UIC – ECE 366: Computer Organization - Spring 2020

Project 3: ISA Design for Perceptron

For this project, you will work in groups to design your own *ISA (Instruction Set Architecture)*, a processor with 8-bit instructions and 16-bit data (registers and memory), which you will optimize for the Perceptron program you have worked on in previous projects, and showcase your result with a Python simulator.

[**ISA Design**]

Your ISA design should have:

- a unique name
- a set of supported instructions, each can be encoded non-ambiguously in 8 bits
- some 16-bit registers
- data memory (one address per half-word / 16-bit data)
- PC and instruction memory (one address per instruction in a byte)

General specifications:

- Your processor can have a 16-bit ALU, supporting all the arithmetic / logic operations that your instructions need. When implemented, this will be a single-cycle CPU, so each instruction needs to be carried out by ALU with combinational circuit implementation. If you want to have some special, "powerful" instruction, you need to show its implementation in ALU as combinational logic.
- You can have as many general registers as you want, but all the registers have to be 16-bit in size, which matches the width of ALU.
- You can assume all the registers and data memory bytes are initialized to be 0.
- Instructions and data are stored in a separate memory blocks (both begin at address 0):
- PC is assumed to be initialized at 0, and instruction memory should be as large as you need to hold your program. PC can be more than 8 bits if your program is longer than 256 instructions).
- Data memory has the following access limit: any instruction can either load or store (but not both), and the bandwidth is exactly 2 byte (16-bits). The data memory should support at least 1K entries (each is 16-bit). In other words, it should contain a minimum of 0x400, or 2^{10} 16-bit data entries, to fit all the data including final a, b, and arrays (up to 256 in length) x, y, C.

Optimization goals:

1. High Speed in SW: (i.e., minimizing dynamic instruction count for the Perceptron program).
2. Low Cost in HW: (i.e., making your CPU's hardware design simplified).

[Hardware implementation]

You should provide the detailed hardware implementation for the ISA design, including:

1. **CPU Datapath design.** A schematic including register file, ALU, PC logic, and memory components (see textbook ch 7.3.1).
2. **Control logic design.** Truth-Table indicating how each control signal is generated for each instruction (see textbook ch 7.3.2).
3. **ALU schematic.** A hierarchical sketch of your Arithmetic Logic Unit which implements whatever computation operations that your ISA instructions use (See textbook ch 5.2.4).

[Perceptron Program]

Your program should begin with some code that initializes the value of (`n, A, B),` run the perceptron algorithm, and finish with storing in memory the results of `a, b`, array `x, y, C.`

Input: you should be able to initialize `n` to any 8-bit binary, and `A, B` to any 16-bit binary.

Output: all the results should be written into data memory:

```
a, b at M[0], M[1]

array x starting at M[256]
array y starting at M[512]
array C starting at M[768]
```

[**Python simulator**]

Your ISA should be completed with a python simulator to execute your perceptron program (and, of course, any program that is written according to your ISA instructions).

- Input: a machine code txt file that your python code can "run"
- Output: on screen and to a file, important information of execution the machine code, including but not limited to:
    - Dynamic Instruction Count
    - PC and Register content
    - Data memory content
    - Instruction debugging information

PDF Report components:

Activity Log (10pts)

I) Table of group activity
II) Individual activity list
III) Group work reflection

a) How does this group differ from your previous one?
b) List out one new or important thing that you have learned from this group.
c) If you were to do this project entirely by yourself, how would it be different?

Project Content (90pts)

Part A) ISA intro, Q&A (10pts)

1. **Introduction & Instruction list**. Name of the architecture, overall philosophy, specific goals strived for and approaches used, all the instructions, their formats, opcodes, with example. What are the most significant features of your ISA (with regard to the Perceptron program, hardware implementation, ease of programming, etc)? What have you done towards the goals of low instruction count and HW simplification? What are the main limitations? What are the main compromises that you have done to make things work, rather than perfecting everything?
2. **Register design.** How many registers are supported? Is there anything special about the registers?
3. **Branch design.** What types of branches are supported? How are the target addresses calculated? What is the maximum branch distance supported?
4. **Data memory addressing modes.** What kind of instructions are used to access data memory? What is the range of addresses that can be accessed with your design?
5. **Results for Perceptron.** What are the Dynamic Instruction Count for the two cases V1 and V2 listed? What input combination would give the worst case Dynamic Instruction Count? Why? What would that be?

Part B) Hardware sketches (20pts)

1. **CPU Datapath design.** A schematic including your registers, ALU, PC logic, and memory components. Clearly mark out all the signal lines, their names, and how many bits for each.

