Project Dojo

Haig Emirzian | Nadeya De Diago

Job Description

Nadeya De Diago	Haig Emirzian
● CPU architecture - Implemented instruction memory, 4 register files, ALU, data memory, and PC - Computes addition and subtraction - Loads from memory - Has a sequential datapath ● Manual - Composed a handbook on how to use the assembler program, architecture description of the CPU, and the binary encoding of the instructions	● Assembler - Implemented read/write function in Python - Converts assembly file to a hexadecimal image file ● Assembly language - Created a simple set of instructions to run the assembler on ● Manual - Composed a handbook on how to use the assembler program, architecture description of the CPU, and the binary encoding of the instructions

Assembler Program

Our assembler was coded in Python. When running the program, the user is immediately given an image file. The program would input the file simpleInstructions.s, and it would convert the assembly program into an image file that can be inputted into our CPU. The result is an image file that is eligible to be inputted in the RAM of the CPU. The CPU would then run on the given transcribed input. Our assembler works as follows: it decides which of the three instructions it's going to take in(LDR, SUB, or ADD), then based on the instruction it adds the specific opcode we gave it, after that it converts it to hex for our RAM in the CPU to read. All you have to do at that point is input your data into the memory portion of the RAM and our CPU will do LDR, SUB, or ADD with whatever numbers provided. Our demo assembly code runs perfectly with any numbers.

Architecture Description

Our CPU includes the essential parts of a sequential path. These parts are instruction memory, register file, ALU, data memory, and PC. The instruction memory, register file, and data memory are all read combinationally. Our CPU contains 4 registers: X0, X1, X2, and X3. Our CPU can perform LDR, ADD, and SUB operations. As it does addition, subtraction, and load, the CPU must pick between three possible paths, while the opcode determines which one should be outputted. When an instruction is finished, the clock keeps running and the CPU keeps working using 00s as input because that is what is immediately added to the file to represent empty space.

Binary Encoding

1. LDR X1 17
2. LDR X2 28
3. LDR X3 33
4. ADD X1 X1 X1
5. ADD X1 X2 X1
6. SUB X1 X1 X3

1. LDR X1 17

Opcode	imm11	1st Register	Target
31 <—>21	20<———>10	9<—>5	4<—>0
11111000010	00000010001	00000	0001

Result: 11111000010000000100010000000001

2. LDR X2 28

Opcode	imm11	1st Register	Target
31 <—>21	20<———>10	9<—>5	4<—>0
11111000010	00000010001	00000	0010

Result: 11111000010000000100010000000010

3. LDR X3 33

Opcode	imm11	1st Register	Target
31 <—>21	20<———>10	9<—>5	4<—>0
11111000010	00000100001	00000	00011

Result: 11111000010000001000010000000011

4. ADD X1 X1 X1

31 <—>21	20<—>16	15<—>10	9<—>5	4<—>0
10001011001	00001	111000	00001	00001

Result: 10001011001000011110000000100001

5. ADD X1 X2 X1

31 <—>21	20<—>16	15<—>10	9<—>5	4<—>0
10001011001	00001	111000	00010	00001

Result: 10001011001000011110000001000001

6. SUB X1 X1 X3

31 <—>21	20<—>16	15<—>10	9<—>5	4<—>0
11001011001	00011	111000	00001	00001

Result: 11001011001000111110000000100001