ASSIGNMENT 3

Sachin Kumar(220120019), Amogh R (220010005)

1 Introduction

In ths assignment, we have created a working simulator for ToyRISC. The simulator simulates a single-cycle processor which works on the ToyRISC ISA. The five stages implemented in the simulator are:

1. Instruction Fetch Unit
2. Operand Fetch Unit
3. Execute Unit
4. Memory Access Unit
5. Register Write Unit

An additional Control Unit has been created which provides the control signals for every instruction the Operand Fetch Unit receives.

2 Working of the processor

The simulator is programmed to be as close as possible to an actual single cycle processor. The `ControlUnit` is a separate object we have created, which generates a `ControlSignals` object for every instruction. The `ControlSignals` object contains a boolean array for each operation signal and miscellaneous signals.

The program works as follows:

1. The `main()` function calls the function `simulate()`, which loads the object file into memory using the `loadProgram()` function.
2. The Simulator begins the simulation by looping through the five stages of the processor. Initially, a flag called `isIdle` is set to true for the processor object, which says that the processor has not started execution of any program. One the Instruction Fetch unit fetches an instruction, it sets the processors state to 'not idle'.
3. The program continues execution with each instruction passing through their corresponding latches, per cycle.
4. As the number of cycles is synced to the number of instructions executed, there is no difference between their magnitude.

5. The `end` instruction when read, sets the processor back to its idle state. The simulator then ends the loop and writes the statistics to a file.

3 Testing

The simulator was tested on the following files from Assignment 1:

1. `descending.asm`

   Number of instructions: 288

   Number of cycles: 288

2. `fibonacci.asm`

   Number of instructions: 114

   Number of cycles: 114

3. `even_odd.asm`

   Number of instructions: 222

   Number of cycles: 222

4. `prime.asm`

   Number of instructions: 11

   Number of cycles: 11

5. `palindrome.asm`

   Number of instructions: 31

   Number of cycles: 31

The simulator was tested on the same files provided as supporting files:

1. `descending.asm`

   Number of instructions: 277

   Number of cycles: 277

2. `fibonacci.asm`

   Number of instructions: 78

   Number of cycles: 78

3. `even_odd.asm`

   Number of instructions: 6

   Number of cycles: 6

4. `prime.asm`

   Number of instructions: 29

   Number of cycles: 29

5. `palindrome.asm`

   Number of instructions: 49

   Number of cycles: 49

4 Conclusion

The number of dynamic instructions executed here is equal to the number of cycles as no pipelining has been implemented. Also, the high numbers of dynamic instructions is partly because of unoptimized register read-writes from the programmer's side.