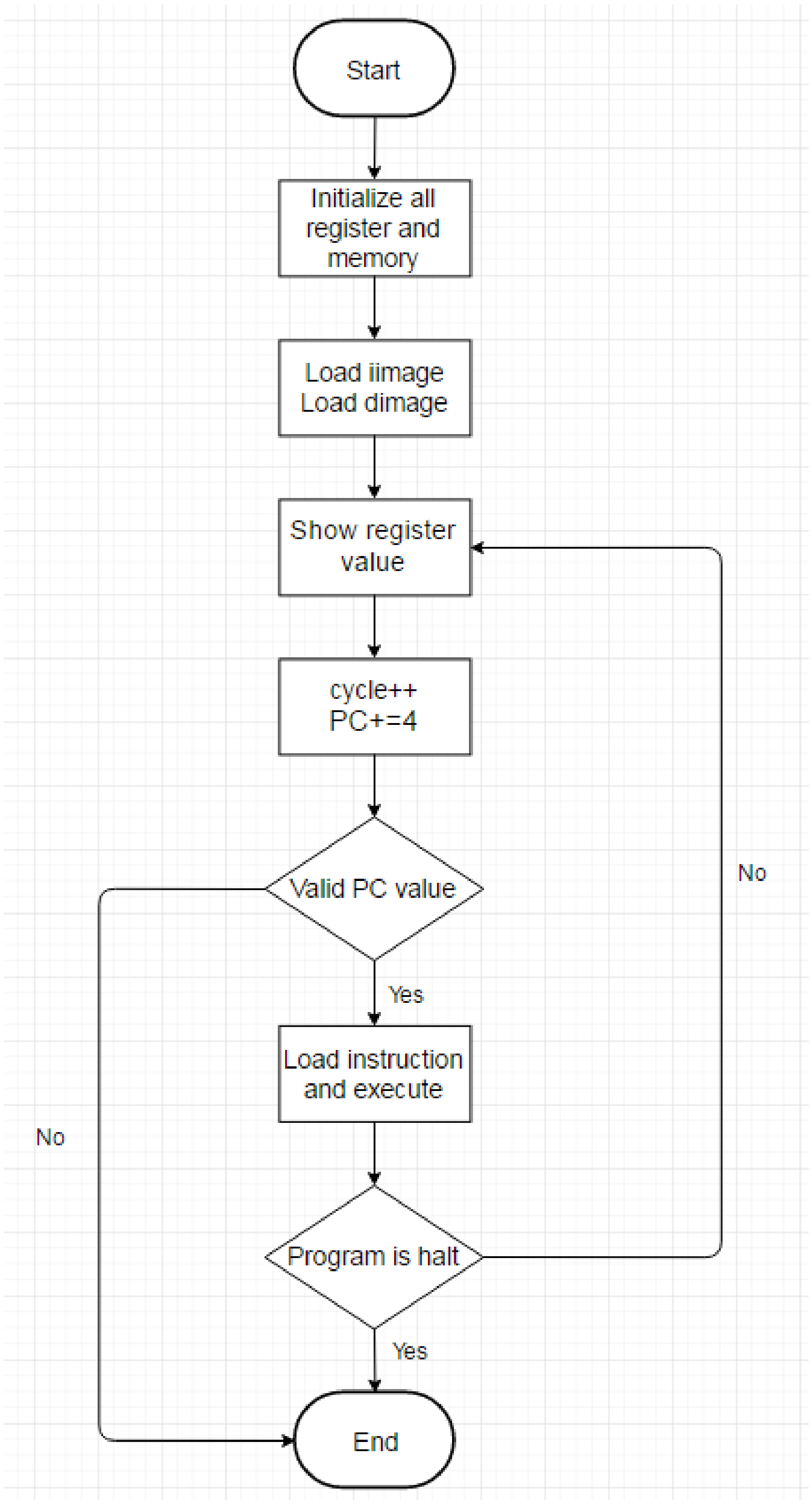
Computer Architecture
Project 1

Due. 2017/3/31

104062203 陳涵宇

Project Description

1. Flow chart



The picture above is the flow chart about my program. At the begging, the program will set all of register arrays and memory arrays to zero to make sure there will be no data in all arrays. Then it will load data in iimage.bin and dimage.bin. After loading data, the program will write all registers' value into snapshot.rpt for cycle 00.

Now, PC will be added by 4 for the first instruction, and cycle will be added by 1, where it is cycle 01 here. Before loading instruction, the program will check if PC is out of range of I memory, where it is set to be 1024 in this project. If PC is valid, then the program will access the corresponding instruction in I memory, and then execute it. Otherwise, the program will show an error message and terminate.

After executing, the program will check whether the program itself is halt or not. If it is halt, then the program won't load new instruction and terminate. If it is not halt, then the program will show the registers' value changed in this cycle and increase PC and cycle by 4 and 1 respectively for the next instruction.

