Programming Assignment 1 - Formal Design
Design for Basic RISC-V Execution
Group 55

Maxwell Bruce
Samuel Woolledge
Jiapei Kuang

Goal:

To simulate a single-cycle RISC-V processor in C that can take in the current value of the program counter and executes a single function. Our simulator are able to read/write memory, read/write the register file, and produce a new value for the program counter. Therefore, when the users provide a simple assembly language program, it can be converted and executed in RISC-V assembly.

Basic Process of the Simulator:

After loads the provided file with assembly language instructions, the simulator calls the function execute_single_instruction(const uint64_t pc, uint64_t*new_pc) for each instruction. The simulator will decide which instruction function to call based on the assembly instruction. It will execute the instruction function for reads/writes to memory or the registers.

Structure:

Header file:

The simulator needs header files for defining the data types for functions that the team will implement later. Data type for core instruction formats are written based on the following chart from Green Card.

CORE INSTRUCTION FORMATS

	31 27	26 25	24 20	19 15	14 12	11 7	6 0
R	funct7		rs2	rs1	funct3	rd	Opcode
I	imm[11:0]			rs1	funct3	rd	Opcode
S	imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode
SB	imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode
U	imm[31:12]					rd	opcode
UJ	imm[20|10:1|11|19:12]					rd	opcode

There should be 4 data types in header file, R-type, I-type, S-type(combination of S and SB type instruction), and U-type(combination of U and UJ type instruction).

Functions provided by Professor:

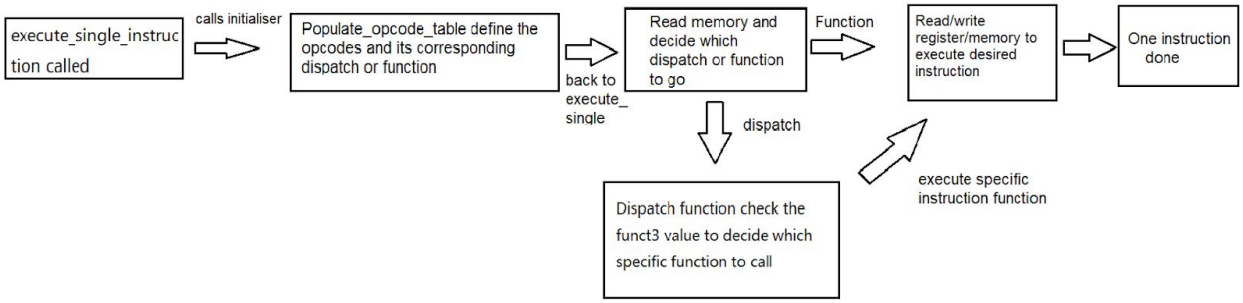
- **Memory**
  - **memory _read**
  - **memory _write**
  - **memory_status**

First call can only be memory_read() and the second call can be memory_read or memory_write. No further calls allow.

- **Registers**
  - **register_read**
  - **register_write**

Call each function once, further calls will be 0. The function will mask off the register identifiers to be 5 bits.

For the main part of the simulator, below is a brief flow chart of how the simulator execute one instruction. And all the functions that need to be defined are listed and described below.



Functions that need to be specified:

- **execute_single_instruction**

  It will read memory from pc to get the instructions. And by checking the definition initialized by opcode_tables function, to decide which instruction dispatch to execute.

- **Populate_opcode_tables**

  Every instruction have an opcode part as an id to help identify which instruction it is. This populate_opcode_tables function include the instruction dispatch (such as riscv_jal, riscv_load) and their corresponding opcodes from the Green Card (example below). It will help the execute_single_instruction function to decide which instruction to execute.

  Example:
  major_dispatch_table[0b0000011] = riscv_load_dispatch;

OPCODES IN NUMERICAL ORDER BY OPCODE

MNEMONIC	FMT	OPCODE	FUNCT3	FUNCT7 OR IMM	HEXADECIMAL
lb	I	0000011	000		03/0

- **Dispatch**

In each dispatch, it will have a type of instructions that the simulator needs to execute. By reading the opcode of FUNT3 in the Green Card, it will decide which specific instruction to execute.

Though not every instructions have dispatch. For special cases, auipc, lui, Jal, jalr are individual functions and one function that does nothings for fence, CSR, SCALL, EBREAK.

Instruction function that needs to be implemented：

- **auipc**
- **lui**
- **jal**
- **jalr**
- **Load dispatch**
  - **lb, lh, lw, ld**
  - **lbu, lhu, lwu, ldu (unsigned)**
- **Branch dispatch**
  - **beq, bne**
  - **blt, ge**
  - **bltu, bgeu (unsigned)**
- **Store dispatch**
  - **sb, sh, sw, sd**
- **Arithmetic 1 dispatch (also contains RV64)**
  - **addiw, addi**
  - **slliw, slli**
  - **slti, sltiu**
  - **srliw, srli**
  - **srai, sraiw**
  - **xori**
  - **andi**
  - **ori**
- **Arithmetic 2 dispatch(also contains RV64)**
  - **addw, add**
  - **subw, sub**
  - **mulw, mul**
  - **sllw, sll**
  - **mulh, mulhsu**
  - **slt, sltu**
  - **mulhu**
  - **xor**
  - **divw, div**

    - **srl**
    - **sraw, sra**
    - **divuw, divu**
    - **or**
    - **remw, rem**
    - **remuw, remu**
    - **and**
- **Nop**
    - **fence**
    - **CSR/SCALL/EBREAK**