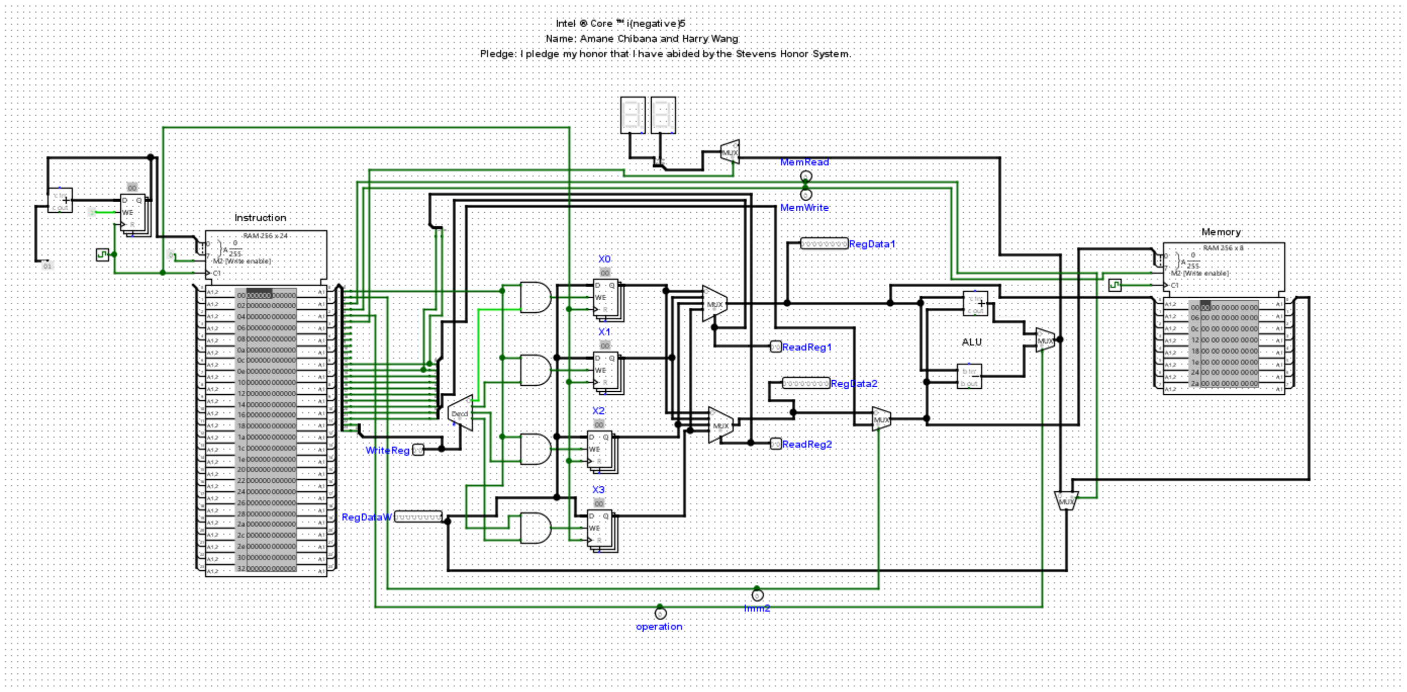
Intel ® Core ™ i(negative)5

1st gen

By: Amane Chibana and Harry Wang

Pledge: I pledge my honor that I have abided by the Stevens Honor System.


Intel ® Core ™ i(negative)5
Name: Amane Chibana and Harry Wang
Pledge: I pledge my honor that I have abided by the Stevens Honor System.
Instruction
RAM 256 x 24
Memory
RAM 256 x 8
MemRead
MemWrite
RegData1
ReadReg1
RegData2
ReadReg2
X0
X1
X2
X3
ALU
MUX
WriteReg
RegDataW
operation
Imm2

JOB DESCRIPTION

Amane:

- Design and create Logism cpu file
- Create manual for design and instructions

Harry:

- Create python assembler for instructions and memory
- Write the demo program

ASSEMBLER PROGRAM AND HOW TO RUN

Writing the instructions is very similar to assembly. The arithmetic functions can be done through "instruction" <destination register> <register 1> <input 2> and for memory instructions you only have to specify two arguments. (further explained in other sections).

```
load r1 2
add r2 r2 r1
# result inside r2 is 0x67 + 0x00
.data 0x45 0x65 0x67 0xa5
```

