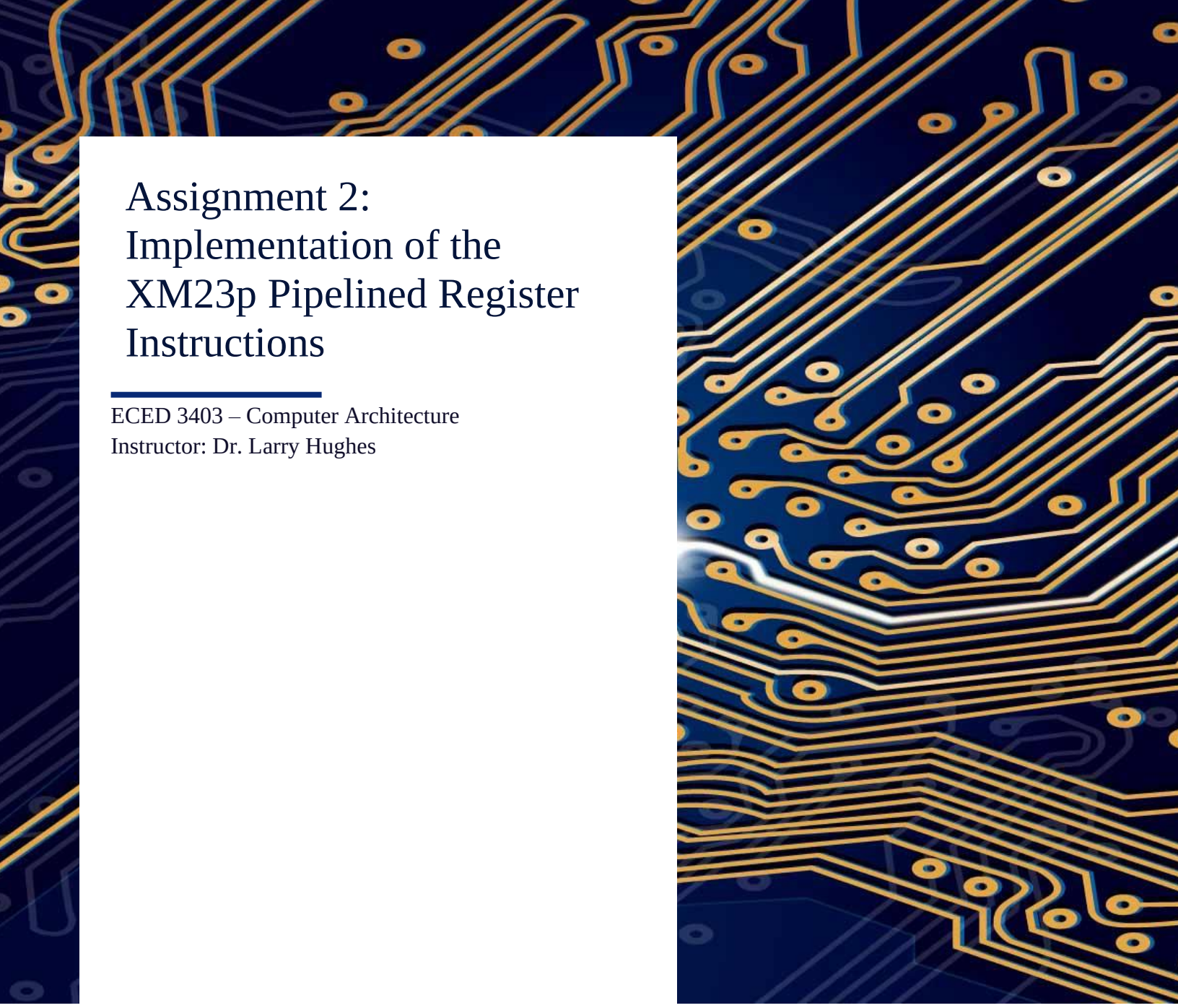

Assignment 2: Implementation of the XM23p Pipelined Register Instructions

ECED 3403 – Computer Architecture
Instructor: Dr. Larry Hughes

JULY 24

Name: MD Iftekhar Hossain Rafi
Student ID: B00871031

Table of Contents

Cover Page
..... 1
Objective 5
Design 6
Source Files: 6
Header Files: 6
loader.c 6
manager.c 6
runmode.c 6
pipeline.c 6
decode_instructions.c 6
execute_instructions.c 7
instructions.c 7
debugger_mode.c 7
memory.c 7
srecord.c 7
pipeline.h 7
decode_instructions.h 7
execute_instructions.h 7
instructions.h 7
loader.h 8
manager.h 8
memory.h 8
debugger_mode.h 8
Flowchart 8
Pseudocode: 14
Data Dictionary 20
Implementation 26
Testing 27
Test ADD Instruction 27
Test ADDC Instruction 29

Test SUB Instruction 31
Expected Results: 32
Test SUBC Instruction 35
Expected Results for `Test_SUBC.asm` 36
Test DADD Instruction 39
Assembly Code and Expected Results: 40
Test CMP Instruction 42
Expected Results: 43
Test XOR Instruction 45
Expected Results: 46
Test AND Instruction 49
Expected Results 50
Summary of Final Register Values 51
Summary of Final PSW Values 51
Test OR Instruction 53
Expected Results: 54
Test BIT Instruction 56
Instructions Breakdown: 57
Expected Output: 57
Test BIC Instruction 59
Expected Results: 60
Test BIS Instruction 62
Expected Results for BIS Instruction Test 63
Final Register State: 64
Final PSW Flags: 64
Test SWAP Instruction 67
Expected Results 68
Summary of Expected Register State: 68
Test SRA Instruction 70
Expected Results 71
Summary of Expected Register State: 71
Test RRC Instruction 73

Expected Results 74
Summary of Register State 74
PSW State Check 74
Test SWPB Instruction 76
Expected Results: 77

Expected Results: 80

Objective

The objectives of this assignment are to:

