DESIGN DOCUMENT

CSL216

Enhanced ARM Simulator

Prepared by

$HimanshuKejriwal - 2016CSJ0011$

$RahulByasSherwan - 2016CSJ0028$

May 4, 2018

Contents

1 Introduction

1.1 Purpose

In this assignment, we are designing an arm simulator. No abstraction is being done from our side, the user will be able to see the contents of the registers and along with that, it will also show things going on in various stages including pipeline registers.

2 Overall Description

2.1 Output Format

CLOCK CYCLE COUNT: It will show the clock cycle going on at that instance say 50.

IF STAGE: Some instruction say ldr r1,[r2,#4] is being fetched at the 50th clock cycle.

IF/ID PIPELINE REGISTER: It will show its content available at that clock cycle.

ID STAGE: it will show the instructions being decoded.

ID/EX PIPELINE REGISTERS: It will also show its content available at that clock cycle.

EX STAGE: It shows the operation going on at that particular clock cycle.

EX/MEM PIPELINE REGISTERS: It will also show its content available at that clock cycle.

MEM STAGE: If instruction like str / ldr will be used then its contentwill be displayed otherwise it will
say "NO USE OF IT IN THIS CLOCK CYCLE".

MEM/WB PIPELINE REGISTERS: just displays contents like above pipeline registers. WB STAGE: it will show the instruction being written on the registers.

HAZARD OCCURRENCE: this part will report various hazards if occurred and print something like following:

DATA HAZARD OCCURRED.
otherwise, if there is no hazard it will print :
NO HAZARD OCCURRED.

REGISTER CONTENTS: After all the above things execution, we will see the final status of all registers at that particular clockcycle.

In this way, a user will see the contents of the simulator.we will also try to implement branch prediction method if possible.

2.2 WORK OF STAGES:

The arm simulator is divided into 5 stages ,namely IF(instruction fetch),ID(instruction decode),EXE(execute stage),MEM(memory stage) , WB(writeback stage).It will have four pipeline registers, namely IF/ID, ID/EX, EX/MEM, MEM/ WB. At each clock, we will show the contents as described in introduction part.

The components are described briefly in the manner an actual simulator should behave, In algorithm section, we will show how actually the simulation is working.
So, the components described are as follows:

Instruction Fetch:
1. PC is incremented and sent to PC register
2. Instruction is fetched from Memory

Instruction Decode:

1. First, it will see the instruction is of RR-type or not.
2. If it is of RR - type then it will decode the data and assign it to registers read and write and forward it to execute stage.
3. If it is a case of immediate operand then the mux will take care of it and direct it to the ALU part where this instruction will be sent directly to execute stage. Mux will take the decision based on the signal control part will send.

Execute Stage:
1. This will first select either read data2 or immediate operand based on the mux.
2. Then it will perform an arithmetic operation according to the opcode which will be evaluated by ALU part.
3. After this, it will forward it to mem stage if it is ldr/str instruction otherwise to the write register. If it a branch instruction then the arithmetic result will be forwarded to PC.

MEM STAGE:
1. This stage will check whether the instruction is to be written in the memory or to be

taken from the memory based on the opcode.
2. After doing step 1. it will forward it to WB stage.

WB STAGE:
1. This stage will determine on which register data is to be written.

NOTE: For sake of simplicity, we have not talked about the pipelined version of it since we are giving an overall view in this section.

Let's consider some hazards which generally occurs due to pipelining.
There are three classes of hazards:

– Structural Hazards:
They arise from resource conflicts when the hardware cannot support all possible co combinations of instructions in simultaneous overlapped execution.

– Data Hazards:
They arise when an instruction depends on the result of a previous instruction in a way that is exposed by the overlapping of instructions in the pipeline.

– Control Hazards:
They arise from the pipelining of branches and other instructions that change the PC

3 Algorithm

3.1 USE OF ASSIGNMENT 4:

We have decided to parse the code and check error with the help of code we have submitted in assignment 4.

IF stage: In assignment 4 we were using a loop to access the instruction stored in vectors. So, that will be our IF STAGE in this assignment.

EXE STAGE: In assignment 4 we were using a member function of arm simulator to do arithmetic calculations. So, that will be our EXECUTE STAGE for this assignment.

CLOCK CYCLE, IPC(avg) AND LATENCY are also taken from assignment 4.

3.2 STRUCTURE OF CODE:

A class with name enhanced_armsimulator will be made.

DATA MEMBERS: 4 X n 2d array/vector having a name pipeline_register (these will do the work of four pipeline registers.)

A 1-D array with 16 element space having a name regs (these will do the work of registers we require.)
A 1-D array of controls.
A 1-D array of labels.
A 1-D array/vector to store instruction.