Write the instructions in a file named "project2.txt". Write each instruction on separate lines. To insert comments, use "#" at the start of the line. You can also specify what is stored in the memory by writing ".data" after your instructions. All the data will be on one line separated by spaces and will be placed in the memory starting from address 0x00. Make sure to contain the assembler and instructions in the same folder. Run the assembler (written in python named "project2.py") and it will generate two image files, one for the memory and another for the instruction memory. Load each file into their respective memories by right clicking the memory in the .circ file and pressing "load image." Go to the directory where the image is saved and load the data on. To start the CPU, go to simulate > auto-tick frequency and select your tick rate for the speed. Then go to simulate > auto-tick-enabled to begin the clock and the CPU will start the program you wrote. Once that is complete you can finally see the CPU run the program

Architecture Design

- Four general purpose registers
  - referred to in assembly program as r0-r3
  - store 1 byte of data each
- Memory File
  - Refer to each byte of memory with their address using load or store(0x00 -> 0xFF)
- Possible instructions include
  - Arithmetic(add, sub)
  - Memory related(load,str)
- Each instruction is written in 24 bits.
  - Bits 0 – 12 as the opcode
  - 12 – 23 choosing specific registers/immediate values
- Digital Display that displays hex digit result of addition and subtraction when they are called

Instruction Descriptions

Instruction	Dst	Reg1	000000	Reg2	Opcode
add <dst> <reg1> <reg2>	00-11	00-11	000000	00-11	000000100001
sub <dst> <reg1> <reg2>	00-11	00-11	000000	00-11	000000110001
load <dst> <reg2>	00-11	00	000000	00-11	000000000101
store <reg1> <reg2>	00	00-11	000000	00-11	000000001000

Instruction	Dst	Reg1	Imm	Opcode
add <dst> <reg1> <imm>	00-11	00-11	8 bit #	000000100011
sub <dst> <reg1> <imm>	00-11	00-11	8 bit #	000000100011
load <dst> <imm>	00-11	00	8 bit #	000000000111
store <reg1> <imm>	00	00-11	8 bit #	000000001010

There are a total of 24 bits per instruction. 00-11 refer to the possible register(r0-r3)

In both situations (starting from the MSB to LSB), the first two bits refer to the destination register. The next two bits refer to the register which will be the first argument. The next 8 bits will be determined if you choose to use an immediate or not. When an immediate is used, all 8 bits will be calculated as the number to use as an argument. If the second argument is a register, there will be 6 filler bits and the last two bits decide the register used. Then the rest will be used for the opcode. (Once again from MSB to LSB), the first 6 bits are filler bits that are not used. Then from left to right a bit for printing, deciding add/sub, writing to the memory, reading from the memory, using an immediate, and finally the LSB decides if the register is written to or not.

ARITHMETIC INSTRUCTION 1: ADD

In the add instruction, the ALU calculates the sum of reg1 and reg2 and stores the result in the destination register. Bit 0 for register writing and bit 5 for printing are set to 1 to be used. Bit 4 for adding/subtracting is kept at 0 because 0 is for addition. If an immediate is used in place for reg 2 bit 1 is also flipped in the opcode and 8 bits are used from bit 12-19 to send the immediate value.

Format: add <dst> <reg1> <reg2> or sub <dst> <reg1> <imm>

ARITHMETIC INSTRUCTION 2: SUB

In the sub instruction, the ALU calculates the difference of reg1 and reg2 and stores the result in the destination register. Bit 0 for register writing and bit 5 for printing are set to 1 to be used. Bit 4 for adding/subtracting is set to 1 because 1 is for subtraction. If an immediate is used in place for reg 2 bit 1 is also flipped in the opcode and 8 bits are used from bit 12-19 to send the immediate value.

Format: sub <dst> <reg1> <reg2> or sub <dst> <reg1> <imm>

MEMORY LOADING: LOAD

In the load instruction, data is extracted from a specific address in the memory that is specified by immediate or reg2 and the stored into destination register. Bit 0 for register writing, and bit 3 for memory reading is set to 1. If an immediate is used in place for reg 2 bit 1 is also flipped in the opcode and 8 bits are used from bit 12-19 to send the immediate value that is used as the address. There is no need to calculate the offset for the address when using an immediate, so you can select the location in memory by using decimal numbers 0-255.

Format: load <dst> <reg2> or load <dst> <imm>

MEMORY STORING: STORE

In the store instruction, data is extracted from reg1 and then stored in a specific address in the memory that is specified by immediate or reg2. Bit 4 for memory reading is set to 1. If an immediate is used in place for reg 2 bit 1 is also flipped in the opcode and 8 bits are used from bit 12-19 to choose the address where the data will be stored. There is no need to calculate the offset for the address when using an immediate, so you can select the location in memory by using decimal numbers 0-255.

Format: store <reg1> <imm> or store <reg1> <reg2>