UIC – ECE 366: Computer Organization - Fall 2018

Project 3: ISA Simulation

For this project, your ISA should be completed with a python simulator to execute the 2 programs. Again, your ISA should feature 8-bit instructions (including 1 parity bit) and 16-bit data (registers and memory), which you will optimize for two programs (ME - "Modular Exponentiation", and BMC - "Best Match Count").

- Extra credit: multi-cycle CPU implementation of your ISA (supported by the simulator for correct DIC counting, and HW schematics including control unit FSM. Consider breaking the process of instruction according to the IF / ID-RR / ALU / MEM / WB stages.

Submission components:

Include the following 7 files in your Bb submission:

- **`p3_group_x_report.pdf`: a self-contain PDF report writeup**
- `p3_group_x_sim.py`: Python simulator for your ISA
- `p3_group_x_dmem_A.txt`: result data memory pattern A
- `p3_group_x_dmem_B.txt`: result data memory pattern B
- `p3_group_x_dmem_C.txt`: result data memory pattern C
- `p3_group_x_dmem_D.txt`: result data memory pattern D
- `p3_group_x_p1_imem.txt`: machine code for P1 (EM)
- `p3_group_x_p2_imem.txt`: machine code for P2 (BMC)
- `(optional) p3_group_x_p0_imem.txt`: machine code for P0 (your choice)

PDF Report writeup components:

Part A) ISA intro – a recap of your ISA design

1. **Introduction**. Name of the architecture, overall philosophy, specific goals strived for and achieved.
2. **Instruction list**. Give all the instructions, their formats, opcodes, and an example.
3. **Register design.** List of the supported registers and any special designs.
4. **Control flow (branches).** Design of conditional and unconditional branches: target addresses calculation, maximum branch distance supported, and other unique designs of your ISA's branch instructions.
5. **Data memory addressing modes.** Mechanisms of your load / store instructions and their corresponding machine code.

Part B) Answers to questions

1. What are the most significant advantages of your ISA (with regard to the two programs, hardware implementation, ease of programming, etc)? What are the main limitations? What are the main compromises that you have done to make things work, rather than perfecting everything?
2. What have you done towards the goals of low DIC and HW simplification? What could have been done differently to better optimize for each of the two goals, if to start over?
3. Reflect on this project(1-3) experience:
    a. What did you learn from this project? What was the best / worst thing about it?
    b. What advice would you give to someone taking this project in a future semester?
    c. How would you describe the value of this project experience in a job interview?

Part C) Simulation results

1. Execution results: data mem[0] – data mem[5], and DIC for each case
    a. for the two given sample data memory pattern version A and B (see Appendix)
    b. two other data sets C and D (of your choice, to present some meaningful features)
2. **Execution process of the two target programs:** show the (important, illustrative parts of the) python screen output during the process of running P1 and P2, based on various data memory patterns (A, B, C, D), so that it is evident:
    a. every instruction is illustrated for its correct simulation.
    b. every program on each pattern is achieving the correct results.
3. If your simulation cannot correctly achieve both P1 and P2, include a simpler program P0 to illustrate a successful simulation process / outcome.

Part D) ISA package

1. Algorithms (in assembly code) of the two programs (and P0 if needed). Make sure your assembly format is either obvious or well described, and that the code is well commented.
2. (Instr mem) Machine Code for P1 and P2 (and P0 if needed).
3. (Data mem) Pattern C and D of your own design to showcase your programs.
4. Python simulator code.
5. HW schematics
    a. ALU schematic. A hierarchical sketch of your Arithmetic Logic Unit which implements whatever computation that your ISA instructions use (See textbook ch 5.2.4).
    b. CPU Datapath design. A schematic including your register file, ALU, PC logic, and memory components (see textbook ch 7.3.1).
    c. Control logic design. Decoder truth-table (or FSM for multi-cycle implementation) indicating how each control signal is generated from an instruction (see textbook ch 7.3.2 and 7.4.2).

Appendix:

[**Program 1: ME - "Modular Exponentiation"**]

Compute the result (R) for: $R = 6^P \% Q$.

Input: the 16-bit positive numbers P and Q are initially stored in data memory:

P = Mem[0]
Q = Mem[1]

Output: the 16-bit answer *R* should be written back into memory address 2 by your program:

Mem[2] = R

Levels of completion:

1) (80% grade) Assume a fixed Q = 17.
2) (100% grade) Support general 16-bit P and Q.

[**Program 2: BMC - "Best Match Count"**]

Find out the Best Match score (S = [0, 16] where 16 indicates total match) and count (C: number of best matches) from an array (Pattern_Array[]) of one hundred 16-bit patterns, to a given target pattern (T).

Input:

The 16-bit pattern T is initially stored in data memory address 3:

T = Mem[3]

The array begins at data memory address 8:

Pattern_Array[1] = Mem[8]
Pattern_Array[2] = Mem[9]
Pattern_Array[3] = Mem[10]
…
Pattern_Array[100] = Mem[107]

Output: the 16-bit answers S and C should be written back into memory by your program:

Mem[4] = S
Mem[5] = C

Levels of completion:

1) (80% grade) Only need to support "total matching": S=0 and C=0 if no total matching is found.
2) (100% grade) Support in general any best-matching score

d_mem content patterns (before executing program P1 and P2):

Pattern A:

- **P = 9,**
- **Q=17,**
- **R = _____**

- **T = 0000 0000 0000 0000,**
- **S = _____**
- **C=_____**

- **Array[1] = 1111 0000 1000 0001**
- **Array[i+1] = Array[i] + 1**

```
0000000000001001
0000000000010001
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
1111000010000001
1111000010000010
1111000010000011
1111000010000100
1111000010000101
1111000010000110
1111000010000111
1111000010001000
1111000010001001
1111000010001010
...
```

```
0000000100001011
0001000000000011
0000000000000000
0101010101010101
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000010
0000000000000011
0000000000000101
0000000000000111
0000000000001011
0000000000001101
0000000000010001
0000000000010011
0000000000010111
...
```

Pattern B:

- **P = 267,**
- **Q = 4099,**
- **R = _____**

- **T = 0101 0101 0101 0101,**
- **S = _____**
- **C=_____**

- **Array[1] - [100] are the first 100 prime numbers (2, 3, 5, 7, 11, 13, …)**