Assignment 5
Design Document

Thaisnang Reang — Mayank Singh

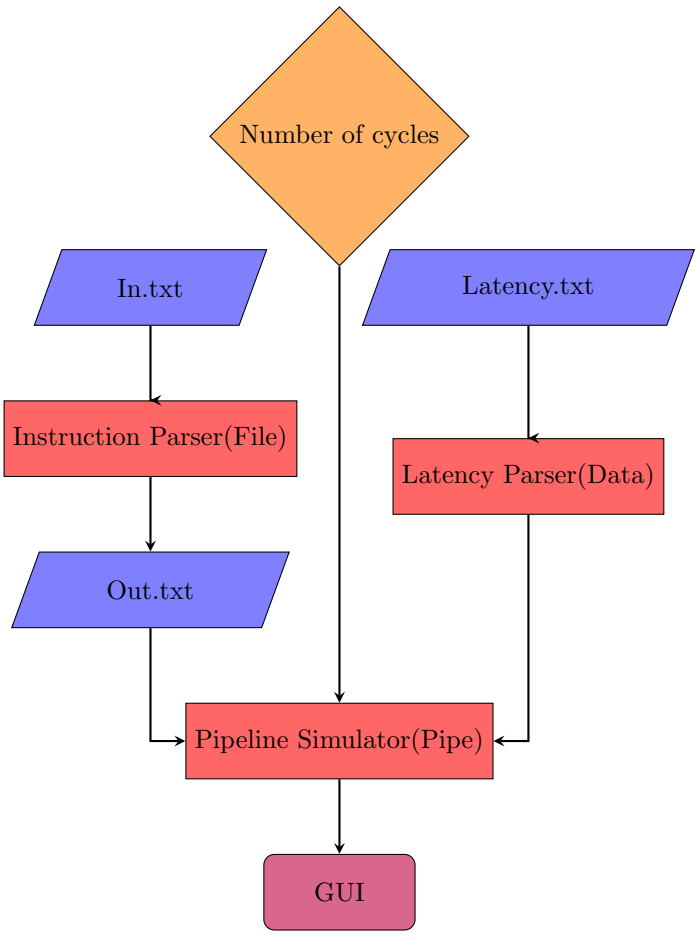
April 30, 2018

Abstract

In this assignment, we have developed an ARM architecture instruction set simulator. All the instructions are performed in a pipeline with five stages and each stage performs the task in the cycles given by the user. We have also created a graphical user interface for viewing the execution of instructions.

1 - Overall Design

The simulator will first ask for an input of the number of cycles it has to simulate,giving an input of "end" will run it till the end. After that the GUI will open and simulate from start to a specific cycle given by the user. We can also increment and decrement the simulation speed and also reverse the execution of instructions by pressing buttons of keyboard. Average instruction per cycle, total no. of cycles will be calculated and shown simultaneously after each cycle. We have split the program into parsing, simulation and GUI as separate components.



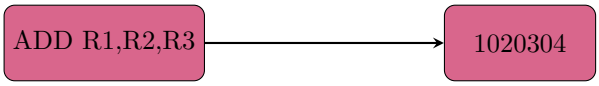
2 - Parser

2.1 File:-

This parser takes input from an "in.txt" file and outputs the encoded strings in an "out.txt" file.It first reads the input for labels and then stores the corresponding instruction number and the branch instruction number, after that it again starts from the first instruction, begins encoding each instruction according to the encoding as follows:-

1- ADD=1
2- SUB=2
3- MUL=3
4- STR=4
5- LDR=5
6- CMP=6
7- MOV=7
8- BNE=8
9- BGE=9
10- BL=10
11- B=11
12- LDR_Pseudo=12
13- Rx=x+1 *(x is the register number)*
14- Offset=Offset + 16

For example:-



Note:- The parser only reads only instructions it does not support comments. The format of instruction needs to be followed strictly.
i.e- Label:—Instruction—space—Destination—comma—Source/Offset—comma—Source/Offset

2.2 Data:-

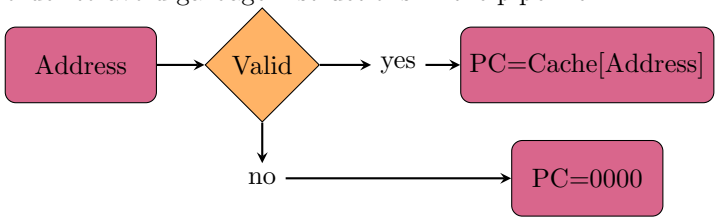
This parser reads the "Latency.txt" file stores the delays in a vector in the designated position.