2. **Control logic design.** Decoder truth-table indicating how each control signal (one per column) is specified (0, 1, or X) from each instruction (one per row). If you have special instructions or register design, explain the control signals briefly.

3. **ALU schematic.** A hierarchical sketch of your Arithmetic Logic Unit which implements whatever computation that your ISA instructions use.

Part C) Software Package: Perceptron code + Python simulator (60pts):

- In case your perceptron program does not work correctly, showcase 4 different toy testcases that work correctly, to convince us that all the instructions work correctly by your ISA design and simulator support. For each program, explain clearly what it is supposed to do. (You can use the my-straightforward-ISA in Appendix as a starting point.)

1. **Provide results with the following values of B:**

```
V1:      n = 100,  A = 5,    B = -6
V2:      n = 250,  A = -17,  B = 80
```

For each of the above cases, show the following:

I. **Assembly code (should be easy to read) of your program.** Make sure your assembly format is either obvious or well described, and that the code is well commented.

II. **Machine code (either in binary or hex) of your program.** This should be the input to your python simulator.

III. **Screenshots of your Python simulator's output for your program.** Highlight the final results (in data memory) and Dynamic Instruction Count.
You should provide sufficient screenshots to convince us that your ISA + Python package work correctly to yield the results of Perceptron algorithm.

- Up to 10 pts extra credit will be given to group(s) with the lowest overall Dynamic Instruction Count and/or the best UI design for simulator output.

2. **Include your Python simulator code here, well commented / annotated so that it's easy to see how it is corresponding to your ISA design.**

Appendix: "my-tiny-ISA" with 4 registers (R0 – R3)

instruction	Functionality	Encoding Format	MC Example	asm example
init Rx, imm	Rx = imm; PC++ imm: [0, 15]	**00 xx iiii**	00011001	init R1, 9
ld Rx, (Ry)	Rx = Mem[Ry]; PC++	**0100 xx yy**	01000100	ld R1, (R0)
st Rx, (Ry)	Mem[Ry] = Rx; PC++	**0101 xx yy**	01011011	st R2, (R3)
add Rx, Ry	Rx = Rx + Ry; PC++	**0110 xx yy**	01101001	add R2, R1
sub Rx, Ry	Rx = Rx – Ry; PC++	**0111 xx yy**	01111111	sub R3, R3
addi Rx, imm	Rx = Rx + imm; PC++ imm: {-2, -1, 1, 3}	**1000 xx ii**	10000001	addi R0, -1
sll Rx, imm	Rx = Rx << imm+1; PC++ imm: [1, 4]	**1001 xx ii**	10011011	sll R2, 4
sltR0 Rx, Ry	R0 = 1 if Rx < Ry R0 = 0 (otherwise) PC++	**1010 xx yy**	10100011	sltR0 R0, R3
bezR0 imm	if (R0==0) PC = PC + imm else PC++ imm: [-8, 7]	**1011 iiii**	10111100	bezR0 -4
jmp imm	PC = PC + imm imm: [-8, 7]	**1100 iiii**	11000010	jump 2
halt	ends the program	**11111111**	11111111	halt

Disassemble the machine code (in Instr mem) and analyze what the program is doing (with the given data mem).

Addr	Instruction mem content
…0000	**0011 1000**
…0001	**0010 0001**
…0010	**0110 1111**
…0011	**0100 0110**
…0100	**0110 0101**
…0101	**0101 0110**
…0110	**1000 1011**
…0111	**1010 1011**
…1000	**1011 0010**
…1001	**1100 1010**
…1010	**1111 1111**
all other	**0000 0000**

Addr	Data mem content
0…0000000	**0000 0000 0000 0010**
0…0000001	**0000 0000 0001 0001**
0…0000010	**0000 0000 0000 0011**
0…0000011	**0000 0000 0000 1000**
0…0000100	**0000 0000 0000 0101**
...	
all other	**0000 0000 0000 0100**

Submission Instructions:

While preparing for your submission files, keep in mind it is your responsibility to make all the files organized, and everything clear.

Individually, upload the following files (can be zipped into a single file) on Bb:

1. **`report.pdf`: a self-contain PDF report**
2. `ISA-sim.py`: Python simulator code for your ISA
3. `mcv1.txt`: machine code for input V1
4. `mcv2.txt`: machine code for input V2

Late submission policy and penalty:

submission by the end of the day of	late penalty on your total score
Sun - due date	0%
Mon	5%
Tue	10%
Wed	15%
Thu	20%