CPU GPU Final Exam

Jake Travis Calhoun

After building a microprocessor in VHDL (which can be found here), I found this Python implementation fairly straightforward. Drawing on material from both classes, I was able to complete the project in about six hours. The structure is heavily inspired by that coursework, using a sequencer, decoder, and fetch unit as the core components. I initially planned to place the machine-execution logic in a separate ALU class, but doing so introduced a circular dependency, so I kept the logic inline instead.

Implementation

The program begins by loading the object file using the provided code. It then starts the sequencer, which executes instructions until the machine halts. Each cycle starts at the instruction counter (IC): the sequencer fetches the instruction, increments the IC, decodes the instruction, executes it, and processes the resulting state until it encounters an exception.

The project uses only Python's standard library, so it runs without external dependencies. Here is the help output:

```
(.venv) PS C:\Users\travi\Documents\projects\cpu_gpu_final> python .\main.py -h
usage: main.py [-h] [--input INPUT] [--start_addr START_ADDR]

CPU GPU Final microprocessor emulator

options:
  -h, --help            show this help message and exit
  --input INPUT         name of the hex file inside the input folder
  --start_addr START_ADDR
                        hexadecimal starting address if none is defined in the program file
```

T1 file: python .\main.py --input t1.obj --start_addr 0c0
T2 file: python .\main.py --input t2.obj --start_addr 100
T3 file: python .\main.py --input t3.obj --start_addr 0FF

Testing/Outputs

t1.obj

```
starting addr:0xc0
 IC     IR    INST  MAR    AC-reg  Index Registers
0C0:  050404  LD    IMM  AC[000050] X0[000] X1[000] X2[FFF] X3[000]
0C1:  200800  ADD   200  AC[000080] X0[000] X1[000] X2[FFF] X3[000]
0C2:  200800  ADD   200  AC[0000B0] X0[000] X1[000] X2[FFF] X3[000]
0C3:  102840  SUB   102  AC[0000A7] X0[000] X1[000] X2[FFF] X3[000]
0C4:  050C00  J     050  AC[0000A7] X0[000] X1[000] X2[FFF] X3[000]
050:  000000  HALT       AC[0000A7] X0[000] X1[000] X2[FFF] X3[000]
Machine Halted - HALT intruction executed
```

t2.obj

```
 IC     IR    INST  MAR    AC-reg     Index Registers
100:  200400  LD    200  AC[000030] X0[000] X1[000] X2[FFF] X3[000]
101:  201840  SUB   201  AC[-00001] X0[000] X1[000] X2[FFF] X3[000]
102:  202800  ADD   202  AC[00000F] X0[000] X1[000] X2[FFF] X3[000]
103:  005844  SUB   IMM  AC[00000A] X0[000] X1[000] X2[FFF] X3[000]
104:  050C84  JN    ???  AC[00000A] X0[000] X1[000] X2[FFF] X3[000]
Machine Halted - illegal addressing mode
```

t3.obj

```
starting addr:0xff
 IC     IR    INST  MAR    AC-reg     Index Registers
0FF:  075400  LD    075  AC[000030] X0[000] X1[000] X2[FFF] X3[000]
100:  040804  ADD   IMM  AC[000070] X0[000] X1[000] X2[FFF] X3[000]
101:  077440  ST    077  AC[000070] X0[000] X1[000] X2[FFF] X3[000]
102:  005884  CLR   IMM  AC[000000] X0[000] X1[000] X2[FFF] X3[000]
103:  076400  LD    076  AC[000020] X0[000] X1[000] X2[FFF] X3[000]
104:  077800  ADD   077  AC[000090] X0[000] X1[000] X2[FFF] X3[000]
105:  0309C4  ????  IMM  AC[000090] X0[000] X1[000] X2[FFF] X3[000]
Machine Halted - undefined opcode
```

Conclusion

This class has given me a much deeper understanding of microprocessors, emulators, and the process of moving between different assembly languages. As a result, I now feel confident in my ability to write an emulator or even explore entirely new assembly languages.