ECEC 355
Project 3 Report and Summary
Justin Ngo
3/10/2023

I. Introduction

RISC-V Pipelining Implementation

II. Progress Summary and Results

1. Implementation

- In this project, I divided up the single-cycle RISC-V into 5 cycles (IF, IF, EX, MEM, WB) and arrange those cycles in a pipeline accordingly. Here is the implementation of pipelining:

```
bool tickFunc(Core *core)
{
    int fetch_instr = 0;
    int decode_instr = -1;
    int exec_instr = -2;
    int mem_instr = -3;
    int wb_instr = -4;
    int num_instr = core->instr_mem->last->addr / 4 + 1;
    int starting_stages;
    int i;
    // create a new pipe
    core->pipe = malloc(num_instr * sizeof(PipeInstr));

    if (num_instr > 5) {
        starting_stages = 4;
    } else {
        starting_stages = num_instr - 1;
    }

    // Starting stages
    for (i=0; i<starting_stages; i++) {
        core->pipe[fetch_instr] = malloc(sizeof(PipeInstr));
        core->pipe[fetch_instr]->dec = malloc(sizeof(Decode));
        core->pipe[fetch_instr]->ex = malloc(sizeof(Exec));
        fetch(core, core->pipe[fetch_instr]);

        if (decode_instr >= 0)
            decode(core, core->pipe[decode_instr]);

        if (exec_instr >= 0)
            execute(core, core->pipe[exec_instr]);

        if (mem_instr >= 0)
            memAccess(core, core->pipe[mem_instr]);

        fetch_instr++;
        decode_instr++;
        exec_instr++;
        mem_instr++;
        wb_instr++;
        ++core->clk;
    }
```

```
    // Steady stages
    while (wb_instr < num_instr) {
        if (fetch_instr < num_instr) {
            core->pipe[fetch_instr] = malloc(sizeof(PipeInstr));
            core->pipe[fetch_instr]->dec = malloc(sizeof(Decode));
            core->pipe[fetch_instr]->ex = malloc(sizeof(Exec));
            fetch(core, core->pipe[fetch_instr]);
        }

        if (decode_instr < num_instr)
            decode(core, core->pipe[decode_instr]);

        if ((exec_instr < num_instr) && (exec_instr >= 0))
            execute(core, core->pipe[exec_instr]);

        if ((mem_instr < num_instr) && (mem_instr >= 0))
            memAccess(core, core->pipe[mem_instr]);

        if (wb_instr >= 0)
            writeBack(core, core->pipe[wb_instr]);

        fetch_instr++;
        decode_instr++;
        exec_instr++;
        mem_instr++;
        wb_instr++;
        ++core->clk;
    }

    // Are we reaching the final instruction?
    if (core->PC > core->instr_mem->last->addr)
    {
        return false;
    }

    // free memory
    for (i=0; i<num_instr; i++) {
        free(core->pipe[i]->dec);
        free(core->pipe[i]->ex);
        free(core->pipe[i]);
    }
    return true;
```

2. Results

Using the pipelined RISC-V on the following program:

```
ld x10, 40(x1)
sub x11, x2, x3
add x12, x3, x4
ld x13, 48(x1)
add x14, x5, x6
```

- x1 = 0; x2 = 10; x3 = -15; x4 = 20; x5 = 30; x6 = -35
- 40(x1) = -63, 48(x1) = 63

I got the following results (in 9 cycles):

```
[jn832@xunil-03 project_2_3_4_5]$ ./RVSim ../cpu_traces/project_four
Loading trace file: ../cpu_traces/project_four
ld x10, 40(x1)

sub x11, x2, x3

add x12, x3, x4

ld x13, 48(x1)

add x14, x5, x6


Instruction at PC: 0
00000010100000001011010100000011

Instruction at PC: 4
01000000001100010000010110110011

Instruction at PC: 8
00000000010000011000011000110011

Instruction at PC: 12
00000011000000001011011010000011

Instruction at PC: 16
00000000001100010100001110011001

*--------------------------------------------*

Original register values (only values != 0):
x[2]: 10
x[3]: -15
x[4]: 20
x[5]: 30
x[6]: -35
```

```
Original memory bytes (only values != 0):
Mem[40]: 11000001
Mem[41]: 11111111
Mem[42]: 11111111
Mem[43]: 11111111
Mem[44]: 11111111
Mem[45]: 11111111
Mem[46]: 11111111
Mem[47]: 11111111
Mem[48]: 00111111

*--------------------------------------------*

Number of clock cycles: 9

*--------------------------------------------*

Final register values (only values != 0):
x[2]: 10
x[3]: -15
x[4]: 20
x[5]: 30
x[6]: -35
x[10]: -63
x[11]: 25
x[12]: 5
x[13]: 63
x[14]: -5

Final memory bytes (only values != 0):
Mem[40]: 11000001
Mem[41]: 11111111
Mem[42]: 11111111
Mem[43]: 11111111
Mem[44]: 11111111
Mem[45]: 11111111
Mem[46]: 11111111
Mem[47]: 11111111
Mem[48]: 00111111

Simulation is finished.
[jn832@xunil-03 project_2_3_4_5]$
```