**Week 1**

**Summary of week’s activities:**

1. Install the running environment (including riscv32-unknown-elf-gcc and qemu-system-riscv32). riscv32-unknown-elf-gcc is the cross compile, which can compile the operating system source code so that it can be used in 32 bit RISC-V. Qemu-system-riscv32 can simulate the 32 bit RISC-V environment on Ubuntu.
2. After finishing the installation, write a simple C program to test the correctness. Firstly, use the riscv32-unknown-elf-gcc to compile the source code and generate an executable file. Then, use qemu-system-riscv32 to run this executable file. This process is shown in following figure.
3. Read the book called <Computer Organization and Design RISC-V Edition: The Hardware Software Interface>. This is a thick book for me to read it because I lack knowledge of computer systems. I particularly focused on the second part <Instruction: Language of the Computer>.
4. Discuss what the project needs to do (general construction of separate part)
5. Search reference of the system on chip and CPU core
6. Reference Book: Computer Organization and Design RISC-V edition, Digital Design and Computer Architecture, Introduction to Computing System
7. Reference website or report: "Longxin" Cup final project, lowRISC (constructed by University of Cambridge), Xlinx forum, Digilent forum, Design of Digital Circuit (course of ETH Zurich)
8. Test the function of Digilent Nexys A7 board
9. Create the github page and determine the timetable of separate parts
10. Discuss what the expected demo of the project with group members

**Problem, issues and concerns:**

1. Fail to configure the OS at first (do not know how to set the compiler and simulator because of lack of knowledge in Makefile).

**Tasks for next week/Actions for next meeting:**

1. achieve some shell functions, such as sleep and find.

2. ALU design of RV32I and simple control logic unit

3. Read information about u-boot

**Week 2**

**Summary of week’s activities:**

Implement several shell commands, such as sleep, pingpong, primes, find and xargs.

1. sleep can pause the operating system for a given time. (eg. sleep 100: the system will pause for 10 second)
2. *pingpong* can transmit a byte message between two processors via a pair of pipes, *parent\_pipe* and *child\_pipe*, one for each direction (in and out). The parent process will write message to *parent\_pipe[1]* and the child receive this message from *parent\_pipe[0]*. Then child responds with writing a message to *child\_pipe[1]* and parent can read from *child\_pipe[0].*
3. primes is based on the idea Unix Pipes. The first process feeds the numbers 2 through 35 into the pipeline. For each prime number, you will arrange to create one process that reads from its left neighbor over a pipe and writes to its right neighbor over another pipe. All the numbers can be divided by the prime number will be discarded as shown in the following figure.
4. *find* is used to find the directory or file under the given path. For example, we make a directory *aaa* and then create a file *bbb* inside *aaa*. If we want to find all files called *bbb* in current directory, we just need to write *find . bbb*. The second argument . means it starts in the current directory.
5. The reading plan should be completed last week took an extra week because it could not be read.
6. Discuss which instruction set the operating system (xv6 or Linux kernel) needs (RV32IM)
7. Construct the simple Arithmetic Logic Unit Structure on Vivado 2019.1 (commit detail)
8. Update ALU structure and support RV32M extension without verification (commits on Feb 8, 2020)
9. Update ALU Control Unit without verification (commits on Feb 8, 2020)
10. Construct register file (32 registers following RISC-V specification)

**Problem, issues and concerns:**

1. How is ECALL achieved

**Tasks for next week/Actions for next meeting:**

1. Build U-boot

2. write a simulating shell (shell-in-shell)

3. Physical memory allocation

4. Memory controller design and test

**Week 3**

**Summary of week’s activities:**

1. Implement a simulated shell, called nsh. When running the operating system, write nsh in command line to start. In this simulated shell, we can also run some commands as in the original shell.
2. Implement the buddy allocation (kernel/buddy.c), which allocates and free file structs so that xv6 can have more open file descriptors.
3. Design how to create address space. The central data structure is pagetable\_t (it can be used for both kernel page table and process page table), which is really a pointer to a RISC-V root page-table page. The walk function will find the PTE for a virtual address, and mappages, which installs PTEs for new mappings.
4. Search for the knowledge about u-boot on the Internet, watch videos about u-boot uploaded by others online, and learn how to install and write u-boot based on the tutorial.
5. Construct Control Unit, and test ALU control and register file (commits on Feb 10, 2020)
6. Test the ALU with new extension support and fix the bugs (commits on Feb 9, 2020)
7. Confirm the function of SDRAM to DDR Component provided by Digilent on board (commits on Feb 12, 2020)
8. Construct register between each stage used in pipeline processor
9. Test Control Unit and fix the bugs (commits on Feb 15, 2020)