3 - Simulator:-

The simulator has five functions each for one stage.

3.1 Stage-1:- Instruction Fetch

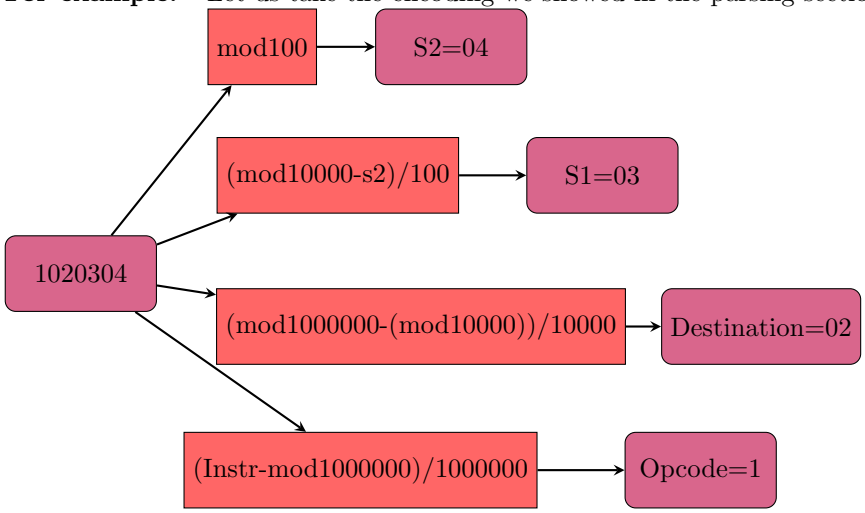
This stage takes an Instruction address and checks if the address is appropriate. If the address is appropriate then it fetches instruction from the instruction memory, which is a vector named cache in our program, and stores it in the PC*(Register 16)* register.If the address is inappropriate then it sets PC to 0 in order to avoid garbage instructions in the pipeline.



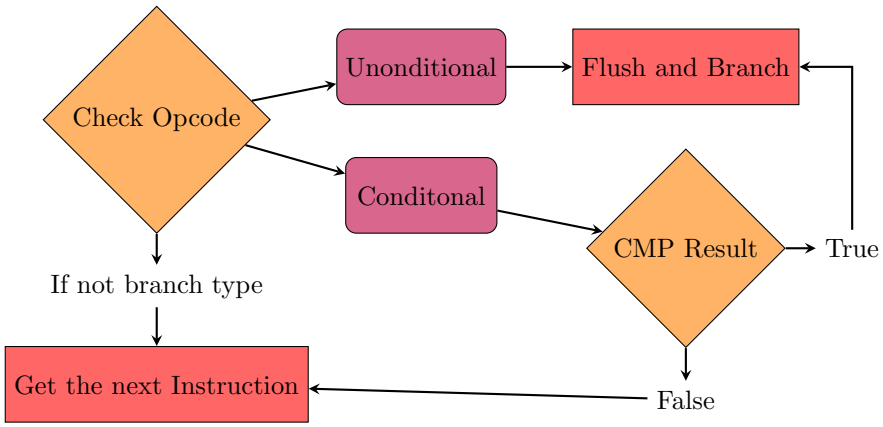
3.2 Stage-2:- Instruction Decode

This function takes the instruction in PC and decodes it into opcode, destination, source1, source2 by doing modulo operation with appropriate multiples of 10 and send the data forward to the next stage.