2. Detailed description

In simulator.cpp:

In this file, the program will do the same thing as shown in flow chart. Most of instructions are finished by calling function in other files. For example, when loading iimage.bin and dimage.bin, functions “loadIimage” and “loadDimage” in memory.cpp are called.

A Boolean variable called “isHalt” is used for the step “Program is halt” in the flow chart. If something that will cause the program halt happens during executing instruction, the value of “isHalt” will be set true, and the program will find that it needs to terminate in next cycle by checking “isHalt”.

To record what cycle is it now, an integer variable called “cycle” is used. At the beginning, it will be initialize to 0. Then by executing instruction, “cycle” will increase by 1 to represent the cycle for next instruction.

In memory.cpp:

First, an integer variable “MemorySize” is declared and set to 1024, for the maximum size of memory is set to be 1K bytes in this project. Then, there are another two integer variables “insNum” and “dataNum” are declared to represent the data number will be load from iimage.bin and dimage.bin, respectively.

Two arrays are declared for data storing, and they also represent I memory and D memory respectively. The size of “dataMemory” is set to 1024, for it can load 1024 data at most. However, the size of “insMemory” is only 256, since an instruction needs 32 bits (4 bytes) to represent, the maximum number of instruction can be loaded is at most 256.

There are 7 functions implemented in this file. “initMemory” is for initializing, and it is called at the beginning of program in simulator.cpp.

“readWord” and “readByte” are used to load data in 4 bytes and 1 byte. These two functions are used for loading data from iimage.bin and dimage.bin, and they’re called by the following functions, “loadIimage” and “loadDimage”. As their name shows, they’re used to load data from .bin file. In “loadIimage”, it will first load data for PC, then load data for ”insNum”, and finally started to load instructions and save them into “insMemory”. As for “loadDimage”, the first data is loaded for SP, then load data for “dataNum”, finally all data for “dataMemory”.

The last two functions are “loadMemory” and “saveMemory”. These functions are used to deal with data need to be load from or save to “dataMemory”. Before loading or saving, the function will call functions in error.cpp to check if there is any error occurred in this instruction. If error does occur, “isHalt” will set to be true, so after checking error, the functions need to check “isHalt” to determine whether doing memory access or not.

In regfile.cpp:

I declared an array “reg” to represent the 32 registers and HI, LO and PC. reg[32] represents HI, reg[33] represents LO, and reg[34] represents PC. A set is used to help me record those registers whose value changed in each cycle. There are two functions in this file, “initReg” and “showRegValue”. “initReg” is used for initializing, and it is called in simulator.cpp at the beginning of program. “showRegValue” is used to write the registers which have different value in each cycle into snapshot.rpt. The program will go through the set and print all of data to snapshot.rpt, then the set will be cleared for storing data in next cycle.

In instruction.cpp:

This file is designed to analyze the data read from “insMemory”, so I use switch-case to select which instruction is executed and implement each case according to the description written in Appendix A for each instruction. If it is needed to detect error, the functions of error detection in error.cpp will be called in corresponding case. After detecting, the program will start to execute. Since the only type of error that shouldn’t to any calculation is “Write to register $0”, I will set reg[0] to 0 after each instruction to ensure that no matter reg[0] is overwritten or not, the value of reg[0] can always remain 0.

In error.cpp:

In this file, 5 kinds of error listed in Appendix D will be detected by 5 functions. Each function will detect one kind of error and do something if error occurs.

“WriteToZero_Check” is used to check if register $0 is overwritten. It uses operation code and write register to judge.

“NumberOverflow_Check” checks whether the result of calculation is overflow or not. If the two operands are in the same sign but different from the result, then the error occurs.

“OverwriteHI_LO_Check” is used to detect if the two registers HI and LO are overwritten before accessed to other registers. I use a boolean variable “needAccessed” to record the state of HI and LO. If it is true, it means that HI or LO need to be accessed before next multiplication happens. The argument passed in is also a boolean variable called “doMult”. It represents whether a multiplication is executed or not. If both of boolean variables are true, it means that HI or LO need to be accessed, but a multiplication is executed and thus causes error.

“AddressOverflow_Check” and “Misaligned_Check” are used to detect error when accessing memory, such as instruction lw, lh, sw, sh…etc. Different from other types of errors, when these two types of error occur, the program needs to terminate, so after the functions detect error, they'll not only write error message to

error_dump.rpt but also change "isHalt" to true.

Testcase Description

I just randomly select some instructions and execute them. To test for error, I add some of error that won’t cause the program terminate into the testcase. A misaligned error is added to test if the program will terminate after detecting error.

However, I did not notice that the testcase will be thought as a valid one only if the result between golden simulator and my simulator is same. In first submission, I only test if it can execute in my simulator. After fixing my simulator, the result between golden simulator and my simulator is same now.