Table of Contents

A) ISA Intro

1. Introduction

- **Name of Architecture:** JHT
- **Overall Philosophy:**

  The philosophy behind this architecture is to have an efficient and user-friendly ISA Design.

- **Goals:**
  - Achieve Modular Exponentiation Program for any 16-bit P and Q
  - Achieve Best Match Count Program for any best-matching score

2. Instruction List

Instruction	Format	Opcode	Example
Add	Add Rx, Ry	000 xx yy	Rx = Rx + Ry
Sub	Sub Rx, Ry	001 xx yy	Rx = Rx - Ry
load	Load Rx, (Ry)	010 xx yy	Rx <- M[Ry]
Store	Store Rx, (Ry)	011 xx yy	M[Ry] <- Rx
Jump	Jump imm	100 iiii	PC -= imm
bezR1	bezR1 imm	101 iiii	If R1==0, PC += imm
SltR1	SltR1 Rx, Ry	110 xx yy	if Rx < Ry, R1 = 1
Init	Init Rx, imm	111 0 xx i	Rx = imm
ShiftL	ShiftL	111 10 00	R2 <<
ShiftR	ShiftR	111 10 01	R2 >>
And	AndR3	111 11 01	R3 AND 1
XOR	XorR0R1	111 11 00	R1 XOR R2

3. Register Design

Number of registers: 4 (R0 to R3)

R1 is for testing purposes when using bezR1 or sltR1.

4. Control Flow (Branches)

What types of branches are supported? How are the target addresses calculated? What is the maximum branch distance supported? Give examples of your assembly branch instructions and their corresponding machine code.

- **Branches supported:**
- **Target address calculus:**
- **Maximum branch distance:**
- **Examples:**

5. Data addressing modes

What addressing modes are supported for data memory? How are the addresses calculated? Give examples of your assembly load / store instructions and their corresponding machine code.

- **Addressing Modes:**
- **Address Calculation:**
- **Examples:**

B) Answers to Questions

1. Comparing to the sample of "My_straightforward_ISA", what are the unique features of your ISA? Explain why your ISA is better.

2. In what ways did you optimize for the two goals? If you optimized for anything additional, what and how?

3. What would you have done differently if you had 1 more bit for instructions? How about 1 fewer bit?

4. How did your team work together to accomplish this project? (Role of each team member, progress milestones, time spent individually and together?)

5. If you had a chance to restart this project afresh with 3 weeks' time, how would your team have done differently?

C) Software Package

1. Algorithms (in assembly code) of the two programs

Make sure your assembly format is either obvious or well described, and that the code is well commented.

Extra credits: provide a convincing estimation:

- on the dynamic instruction count for P1 (ME) with P = ~1000 and Q = ~500
- on the worst-case scenario of dynamic instruction count for P2(BMC).

2. Machine code for each of the programs

We will not correct/grade the machine code. You will also not be able to verify whether your code works correctly or not in this project (without a simulator). Therefore, you have to rely on the help of the disassembler, strive to make your algorithm simple and easy to

understand, as well as pursue the sw-hw "codesign" – this will avoid putting tremendous complexity at either the software or the hardware end. State any assumptions you make. You cannot assume anything about the values in registers or data memory, other than those specifically given, when the program starts. This means, for example, that if you need zeroes in registers or memory, you need to put them there.

3. Output of your Python disassembler for each program

This should be a line-by-line explanation of the machine code, what is done by each line of code.

4. Python code for your ISA's disassembler

D) Hardware Implantations

1. 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).

2. CPU Datapath

A schematic including your register file, ALU, PC logic, and memory components (see textbook ch 7.3.1).

3. Control logic

Decoder truth-table indicating how each control signal is generated from an instruction (see textbook ch 7.3.2).