1. **Implement a Three-Stage Pipeline:** Develop and integrate a three-stage pipeline (Fetch, Decode, Execute) for the XM23p processor, transitioning from a sequential von Neumann architecture to a pipelined architecture to improve instruction throughput.
2. **Design and Code Register Instructions:** Implement eighteen register instructions (from ADD to SXT) and four register initialization instructions (MOVL to MOVH) in C, ensuring they operate correctly within the pipelined architecture.
3. **Develop a Fetch Mechanism:** Split the Fetch stage into two parts (F0 and F1), where F0 handles copying the Program Counter (PC) to the Instruction Memory Address Register (IMAR) and signalling a read, and F1 retrieves the instruction from memory and supplies it to the Instruction Register (IR).
4. **Decode and Execute Instructions:** Create a single-stage Decode process that extracts opcodes and operands and an Execute stage that performs operations using the decoded information, including updates to the Program Status Word (PSW).
5. **Ensure Proper Pipeline Operation:** Achieve parallel operation of pipeline stages (Fetch, Decode, Execute) to ensure that non-data memory access instructions complete every two clock cycles, enhancing the processor's efficiency.
6. **Test and Validate the Implementation:** Develop comprehensive tests to verify the correct operation of the implemented instructions and pipeline stages, including edge cases and PSW updates.
7. **Document and Demonstrate the Solution:** Provide a detailed design document, a well-commented and structured codebase, and thorough test results. Demonstrate the loader and the implementation to validate the functionality before final submission.

Design

The design for this section will be shown as a combination of flowcharts and pseudocode. The program is structured with the following header and source files with a short description of their purpose:

Header Files:

1. debugger_mode.h
2. decode_instructions.h
3. execute_instructions.h
4. instructions.h
5. loader.h
6. pipeline.h

Source Files:

1. debugger_mode.c
2. decode_instructions.c
3. execute_instructions.c
4. instructions.c
5. loader.c
6. manager.c
7. memory.c
8. pipeline.c
9. srecord.c

loader.c

Purpose: Handles the loading of instructions and data into memory from external files. This file typically includes functions to read files, parse their contents, and store the parsed instructions and data in the appropriate memory locations.

manager.c

Purpose: Manages the overall control and execution flow of the program. This file coordinates the various components of the emulator, such as loading instructions, executing them, and handling input/output operations. It often acts as the main control loop for the program.

runmode.c

Purpose: Implements the different modes of execution for the emulator, such as running in normal mode, debugging mode, or stepping through instructions one by one. This file includes functions to initialize and manage the program counter (PC) and handle breakpoints.

pipeline.c

Purpose: Manages the instruction pipeline stages of the emulator, such as fetching, decoding, and executing instructions. This file includes functions to handle the different stages of the pipeline and ensure instructions flow correctly through them.

decode_instructions.c

Purpose: Handles the decoding of instructions fetched from memory. This file includes functions to interpret the binary representation of instructions and extract relevant fields, such as opcode, source and destination registers, and immediate values.

execute_instructions.c

Purpose: Contains the implementation of the specific instructions that the emulator supports. This file includes functions to execute each type of instruction, such as arithmetic operations, logical operations, and data movement instructions.

instructions.c

Purpose: Implements the detailed execution logic for each supported instruction, updating the program status word (PSW) and register values as needed. This file contains the core functions for performing arithmetic operations, logical operations, and other instructions.

debugger_mode.c

Purpose: Implements the debugging functionality for the emulator, allowing users to step through instructions, set breakpoints, and inspect register and memory values. This file includes functions to manage the debugging state and handle user inputs for debugging commands.

memory.c

Purpose: Manages the memory subsystem of the emulator, including allocation, initialization, reading, and writing memory. This file includes functions to handle memory operations and ensure the memory state is correctly maintained during program execution.

srecord.c

Purpose: Handles the loading and parsing of S-record formatted files, which are a common format for representing binary data. This file includes functions to read S-record files, extract the data, and load it into the appropriate memory locations.

pipeline.h

Purpose: Provides function declarations and macro definitions related to the instruction pipeline stages. This header file defines the interface for pipeline-related functions used in the emulator.

decode_instructions.h

Purpose: Provides function declarations and macro definitions related to instruction decoding. This header file defines the interface for functions that interpret and extract fields from binary instructions.

execute_instructions.h

Purpose: Provides function declarations and macro definitions related to instruction execution. This header file defines the interface for functions that perform the actual execution of instructions.

instructions.h

Purpose: Provides function declarations and macro definitions related to updating the program status word (PSW) and executing specific instructions. This header file defines the interface for functions that handle arithmetic, logical, and other operations.

loader.h

Purpose: Provides function declarations and macro definitions related to loading instructions and data into memory. This header file defines the interface for functions that read and parse input files and store their contents in memory.

manager.h

Purpose: Provides function declarations and macro definitions related to managing the overall control and execution flow of the program. This header file defines the interface for functions that coordinate the various components of the emulator.

memory.h

Purpose: Provides function declarations and macro definitions related to memory management. This header file defines the interface for functions that handle memory operations such as allocation, initialization, reading, and writing.

debugger_mode.h

Purpose: Provides function declarations and macro definitions related to debugging functionality. This header file defines the interface for functions that manage the debugging state and handle user inputs for debugging commands.

Flowchart

Flowcharts have been created for all source code files except for instructions.c- which is shown through pseudocode.

