322/lab1/student_version > master ± > python test.py
[*] test start
starting test task1/1
your register values should be
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 3, -4, 6, 7, 1, -5, 0, 0, 0, 0, 0, 0, 0, 0, 0]
your verilog register values are
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 3, -4, 6, 7, 1, -5, 0, 0, 0, 0, 0, 0, 0, 0, 0]
test passed
-----
```

The provided code should run task1/1 successfully!

Linux & macOS & WSL

- Because the test code is dependent of the path of your files, please try not to change the location of files
- In case you want to change the directory structure, you should change the test code too!!

Linux & macOS & WSL

- Test instruction sequences are inside the “data” directory
 - e.g., “./data/task2/1/inst_disassembled.mem” contains instructions for immediate arithmetic (I-type) instructions
 - e.g., “./data/task2/2/inst_disassembled.mem” contains **more** immediate arithmetic instructions
- Correct register values after running your code are inside reg-out.mem
 - e.g., “./data/task2/1/reg-out.mem” contains correct register values after testing with “./data/task2/1/inst_dissassembled.mem”

Linux & macOS & WSL

- Test Benches
 - Task1 (2 tests): Arithmetic (R-Type)
 - Task2 (2 tests) : Immediate Arithmetic (I-Type)
 - Task3 (2 tests): Load (I-Type) & Store (B-Type)
 - Task4 (1 test): Branch (B-Type)
 - Task5 (1 test): Jump (J/I-Type)
 - Final Task : Fibonacci sequence
- “test.py” may fail to execute if you do not have Python installed
 - Run “python --version” to check if you have python 3 installed

```
(whale) joonhohwangbo ~ python --version
Python 3.8.5
```

- If there is no python, there should be no output (or errors)

Linux & macOS & WSL

- macOS & Ubuntu 20.04
 - Python is preinstalled so there should be no errors while executing 'test.py'
- install python in Linux (ubuntu 18.04 & before)
 - `$ sudo apt update`
 - `$ sudo apt install software-properties-common`
 - `$ sudo add-apt-repository ppa:deadsnakes/ppa`
 - `$ sudo apt update`
 - `$ sudo apt install python3.8`
 - `$ python --version` #check version of python
- wsl
 - `$ sudo apt update && upgrade`
 - `$ sudo apt install python3 python3-pip ipython`
 - `$ python --version`

Submission

- **Due:** 4/13 (**Wednesday**) **11:59 PM**
- **Late Policy**
 - 20% discounted per day (4/14 12:00 AM is a late submission!)
 - After 4/17 12:00 AM, you will get a **zero** score for the assignment
- **What to submit?**
 - Your code that we can simulate and run
- **How to submit?**
 - Upload your compressed file (zip/tar/etc) to eTL
 - **Format:** Your student ID_YOURLASTNAME_lab#
 - e.g.) 2020-12345_KIM_lab1.zip
 - Please make sure you follow the format.
 - **10% penalty for the wrong format**