Design Document

ARM Assembly Progam Simulator

CSL216: Computer Architecture

Vikas Gola, 2016CSJ0023

1. Overall Design

The Design of pipeline is implemented using the different fuctions for different use or functions for different class of instructions.

2. Pipeline Implementation

I used five different functions to implement the main pipeline each for every stage of pipeline whose work is specified to there name. In each point, I first described the work of stage in pipeline and then the work of function for which it's implemented.

The work of each function and the stage is defined as follows:

- **Instruction Fetch:-** This stage fetch the information from the instruction memory and return that instruction. The Implemented function for this does the same which takes the progam counter(pc) and returns the instruction at that pc as a string.
- **Instruction Decode:-** The work of this stage is to decode or analyse the instruction received from IF stage like which registers have been used in this stage and are they vaild or not. The implemented function take the instruction as string from IF and find which registers or memory have been used in this and then send the details to next stage.

- **ALU(Arithmetic Logic Unit):-** This stage does the all calculating work like addition, substraction, multiplication and division. This stage also take the decisions e.g. which work should be done or not. The implemented function for this takes the registers or memory with which it has to work and does the all calculation.
- **Data Memory:-** This stage is where all memory work and handling has done. The function for this stage return the data at any address(for the LDR instructions) or write the data at any given address(for the STR instruction). This function only does work in the instruction related to memory.
- **Write Back:-** This stage is where we write the calculated value from Arithmetic Logic Unit(ALU) to the destination register in the case of instructions which work on Arithmetic Logic Unit which are mainly ADD, SUB, MOV, MUL.

Now, it's come to the way to connect the all stages to make pipeline work. I will connect these functions of stages of pipeline in reverse order as follows...

Write Back -> Data Memory -> ALU -> Instruction Decode -> Instruction fetch

First the 'WriteBack' function then from it 'DataMemory' then from it 'ALU' and so on. The reason behind this is because, if we will do it in a linear way like first Instruction fetch and then others, then we will lost the data from mid registers values after each stage as mid registers values have to be written by instruction which is on that stage. However, there are some hazards in the pipeline which has to be removed before running. So, lets go to next section where we will talk about these hazards and will remove these hazards.

3. Hazards Free

- **Structural Hazards:-** This hazard rise when there is simultaneously calling of stage for their work from different instructions. In my implementation, this is never going to happen as functions can only been called one at a time.
- **Data Hazards:-** This hazard happen when current instruction want to write to any memory or register, but that instruction is currently not reached that stage where it can write the data, and simulatenously the next instruction want to work with that data of memory or register. So, it can produce two hazards of data one with registers and the other with memories when the instruction want to do its work with data but the last instruction has not written its work in the data(or not its work completly).

  Examples of Data hazards:-

  One with registers:

  Instruction1 : SUB R5, R4,R3

  Instruction2 : AND R2,R2,R5

  Other with Memory:

  Instruction1 : LOAD R5,[R2]

Instruction2 : ADD R5,R5,#5

- **Control Hazards:-** Control hazard happen because of the branch instructions which changes the 'pc' suddenely. It leads to unreasonable fetching and decoding of next instructions after the branch instruction if that has not be run(if it has to go to branch that has been called). My code solve this issue by using a 2 predictor method which used two bits on the bases of which flag will raise which tells us next instruction has to be load or not.

4. Test Cases

- **Test case for Data hazard:-**

```
.text
LDR R1 , =NAME
ADD R1 , R1 , #50
SUB R2 , R1 , #1
STR R2 , [R1 , #2]
MOV R3 , #2
MOV R4 , #3
MUL R5 , R3 , R4
.data
NAME: .space 10
```

- **Test Case for Control hazard:-**

```
.text
MOV R1 , #100
MOV R2 , #0
LOOP:
    ADD R2 , R2 , #1
    CMP R2 , R1
    BNE LOOP
MUL R1 , R1 , R2
```

These test cases have hazards and my program should run correctly on these