For example:- Let us take the encoding we showed in the parsing section

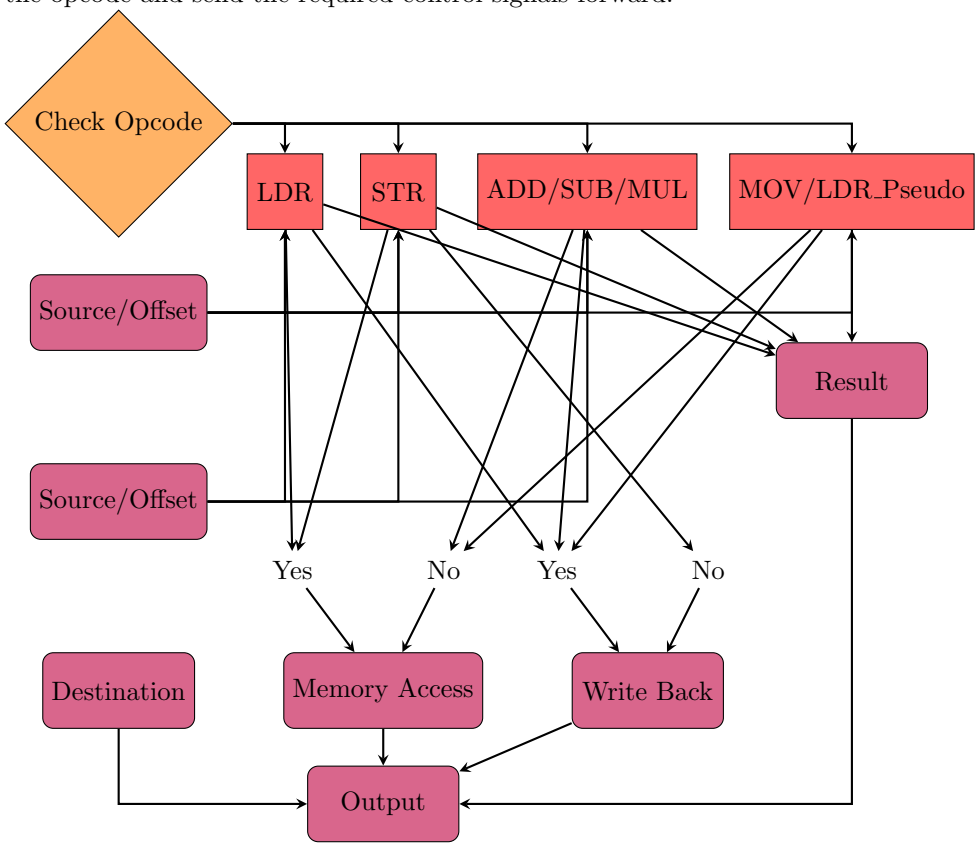


The branching has been added in this stage,if it is an unconditional type then it simply changes the address to appropriate one, if it conditional it checks the result of CMP and then decides whether to take branch or not.In both cases of branch the Instruction from IF stage is flushed if branch was taken.



3.3 Stage-3:- Execution

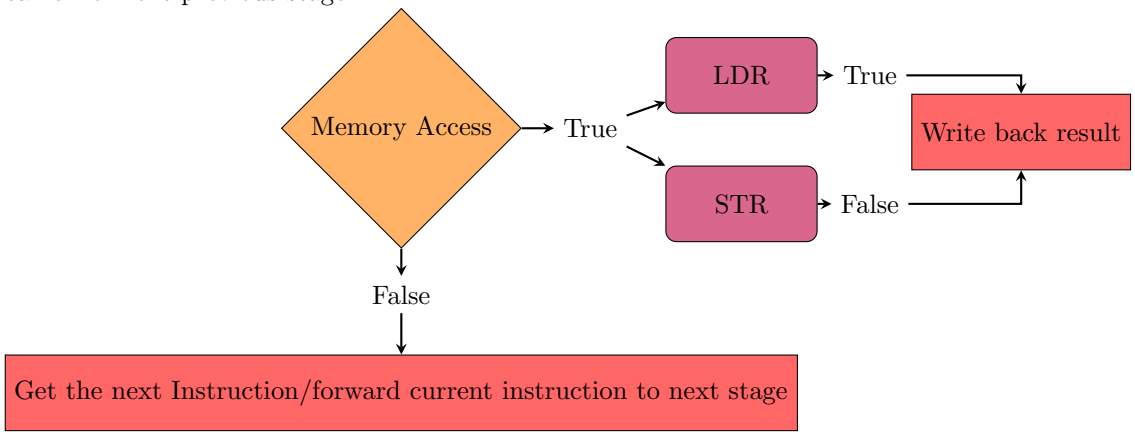
This function takes the decoded data from the previous stage of the pipeline fetches the required data from the registers and do the operation according to the opcode and send the required control signals forward.