SOME FLAGS DATA MEMBERS WILL ALSO BE HERE BUT WE NOW DON'T HAVE AN EXACT IDEA THAT'S WHY WE ARE NOT MENTIONING IT HERE.

MEMBER FUNCTION: Note arguments are not being provided for the following member function since it is still ambiguity.

```
void getdata(); // to provide initial values
int INSTRUCTION_SECTION();
int IF_STAGE();
int ID_STAGE();
int EXE_STAGE();
int MEM_STAGE();
int WB_STAGE();
```

Note all the above functions are of int type because it will return a value which will determine which data to be considered for the next stage.

Some output member function :

```
void clock_number();
```

void output_IF(); // it will also diplay contents in if/id pipeline register which is stored in pipeline_register[0][x] (data member defined above in data member section).

void output_ID(); // it will also diplay contents in id/ex pipeline register which is stored in pipeline_register[1][x]. void output_EXE(); // it will also diplay contents in ex/mem pipeline register which is stored in pipeline_register[2][x].

void output_MEM(); // it will also diplay contents in ex/wb pipeline register which is stored in pipeline_register[3][x].

```
void output_WB();
void output(); // this will show the contents of all registers.
```

MAIN(): the main function will do the following things:

1. read the instruction file and store it in a 1d vector.
2. it will find all labels and will pass it to enhanced_armsimulator class through an object of its type.
3. then it will pass instruction vector to the class.
4. when it will return 0; and the program will exit.

We will make separate files for the main section and class section since it is too hectic to rectify errors in a small file and compile it through makefile.

3.3 Hazard Detection:

1) Handling Data Hazard:-

We have planned to let control function return some value according to the need in a particular stage.Then instruction_section function will call a hazard_detection function to check in the register whether the values required are there or not.
If it is still not reached then it returns an integer value which will be like a key for the various output section.
If hazard actually detected then values will not be updated and we will see the previous value as it is along with a message that stall has occurred.

2) Handling Control Hazard:-

If a branch instruction occurred then instruction_section will call Hazard_detection function to check how many stalls will be there.
Then it will return a value which will determine when to change the contents inside the data members and which content need not be printed initially.
If this case will work properly then we will try to resolve control hazards by introducing another function which will predict the behavior of instruction and removes some stalls according to the prediction.

3) Handling Structural Hazard:- This case will be handled by the Insturction_section function itself by rearranging the instructions so that it will not occur.The logic for it still needs to be designed.

4 Sequence of execution

4.1 Execution

1). main() function will read the input file and stores it contained in an array/vector. It also creates a separate array for storing labels.Then it will provide these data to getdata() function defined in the enhance_armsimulator class.

2). getdata will initialize all the data member defined in the class.

3). Then get data will call instruction_section() function .

4). In instruction_section() function we will run a loop until end of the array which will store the instruction.

5). In the loop, we will call all the five-stage to do the processes required.

6). But before calling a particular stage associated function we will first call control to decide which part of instruction to consider. Then we will call Hazard_detectection function to check for the possibility of hazards. The hazard function will return a value which will let us know whether the output or stage associated function needed to be called or not.

7). Then we will call all the output function defined above to show the contents to the user after processing it.

8). when the loop ends we will report the final memory status, average IPC, and Latency.

9). The function will return to main() and the program will end.

Note: we have not talked about the parsing and checking the validation of the instruction since we have already done it before in assignment 4.

5 Test Plan

5.1 testcases

We have made 11 test cases for checking various possibilities of hazards,branches etc which is contained in subfolders of testcases folder.
We have written subfolders's name by name of cases which the code is handling .

For Example : one subfolder has name "hazard_not_removable_test_case". It means that a sample.txt file is there inside this subfolder which is showing that hazard removing after creating a stall (like anyinstruction followed by an ldr instruction in which rt and rs source resistors are same).

5.2 Note

Input file must be named "sample.txt".
Do not open the input file in gedit or vi or vim , since these editors append empty line in the end of any file which will result in an error. Use editors like Sublime to open input files, so that you can know if any emptyline might not have been there as these editors show the line numbers.

6 Experimental Observations

The complexity of the code is big-oh(n) where n is the total number of lines in the input file.

The input file can have million lines also , the code will run . Since, we using vector for storing instructions line by line.

For insturction like cmp,add,sub,mul the code takes more time than instructions like b,ble,bge etc . This is due to the fact that cmp,add types of statements have many hazard possibities(forwarding hazard), so it have more conditional statements in the code.

Practically , str,ldr statements take more time since, these statements need to be accessed from memory. However, this is a simulator code . So, in this code, these are accessed easily as we are using memory as a class data member . We just need to call that to access it.