**Computer Architecture and Organization (CSC3750)**

**Project 01**

The assignment is a multipart assignment. The overall goal is to develop a virtual computer (VC) that can execute a simple set of machine-level program instructions.

# Part 1: Instruction Set Architecture Definition

# & Virtual Computer Assembler

In this part of the assignment, your task is to implement an assembler that can translate assembly-level instructions into virtual-computer machine instructions (VCI). The VC assembly-level instructions define the VC’s ISA (Instruction Set Architecture). The VC register set and ISA are defined below, along with the VC instruction formats. Use Visual Studio 2019 with C# to implement the assembler as a console-based application.

# Virtual Computer Register Set

1. ACC – 32-bit Accumulator
2. REG\_A – 32-bit General Purpose Register
3. REG\_B– 32-bit General Purpose Register
4. REG\_C – 32-bit General Purpose Register
5. REG\_D – 32-bit General Purpose Register
6. CMP – Stores the result of a comparison operation (0, 1, -1)
7. PC – Program Counter

# Virtual Computer Instruction Set Architecture (ISA)

## Special Instructions

1. HALT – Transfers control to the operating system

## Clear Instructions

1. CLR ACC – Sets ACC to 0
2. CLR REG\_A – Sets REG\_A to 0
3. CLR REG\_B – Sets REG\_B to 0
4. CLR REG\_C – Sets REG\_C to 0
5. CLR REG\_D – Sets REG\_D to 0

## Add Instructions

1. ADD REG\_A – Add the 32-bit integer value stored in REG\_A to ACC
2. ADD REG\_B – Add the 32-bit integer value stored in REG\_B to ACC
3. ADD REG\_C – Add the 32-bit integer value stored in REG\_C to ACC
4. ADD REG\_D – Add the 32-bit integer value stored in REG\_D to ACC

## Move Instructions

1. MOV ACC REG\_A – Moves a 32-bit integer value from REG\_A to ACC
2. MOV ACC REG\_B – Moves a 32-bit integer value from REG\_B to ACC
3. MOV ACC REG\_C – Moves a 32-bit integer value from REG\_C to ACC
4. MOV ACC REG\_D – Moves a 32-bit integer value from REG\_D to ACC
5. MOV REG\_A ACC – Moves a 32-bit integer value from ACC to REG\_A
6. MOV REG\_B ACC – Moves a 32-bit integer value from ACC to REG\_B
7. MOV REG\_C ACC – Moves a 32-bit integer value from ACC to REG\_C
8. MOV REG\_D ACC – Moves a 32-bit integer value from ACC to REG\_D
9. MOVI REG\_A VALUE – Moves a literal 32-bit integer value into REG\_A
10. MOVI REG\_B VALUE – Moves a literal 32-bit integer value into REG\_B
11. MOVI REG\_C VALUE – Moves a literal 32-bit integer value into REG\_C
12. MOVI REG\_D VALUE – Moves a literal 32-bit integer value into REG\_D

## Store Instructions

1. STORE REG\_A ADDR – Stores a 32-bit integer value from REG\_A to the memory location defined by ADDR
2. STORE REG\_B ADDR – Stores a 32-bit integer value from REG\_A to the memory location defined by ADDR
3. STORE REG\_C ADDR – Stores a 32-bit integer value from REG\_A to the memory location defined by ADDR
4. STORE REG\_D ADDR – Stores a 32-bit integer value from REG\_A to the memory location defined by ADDR

## Load Instructions

1. LOAD REG\_A ADDR – Loads a 32-bit integer value from the memory location defined by ADDR to REG\_A
2. LOAD REG\_B ADDR – Loads a 32-bit integer value from the memory location defined by ADDR to REG\_B
3. LOAD REG\_C ADDR – Loads a 32-bit integer value from the memory location defined by ADDR to REG\_C
4. LOAD REG\_D ADDR – Loads a 32-bit integer value from the memory location defined by ADDR to REG\_D

## Compare Instructions

1. CMP REG\_A – Compares the 32-bit integer value stored in REG\_A to ACC
2. CMP REG\_B – Compares the 32-bit integer value stored in REG\_B to ACC
3. CMP REG\_C – Compares the 32-bit integer value stored in REG\_C to ACC
4. CMP REG\_D – Compares the 32-bit integer value stored in REG\_D to ACC

# Virtual Computer Instruction Format (Opcodes, Machine Code)

1. HALT – 0x2000
2. CLR ACC – 0x1000
3. CLR REG\_A – 0x1001
4. CLR REG\_B – 0x1002
5. CLR REG\_C – 0x1003
6. CLR REG\_D – 0x1004
7. ADD REG\_A – 0x0001
8. ADD REG\_B – 0x0002
9. ADD REG\_C – 0x0003
10. ADD REG\_D – 0x0004
11. MOV ACC REG\_A – 0x0101
12. MOV ACC REG\_B– 0x0102
13. MOV ACC REG\_C– 0x0103
14. MOV ACC REG\_D– 0x0104
15. MOV REG\_A ACC– 0x0201
16. MOV REG\_B ACC – 0x0202
17. MOV REG\_C ACC – 0x0203
18. MOV REG\_D ACC – 0x0204
19. MOVI REG\_A VALUE – 0x0601 IMMEDIATE\_VALUE
20. MOVI REG\_B VALUE – 0x0602 IMMEDIATE\_VALUE
21. MOVI REG\_C VALUE – 0x0603 IMMEDIATE\_VALUE
22. MOVI REG\_D VALUE – 0x0604 IMMEDIATE\_VALUE
23. STORE REG\_A ADDR – 0x0301 ADDR
24. STORE REG\_B ADDR – 0x0302 ADDR
25. STORE REG\_C ADDR – 0x0303 ADDR
26. STORE REG\_D ADDR – 0x0304 ADDR
27. LOAD REG\_A ADDR – 0x0401 ADDR
28. LOAD REG\_B ADDR – 0x0402 ADDR
29. LOAD REG\_C ADDR – 0x0403 ADDR
30. LOAD REG\_D ADDR – 0x0404 ADDR
31. CMP REG\_A – 0x0501
32. CMP REG\_B – 0x0502
33. CMP REG\_C – 0x0503
34. CMP REG\_D – 0x0504

# Virtual Computer Assembler

Implement an assembler to assemble the VC ISA instructions into VC machine code, also called opcode (operation-code). Store the generated machine code as ASCII characters in a text file. Store each generated machine opcode on a separate line. The assembler should be written using the C# programming language.

Here’s an example VC assembly program:

CLR ACC

MOVI REG\_A 0x0000 0001

ADD REG\_A

MOVI REG\_B 0x0000 0002

ADD REG\_B

MOV REG\_C ACC

HALT

Here’s an example VC assembly program:

CLR ACC

LOAD REG\_A 0x0400 0000

ADD REG\_A

LOAD REG\_B 0x0400 0004

ADD REG\_B

MOVE REG\_C ACC

STORE REG\_C 0x0800 0000

HALT

# Part 2: Virtual Computer Program Loader & Step-Through Debugger

In this part of the assignment, your task is to implement a VC Program Loader and a Step-Through Debugger. Use Visual Studio 2019 with C# to implement the Program Loader and Step-Through Debugger as a Windows-based Graphical User Interface (GUI) application. The GUI should include TextBoxes to display the Register values and a GridView to display the VC memory. Values in memory should be stored in an ASCII text file with each memory location/memory value pair on a line together. This ASCII text file represents the VC Virtual Memory component.

# Virtual Computer Virtual Memory ASCII File Example

0x0040 0000 0x0000 0001

0x0040 0002 0x0000 0002