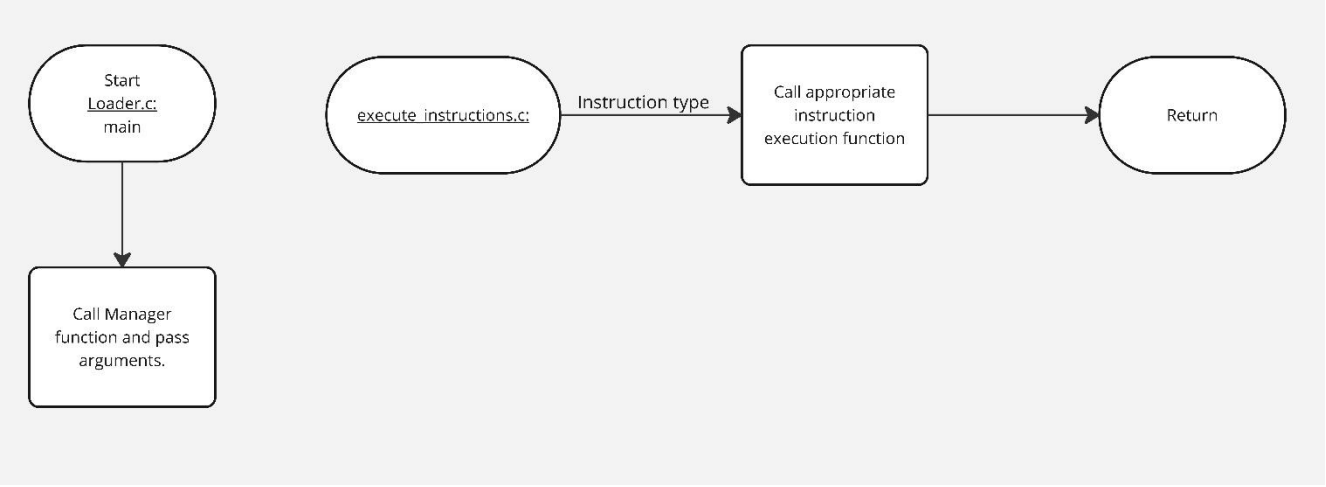

Figure 1 Flowchart showing control flow for Loader.c (entry point) and execute_instructions.c