**Problem, issues and concerns:**

1. Long memory access latency.

2. Need faster algorithm for buddy allocation.

**Tasks for next week/Actions for next meeting:**

1. Test U-boot and integrate with OS

2. Drivers on soc design.

3. lazy allocation, alarm call, file system and memory map

**Week 4**

**Summary of week’s activities:**

1. Implement lazy allocation, which will only allocate resource until it is actually needed.
2. It was found that the u-boot transplantation tutorial on the Internet was performed on the Samsung template, which is different from the source code used by risc-v.
3. Discuss how the operating system communicate with the CPU (system call)
4. Search information to solve the problem of DDR2 SDRAM, which cause the whole project not to meet the expectation progress.
5. Replace on chip DDR2 RAM with LUT RAM and Block RAM
   1. Cannot support the compile file of operating system. Therefore, memory support is necessary

**Problem, issues and concerns:**

1. Not finish U-boot on time

2. Difficult to achieve the alarm mechanism

3. No time for doing file system

**Tasks for next week/Actions for next meeting:**

1. Continue and test U-boot

2. Drivers design

3. Rewrite file system

4. Copy-on-write

**Week 5**

**Summary of week’s activities:**

1. Design a simple file system. All files in xv6 are limited to 268 blocks. This design has limited function but at least works.^\_^
2. Keyboard driver, console driver, time interrupt.
3. Write this project report about Sustainable Development and Ethics, and continue to search online about risc-v u-boot.
4. Integrate all the components to be a pipeline processor with 5 stages (commits on Feb 27, 2020)
5. Construct Hazard Unit to solve control dependencies and data dependencies by data forwarding and pipeline stall (commits on Feb 28, 2020)
6. Construct Static branch prediction (commits on Feb 28, 2020)

**Problem, issues and concerns:**

1. Do not have enough time to finish Copy-on-write allocation, but it can be replaced by lazy allocation and buddy allocation, which have been finished before.
2. Perhaps this OS cannot meet the initial goal, such as connecting to internet.

**Tasks for next week/Actions for next meeting:**

1. Integrate the project

2. Test bootloader, transplant OS on RISC-V

3. Ethernet driver and VGA driver design

**Week 6**

**Summary of week’s activities:**

1. Compile xv6. Since the CPU is 32 bit, the source code must be modified (mainly change pointer's increasing step length). Set the compile tool and simulating tool in Makefile to riscv32-unknown-elf-gcc and qemu-system-riscv32.
2. Booting the U-boot to the memory so that it can read the kernel from USB.
3. Participate in designing and editing posters.
4. Construct communication between cache system and memory by AXI4 full bus (commits on Mar 3, 2020)
5. Construct communication between cache and cpu core (commits on Mar 3, 2020)
   1. Asynchronous FIFO (Xilinx Parameterized Macro)
6. Try to integrate operating system into the SoC
7. Construct Cache System (commits on Mar 2, 2020)
   1. Instruction cache: Single Port Distributed RAM (Xilinx Parameterized Macro)
   2. Data cache Single Port Block RAM (Xilinx Parameterized Macro)

**Problem, issues and concerns:**

1. No GPIO, No USB, Cache system have unknown bugs, boot cannot correctly load the system.

**Further improvement:**

1. System on Chip

a. General-purpose input/output (GPIO) to AXI4 interface (memory)

b. USB driver to AXI4 interface (memory)

c. PS/2 and VGA driver to AXI4 interface (memory)

d. Complete verification of all the components

2. CPU Core

a. Reorder buffer and Out of order execution

b. Dynamic branch prediction

i: Reference document: S. McFarling, "Combining Branch Predictors" DEC WRL Technical Report" 1993

c. Superscalar processor