We have assumed the calculation of address for STR and LDR takes the same cycle as ADD instruction. The CMP function is also here as well. It does not write_back or do memory_access it simply just updates a global variable each time comparison is done.

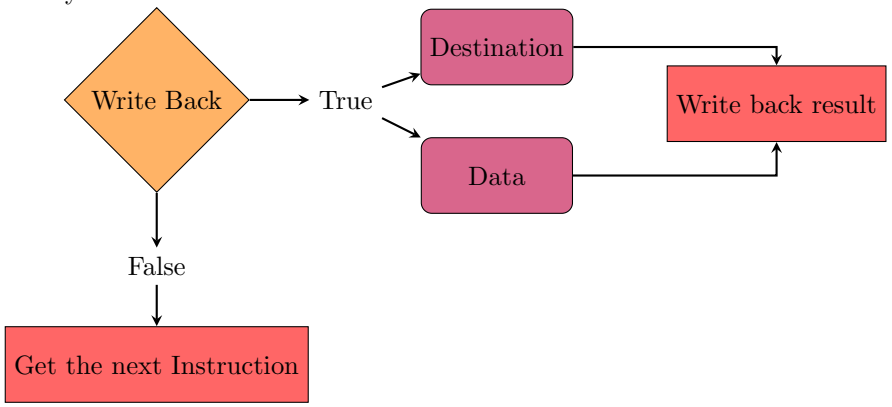
3.4 Stage-4:- Memory

This function either reads from memory or write to memory or neither and just forwards the instruction to next stage, depending on the control signals which came from the previous stage.



3.5 Stage-5:- Write Back

This will write back the data, which came from the previous stage, to the designated register if write back signal is true.We have assumed this takes only one cycle.



4 Hazard Detection

4.1 Data Hazard:-

4.1.1 Stage-3 Hazard

If either of the sources in the Stage-3 match with the destination of Stage-4 then it is an hazard. We will be checking this in the Stage-3 of the pipeline.

4.1.2 Stage-4 Hazard

If either of the sources in the Stage-3 match with the destination of Stage-5 then it is an hazard. We will be checking this in the Stage-3 of the pipeline.

4.2 Control Hazard:-

As we know control hazard usually occurs because of branch instruction.So that is why we are checking branch instruction to be taken or not taken in second stage and if branch is not taken then the instruction in IF stage is flushed out.

4.3 Stalling:-

Stalling is performed according to the instruction latency which we get from the latency.txt file.
For example- If LDR instruction has latency 20 then the MEM stage of LDR will take 20 cycles to complete and upto then all the instruction after it will get stalled and for arithmetic type instructions stalling is performed in ALU stage.

5 - Graphic User Interface

For making GUI we have used OpenGL library.
The GUI will show 5 stages pipeline processor in which instructions will be shown according the clock cycle. In case of no instruction it will show NOP in that stage. when there is a stalling then instruction after the stalling stage shows NOP and the instruction in which stalling occurs remains in that instruction until stalling completes.
For each stage text coordinates are fixed and the text changes according to the instructions we get.
At the bottom of the window on left side it shows Clock cycles and on the right side it shows Instruction per count which gets updated in every cycle.
1.Keyboard Input- It detects keyboard input from the user and acts accordingly.
1.1: Press “M”- To increase the speed by 2 times.
1.2: Press “N” - To decrease the speed by 2 times
1.3: Press “P” - To pause and resume the pipeline.

1.4: Press “R”- for reversing the execution of instructions.
1.5: Press “ESC”- for exiting.