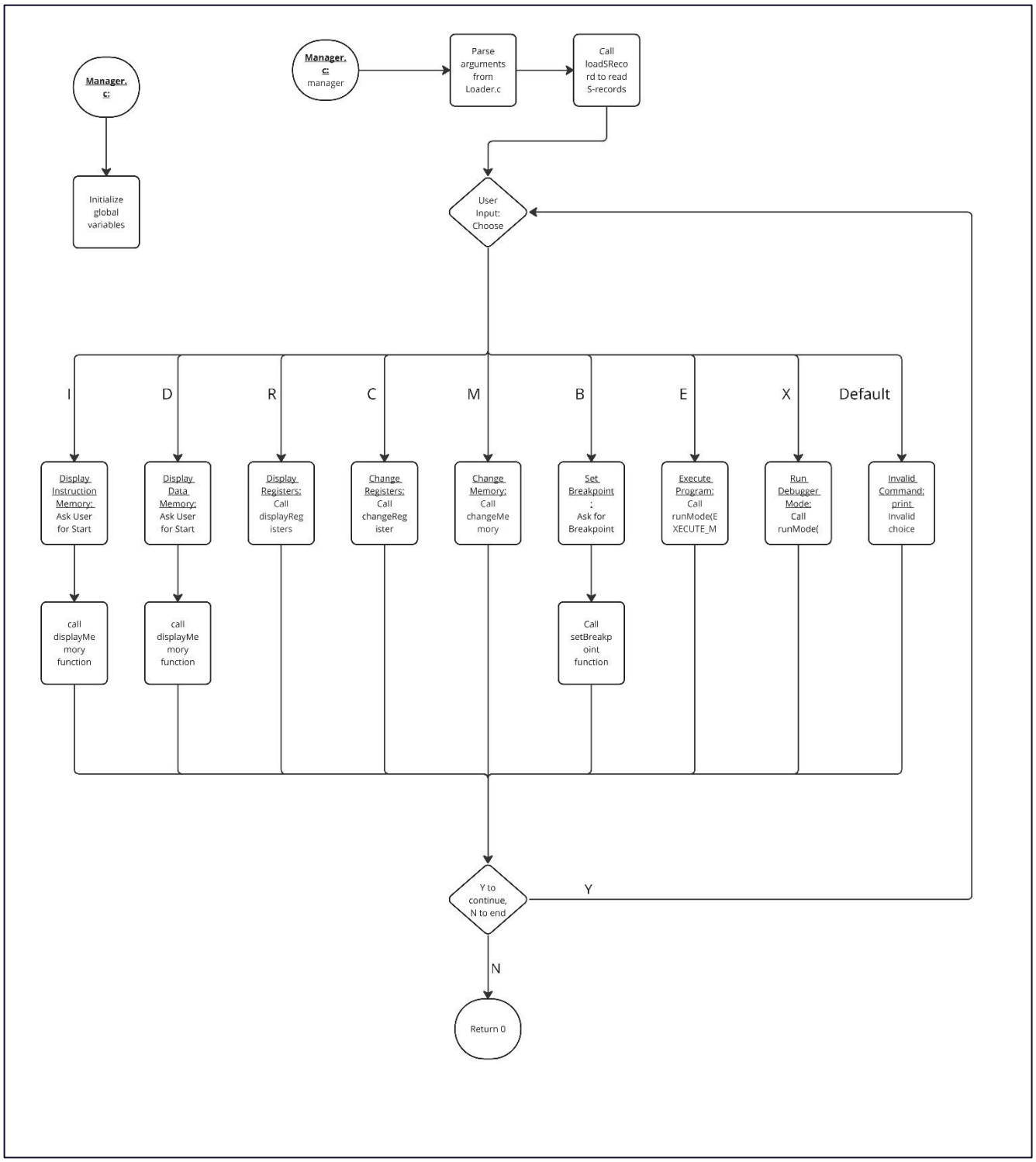

Figure 2 Flowchart for manager.c

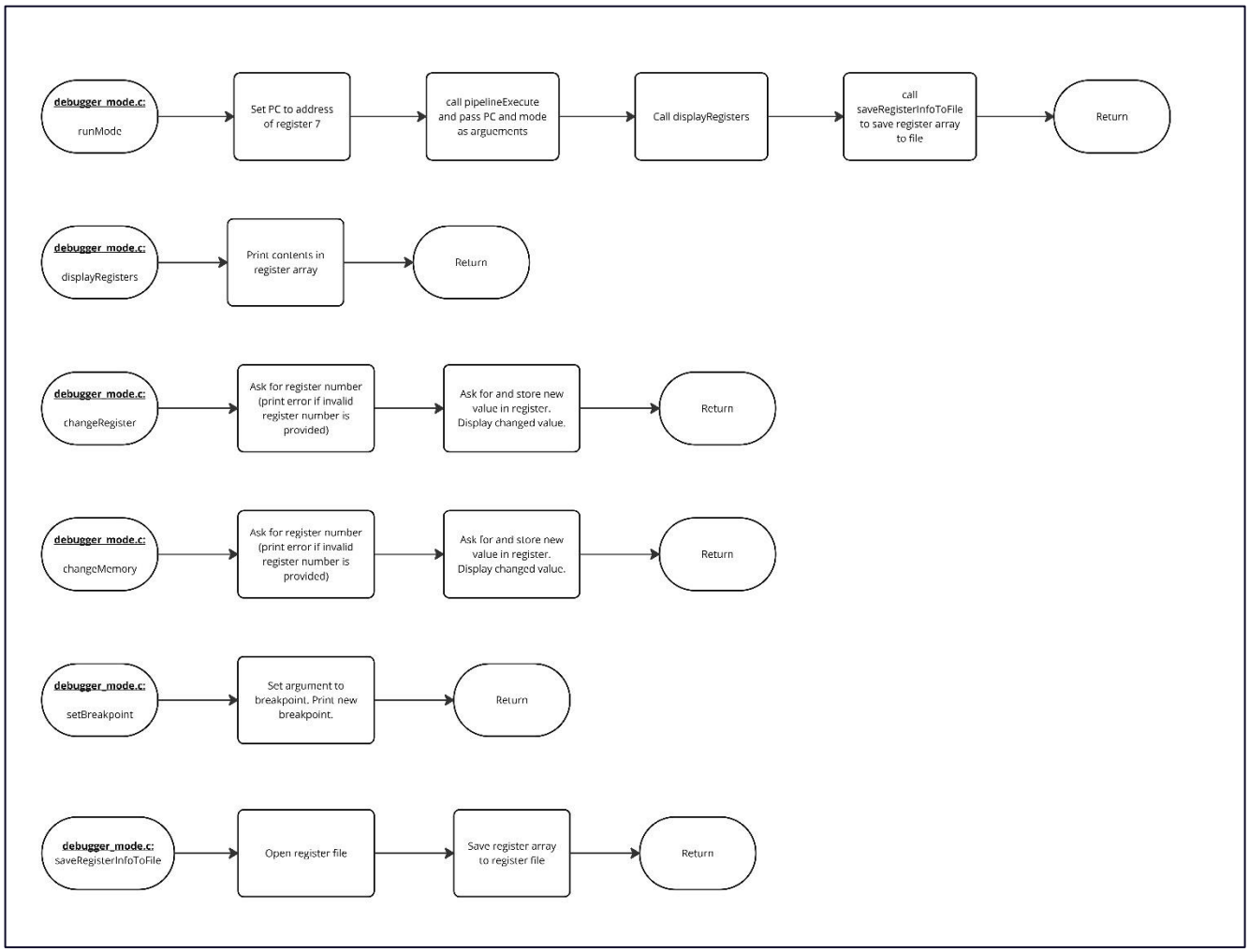

