CSL216 : Assignment 5 - ARM Assembly Simulator with Multicycle operations and Pipelining
Design Document and Test Plan

Aakar Sharma (2016CSJ0066) & Sahil Bansal(2016CSJ0008)

April 10, 2018

1 Overall Design :

An object-oriented design is followed while implementing all the features of the pipeline. Each hardware resource used in the pipeline is simulated by a corresponding function for the same. The pipeline registers are implemented as a structure datatype for each stage. The basic strategies to handle and detect the hazards are planned to be implemented and more advanced techniques can be implemented later. Stalling of the pipeline is simulated by simply incrementing the clock cycle count. It is also planned to implement instruction and cache memories.

2 Simulation of the Pipeline :

- An exact simulation of the ARM's pipeline is being performed by the program. For handling each stage of the pipeline, a function has been made which fetches the data from the previous stage's pipeline register and updates the data in the next stage's pipeline register.
- So, the following functions have been made for the 5 stages of the pipeline:
  1. **InstructionFetch**
  2. **InstructionDecode**
  3. **Execute**
  4. **Memory**
  5. **WriteBack**
- The **IF**, **ID** and **WB** stages each take one clock cycle whereas the stages **ALU** and **MEM** take clock cycles depending on the instruction and its latency. The pipeline would be stalled if any of these stages takes more than one clock cycle.

- During the simulation, in a particular clock cycle, these functions are called in the reverse order so that the value in pipeline register to be used later is not lost.
- **Memory-Reference Instructions:**
  - Load instruction goes through the entire 5 stages of the pipeline.
  - Store instruction is active only in the first 4 stages of the pipeline, since there is no write back needed.
- **Arithmetic-logical instructions:**
  - Active in the first 3 stages of the pipeline.
  - Also active in the 5th stage, i.e. WriteBack.
- **Branch Instructions:**
  - These instructions are only active in the first 3 stages of the pipeline.
  - They also update the PC when the branch is taken.

3 Pipeline Datapath and Control:

3.1 Pipeline Registers

The values stored in each of the pipeline register is mentioned below.

1. **IF_ID**
   - The value of $PC + 4$.
   - The index representing the instruction.
2. **ID_EX**
   - The value of $PC + 4$.
   - 2 integers for the data in the registers $rn, operand2$, the values on which the ALU operates.
   - rd - the destination register.
   - A boolean value indicating whether the instruction has **immediate operand**.
   - A string representing the type of instruction useful for the ALU to operate.
3. **EX_MEM**
   - The value of $PC + 4$ or the branch label's address.
   - rd - the destination register.

- The result of the ALU (which acts as the memory address).
- The data in the register *rn* to be written in the memory if *str* instruction.
- A string *op* representing whether *ldr* instruction or *str* instruction.

4. **MEM_WB**
   - Either the data read from the memory or the result of the ALU passed directly depending on the value of *op* from the previous register.
   - *rd* - the destination register.

3.2 Control Signals

Since the string *op* is stored in each of the pipeline register and conditionals are used for implementing multiplexers, there is sufficient information so that control signals need not be explicitly stored.

4 Graphically representing the Pipeline[if time permits] :

- A **single-clock-cycle pipeline diagram** is planned to be generated as SVG.
- Since the basic template of the diagram is the same, i.e. there are five stages to be shown, they are generated only once in the SVG whereas the instructions in a particular stage is shown by a text above the stage, which changes at every clock cycle.
- It is also planned that an interactive SVG can be made which shows the stages of the pipeline at any clock cycle (can be changed instantly through interaction).

5 Handling the hazards:

5.1 Data Hazards

- The hazard would be detected in the **ID** phase of the pipeline. The previous 2 instructions would be checked for a hazard.
- If an hazard is detected, there can be two possibilities:
  - If it is detected from the last instruction and it is a memory(ldr/str) instruction, then the pipeline is **stalled** for the next clock cycle.
  - Otherwise, it would be handled by **forwarding**. The result of the **execute** stage in the **EX_MEM** pipeline register is passed to the current instruction's **ID_EX** pipeline register instead of reading from register file.

5.2 Control Hazards

- For handling the branch instructions, initially **static** branch prediction will be used assuming that backward branches are taken whereas forward branches are not taken. In case of a wrong prediction, pipeline will be stalled.
- When this works well, the **dynamic** branch prediction technique of storing the history of last two predictions will be used.
- Other modern techniques of branch prediction will be switched to if the previous one works well.

6 Test Plan:

6.1 test1.s :

```
1 mov r1, #5
2 mov r2, #11
3 add r0, r1, r2
4 add r3, r0, r1
5 sub r0, r3, r2
```

- Two continuous **data hazards** in the lines $(3, 4)$ due to $r0$ and $(4, 5)$ due to $r3$.

6.2 test2.s :

```
1 mov r0, #50
2 ldr r3, =AA
3 add r4, r0, #10
4 str r0, [r3]
5 AA: .space 4
```

- It has **load-use data hazard** with forwarding since value of $r3$ updated in line 2 is used in line 4.
- Data in $r3$ must be forwarded from **MEM_WB** to **ID_EX** pipeline register.

6.3 test3.s :

```
1 mov r0, #50
2 ldr r3, =AA
```

```
3 str r0, [r3]
4 AA: .space 4
```

- It has **load-use data hazard** and a stall of one cycle must be inserted since value of $r3$ would not be available until the MEM stage of the pipeline.

6.4 test4.s :

```
1 mov r1, #10
2 mov r0, #0
3 loop:
4 add r0, r0, #1
5 cmp r0, r1
6 bne loop
7 add r3, r0, r1
8 add r4, r3, r0
```

- It is used to check whether **control hazard** is handled properly as it has a **branch** instruction.

6.5 test5.s :

```
1  mov r2, #1
2  mov r1, #10
3  mov r0, #0
4  loop:
5      add r0 ,r0 ,#1
6      cmp r2, #0
7      bne notzero
8      add r2, r2, #2
9      notzero:
10     sub r2, r2, #1
11     cmp r0, r1
12 bne loop
13 add r3, r1, r2
14 add r4, r3, r0
```

- It has two branches. The outer branch is a loop which runs 10 times.
- The inner branch is taken in alternate turns and not taken in the other half of the time.
- Also, a data hazard exists in the inner branch when the branch is not taken.

6.6 test6.s :

```
1 mov r0 , #1
2 mov r1 , #2
3 add r2 , r0 , r1
4 mul r2 , r0 , r1
5 mul r3 , r2 , r1
6 add r4 , r3 , r2
```

- It has **mul** instructions which has high latency and hence would require to stall the pipeline.
- Data Hazard exists in line 4 and line 5 due to $r2$.
- Another data hazard in line 5 and line 6 due to $r3$ but different instructions **mul** and **add**.