Figure 3 Flowchart for debugger_mode.c

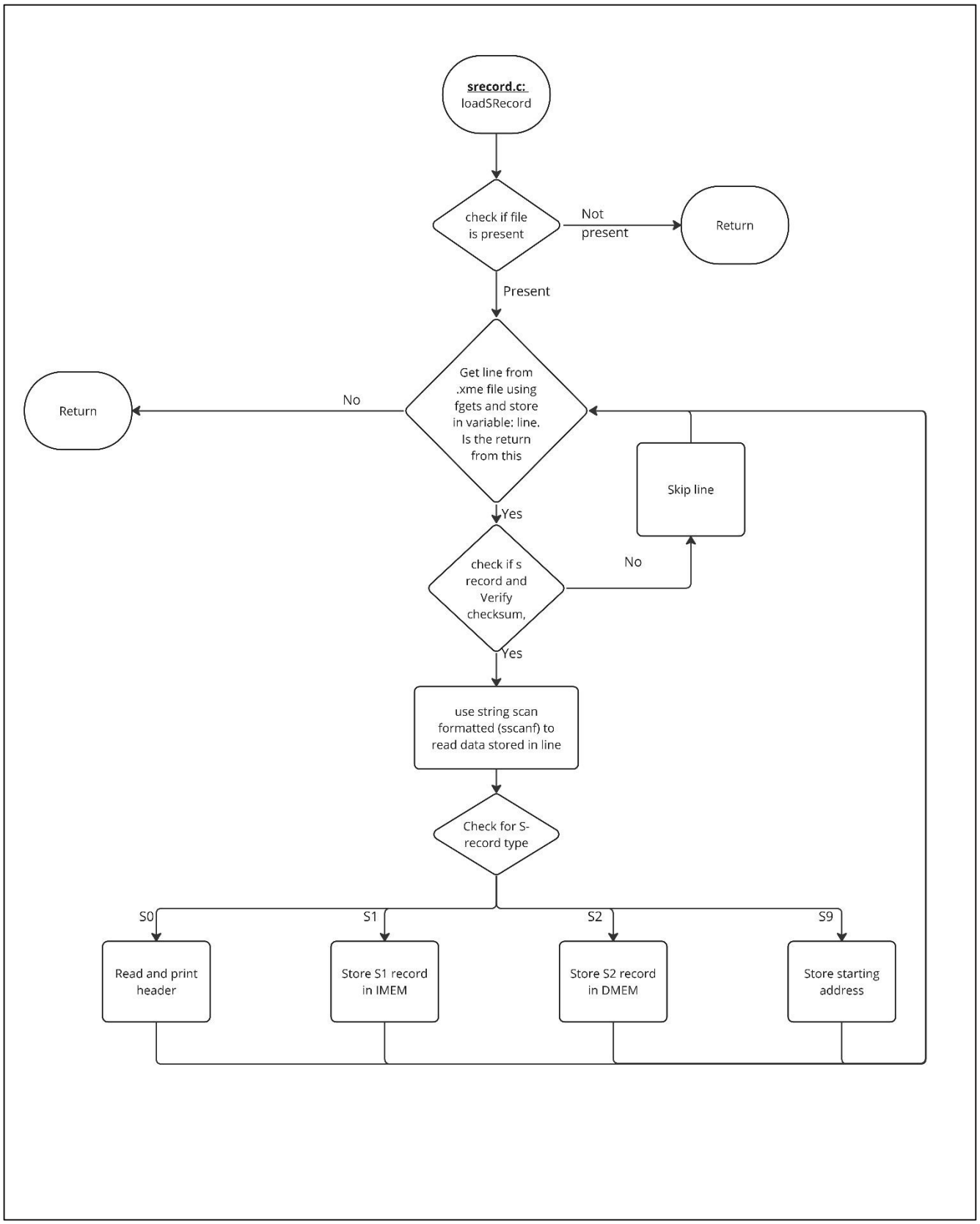

Figure 4 Flowchart for srecord.c

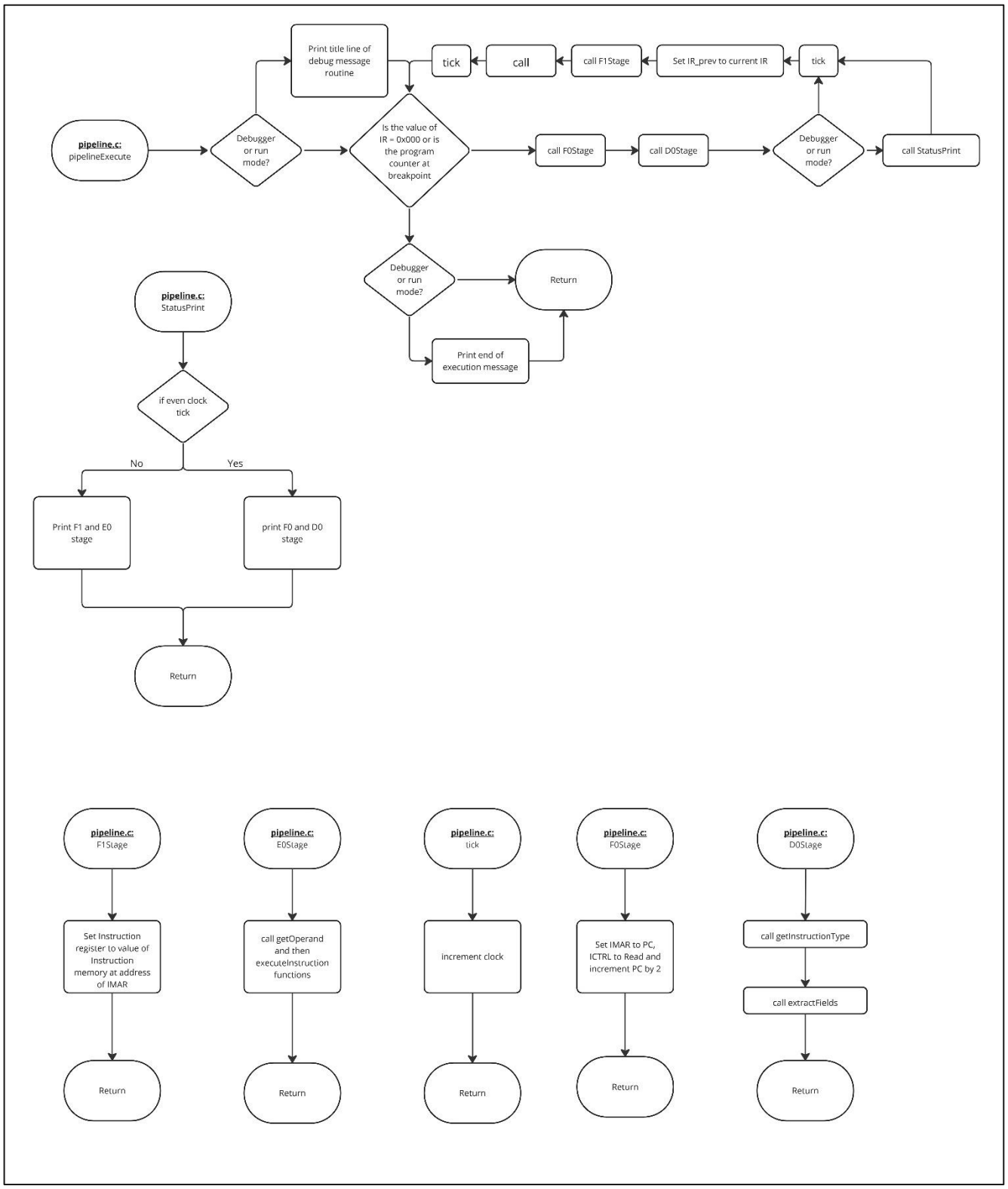

Figure 5 Flowchart for pipeline.c

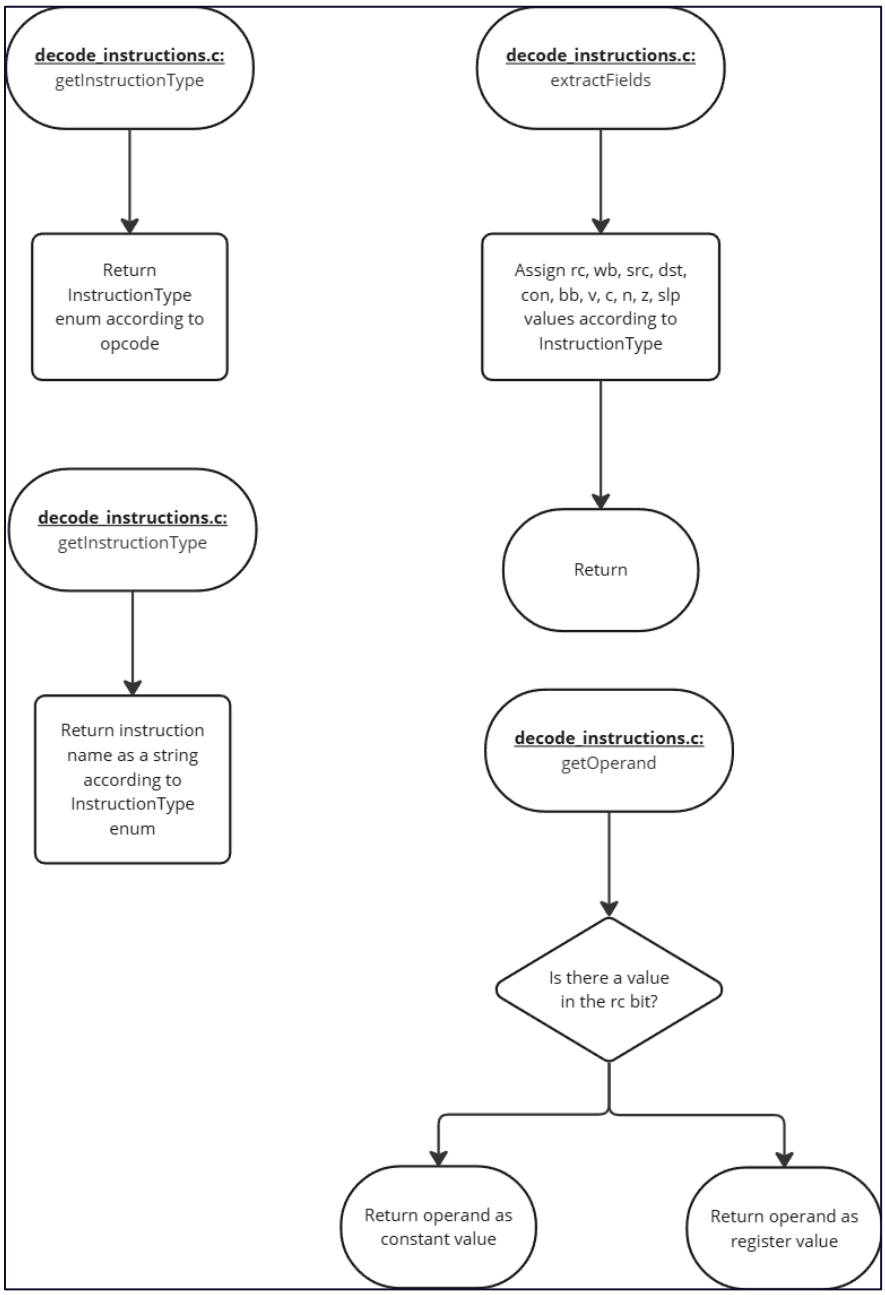

Figure 6 Flowchart for decode_instructions.c

Pseudocode:

Psudocode for instructions.c provided below:

```
# Instructions Pseudocode
# File Purpose: This file contains the functions to handle execution of specific instructions for the XM23P emulator.

# Function to update the Zero Flag (ZF)
FUNCTION updatePSW_ZF(result: UNSIGNED SHORT)
   SET psw.ZF TO (result IS EQUAL TO 0)
END FUNCTION

# Function to update the Sign Flag (SF)
FUNCTION updatePSW_SF(result: UNSIGNED SHORT)
   SET psw.SF TO ((result AND 0x8000) IS NOT EQUAL TO 0)
END FUNCTION

# Function to update the Overflow Flag (OF)
FUNCTION updatePSW_OF(a: UNSIGNED SHORT, b: UNSIGNED SHORT, result: UNSIGNED SHORT)
   SET psw.OF TO (((a XOR b XOR result XOR (result SHIFT RIGHT 1)) AND 0x8000) IS NOT EQUAL TO 0)
END FUNCTION

# Function to update the Carry Flag (CF)
FUNCTION updatePSW_CF(result: UNSIGNED INT)
   SET psw.CF TO (result IS GREATER THAN 0xFFFF)
END FUNCTION

# Function to execute an ADD instruction
FUNCTION executeADD(dst: UNSIGNED CHAR, operand: UNSIGNED SHORT)
   SET result TO reg_file[dst] + operand
   CALL updatePSW_ZF(result)
   CALL updatePSW_SF(result)
   CALL updatePSW_OF(reg_file[dst], operand, result)
   CALL updatePSW_CF((UNSIGNED INT)reg_file[dst] + operand)
   SET reg_file[dst] TO result
END FUNCTION

# Function to execute a SUB instruction
FUNCTION executeSUB(dst: UNSIGNED CHAR, operand: UNSIGNED SHORT)
```

```
    SET result TO reg_file[dst] - operand
    CALL updatePSW_ZF(result)
    CALL updatePSW_SF(result)
    CALL updatePSW_OF(reg_file[dst], operand, result)
    CALL updatePSW_CF((UNSIGNED INT)reg_file[dst] - operand)
    SET reg_file[dst] TO result
END FUNCTION

# Function to execute an AND instruction
FUNCTION executeAND(dst: UNSIGNED CHAR, operand: UNSIGNED SHORT)
    SET result TO reg_file[dst] AND operand
    CALL updatePSW_ZF(result)
    CALL updatePSW_SF(result)
    SET reg_file[dst] TO result
END FUNCTION

# Function to execute an OR instruction
FUNCTION executeOR(dst: UNSIGNED CHAR, operand: UNSIGNED SHORT)
    SET result TO reg_file[dst] OR operand
    CALL updatePSW_ZF(result)
    CALL updatePSW_SF(result)
    SET reg_file[dst] TO result
END FUNCTION

# Function to execute an ADDC instruction
FUNCTION executeADDC(dst: UNSIGNED CHAR, operand: UNSIGNED SHORT)
    SET result TO reg_file[dst] + operand + psw.CF
    CALL updatePSW_ZF(result)
    CALL updatePSW_SF(result)
    CALL updatePSW_OF(reg_file[dst], operand, result)
    CALL updatePSW_CF(result)
    SET reg_file[dst] TO (UNSIGNED SHORT)result
END FUNCTION

# Function to execute a SUBC instruction
FUNCTION executeSUBC(dst: UNSIGNED CHAR, operand: UNSIGNED SHORT)
    SET result TO reg_file[dst] - operand - (1 - psw.CF)
    CALL updatePSW_ZF(result)
    CALL updatePSW_SF(result)
    CALL updatePSW_OF(reg_file[dst], operand, result)
    CALL updatePSW_CF(result)
    SET reg_file[dst] TO (UNSIGNED SHORT)result
```

```
END FUNCTION

# Function to execute a DADD instruction
FUNCTION executeDADD(dst: UNSIGNED CHAR, operand: UNSIGNED SHORT)
    SET result TO reg_file[dst] + operand + psw.CF
    # Decimal addition logic here (if needed)
    CALL updatePSW_ZF(result)
    CALL updatePSW_CF(result)
    SET reg_file[dst] TO result
END FUNCTION

# Function to execute a CMP instruction
FUNCTION executeCMP(dst: UNSIGNED CHAR, operand: UNSIGNED SHORT)
    SET result TO reg_file[dst] - operand
    CALL updatePSW_ZF(result)
    CALL updatePSW_SF(result)
    CALL updatePSW_OF(reg_file[dst], operand, result)
    CALL updatePSW_CF((UNSIGNED INT)reg_file[dst] - operand)
END FUNCTION

# Function to execute an XOR instruction
FUNCTION executeXOR(dst: UNSIGNED CHAR, operand: UNSIGNED SHORT)
    SET result TO reg_file[dst] XOR operand
    CALL updatePSW_ZF(result)
    CALL updatePSW_SF(result)
    SET reg_file[dst] TO result
END FUNCTION

# Function to execute a BIT instruction
FUNCTION executeBIT(dst: UNSIGNED CHAR, operand: UNSIGNED SHORT)
    SET result TO reg_file[dst] AND operand
    CALL updatePSW_ZF(result)
    CALL updatePSW_SF(result)
END FUNCTION

# Function to execute a BIC instruction
FUNCTION executeBIC(dst: UNSIGNED CHAR, operand: UNSIGNED SHORT)
    reg_file[dst] = reg_file[dst] AND NOT operand
    CALL updatePSW_ZF(reg_file[dst])
    CALL updatePSW_SF(reg_file[dst])
END FUNCTION
```

```
# Function to execute a BIS instruction
FUNCTION executeBIS(dst: UNSIGNED CHAR, operand: UNSIGNED SHORT)
    reg_file[dst] = reg_file[dst] OR operand
    CALL updatePSW_ZF(reg_file[dst])
    CALL updatePSW_SF(reg_file[dst])
END FUNCTION

# Function to execute a MOV instruction
FUNCTION executeMOV(dst: UNSIGNED CHAR, operand: UNSIGNED SHORT)
    reg_file[dst] = operand
    CALL updatePSW_ZF(reg_file[dst])
    CALL updatePSW_SF(reg_file[dst])
END FUNCTION

# Function to execute a SWAP instruction
FUNCTION executeSWAP(src: UNSIGNED CHAR, dst: UNSIGNED CHAR)
    value = reg_file[dst]
    reg_file[dst] = reg_file[src]
    reg_file[src] = value
END FUNCTION

# Function to execute a SRA instruction
FUNCTION executeSRA(dst: UNSIGNED CHAR)
    value = reg_file[dst]
    reg_file[dst] = (value SHIFT RIGHT 1) OR (value AND 0x8000)
    CALL updatePSW_ZF(reg_file[dst])
    CALL updatePSW_SF(reg_file[dst])
END FUNCTION

# Function to execute a RRC instruction
FUNCTION executeRRC(dst: UNSIGNED CHAR)
    value = reg_file[dst]
    newCarry = value AND 0x0001
    reg_file[dst] = (value SHIFT RIGHT 1) OR (psw.CF SHIFT LEFT 15)
    psw.CF = newCarry
    CALL updatePSW_ZF(reg_file[dst])
    CALL updatePSW_SF(reg_file[dst])
END FUNCTION

# Function to execute a SWPB instruction
FUNCTION executeSWPB(dst: UNSIGNED CHAR)
    value = reg_file[dst]
```

```
    reg_file[dst] = (value SHIFT LEFT 8) OR (value SHIFT RIGHT 8)
    CALL updatePSW_ZF(reg_file[dst])
    CALL updatePSW_SF(reg_file[dst])
END FUNCTION

# Function to execute a SXT instruction
FUNCTION executeSXT(dst: UNSIGNED CHAR)
    reg_file[dst] = (reg_file[dst] AND 0xFF) OR (IF (reg_file[dst] AND 0x80) THEN 0xFF00 ELSE
0x0000)
    CALL updatePSW_ZF(reg_file[dst])
    CALL updatePSW_SF(reg_file[dst])
END FUNCTION

# Function to execute a MOVLZ instruction
FUNCTION executeMOVLZ(dst: UNSIGNED SHORT, operand: UNSIGNED SHORT)
    reg_file[dst] = operand AND 0x00FF  # Low byte of operand to low byte of destination
END FUNCTION

# Function to execute a MOVL instruction
FUNCTION executeMOVL(dst: UNSIGNED CHAR, operand: UNSIGNED SHORT)
    reg_file[dst] = (reg_file[dst] AND 0xFF00) OR (operand AND 0x00FF)  # Low byte of operand to
low byte of destination
END FUNCTION

# Function to execute a MOVLS instruction
FUNCTION executeMOVLS(dst: UNSIGNED CHAR, operand: UNSIGNED SHORT)
    reg_file[dst] = 0xFF00 OR (operand AND 0x00FF)  # Low byte of operand to low byte of
destination, high byte set to 11111111
END FUNCTION

# Function to execute a MOVH instruction
FUNCTION executeMOVH(dst: UNSIGNED CHAR, operand: UNSIGNED SHORT)
    reg_file[dst] = (operand SHIFT LEFT 8) OR (reg_file[dst] AND 0x00FF)  # High byte of operand to
high byte of destination, low byte unchanged
END FUNCTION

# Function to execute a SETCC instruction
FUNCTION executeSETCC(v: UNSIGNED CHAR, c: UNSIGNED CHAR, slp: UNSIGNED CHAR,
n: UNSIGNED CHAR, z: UNSIGNED CHAR)
    IF c THEN SET psw.CF TO 1
    IF v THEN SET psw.OF TO 1
    IF n THEN SET psw.SF TO 1
```

IF z THEN SET psw.ZF TO 1

Data Dictionary

Filename = String
  * The name of the S-Record file to be loaded
  * Values: Any valid string representing a filename

IMEM = 32768{Unsigned Short}32768
  * Instruction memory
  * Values: Hexadecimal values representing instructions

DMEM = 65536{Unsigned Char}65536
  * Data memory
  * Values: Hexadecimal values representing data

StartAddress = Unsigned Int
  * The starting address for memory display
  * Values: Hexadecimal values from 0x0000 to 0xFFFF

EndAddress = Unsigned Int
  * The ending address for memory display
  * Values: Hexadecimal values from 0x0000 to 0xFFFF

MemoryChoice = Char
  * The memory type to display
  * Values: 'I' for IMEM, 'D' for DMEM

ContinueChoice = Char
  * The user's choice to display another range
  * Values: 'Y' for yes, 'N' for no

Instruction = Unsigned Short
  * An instruction fetched from IMEM
  * Values: Hexadecimal values representing instructions

PC = Unsigned Short
  * Program counter
  * Values: Hexadecimal values from 0x0000 to 0xFFFF

InstructionType = [ADD | ADDC | SUB | SUBC | DADD | CMP | XOR | AND | OR | BIT | BIC | BIS | MOV | SWAP | SRA | RRC | SWPB | SXT | MOVL | MOVLZ | MOVLS | MOVH | INVALID]
  * The type of instruction

RC = Unsigned Char
  * Register control field
  * Values: 0 or 1

WB = Unsigned Char
  * Write-back field
  * Values: 0 or 1

SRC = Unsigned Char
  * Source register field
  * Values: 0 to 7

DST = Unsigned Char
  * Destination register field
  * Values: 0 to 7

CON = Unsigned Char
  * Constant value field
  * Values: 0 to 7

BB = Unsigned Char
  * Byte-level data field
  * Values: 0 to 255

LoadSRecord Function = const char* filename
  * Loads S-Records from a file into IMEM and DMEM

DisplayMemory Function = unsigned char* memory + int start + int end
  * Displays a range of memory

Manager Function = int argc + char* argv[]
  * Manages user input and memory display

CalculateChecksum Function = const char* line + int count + int dataLength
  * Calculates the checksum of an S-Record line

PipelineExecute Function = unsigned short* PC + int display
  * Executes instructions through the pipeline stages

F0Stage Function = unsigned short* PC
  * Fetches the instruction from memory

D0Stage Function = InstructionType* type + unsigned char* rc + unsigned char* wb + unsigned char* src + unsigned char* dst + unsigned char* con + unsigned char* bb + int display + unsigned short* PC + unsigned char* v + unsigned char* c + unsigned char* slp + unsigned char* n + unsigned char* z
  * Decodes the instruction and extracts fields

F1Stage Function = void
  * Fetches the next instruction from memory

E0Stage Function = InstructionType type + unsigned char rc + unsigned char wb + unsigned char src + unsigned char dst + unsigned char con + unsigned char bb + unsigned char* v + unsigned char* c + unsigned char* slp + unsigned char* n + unsigned char* z
  * Executes the decoded instruction

Tick Function = void
  * Advances the clock for the pipeline

StatusPrint Function = unsigned short* PC
  * Prints the current status of the pipeline

ExecuteADD Function = unsigned char dst + unsigned short operand
  * Executes the ADD instruction

ExecuteADDC Function = unsigned char dst + unsigned short operand
  * Executes the ADDC instruction

ExecuteSUB Function = unsigned char dst + unsigned short operand
  * Executes the SUB instruction

ExecuteSUBC Function = unsigned char dst + unsigned short operand
  * Executes the SUBC instruction

ExecuteDADD Function = unsigned char dst + unsigned short operand
  * Executes the DADD instruction

ExecuteCMP Function = unsigned char dst + unsigned short operand
  * Executes the CMP instruction

ExecuteXOR Function = unsigned char dst + unsigned short operand
  * Executes the XOR instruction

ExecuteAND Function = unsigned char dst + unsigned short operand
  * Executes the AND instruction

ExecuteOR Function = unsigned char dst + unsigned short operand
  * Executes the OR instruction

ExecuteBIT Function = unsigned char dst + unsigned short operand
  * Executes the BIT instruction

ExecuteBIC Function = unsigned char dst + unsigned short operand
  * Executes the BIC instruction

ExecuteBIS Function = unsigned char dst + unsigned short operand
  * Executes the BIS instruction

ExecuteMOV Function = unsigned char dst + unsigned short operand
  * Executes the MOV instruction

ExecuteSWPB Function = unsigned char dst
  * Executes the SWPB instruction

ExecuteSRA Function = unsigned char dst
  * Executes the SRA instruction

ExecuteRRC Function = unsigned char dst
  * Executes the RRC instruction

ExecuteSWAP Function = unsigned char src + unsigned char dst
  * Executes the SWAP instruction

ExecuteSXT Function = unsigned char dst
  * Executes the SXT instruction

ExecuteMOVLZ Function = unsigned short dst + unsigned short operand
  * Executes the MOVLZ instruction

ExecuteMOVL Function = unsigned char dst + unsigned short operand
  * Executes the MOVL instruction

ExecuteMOVLS Function = unsigned char dst + unsigned short operand
  * Executes the MOVLS instruction

ExecuteMOVH Function = unsigned char dst + unsigned short operand
  * Executes the MOVH instruction

ExecuteSETCC Function = unsigned char v + unsigned char c + unsigned char slp + unsigned char n + unsigned char z
  * Sets condition codes

ExecuteCLRCC Function = unsigned char v + unsigned char c + unsigned char slp + unsigned char n + unsigned char z
  * Clears condition codes

GetInstructionType Function = unsigned short instruction
  * Determines the type of an instruction

ExtractFields Function = unsigned short instruction + InstructionType type + unsigned char* rc + unsigned char* wb + unsigned char* src + unsigned char* dst + unsigned char* con + unsigned char* bb + unsigned char* v + unsigned char* c + unsigned char* slp + unsigned char* n + unsigned char* z
  * Extracts fields from an instruction

GetInstructionName Function = InstructionType type
  * Gets the name of an instruction type

PrintDecodedInstruction Function = unsigned short PC + InstructionType type + unsigned char rc + unsigned char wb + unsigned char src + unsigned char dst + unsigned char con + unsigned char bb
  * Prints the decoded instruction

GetOperand Function = unsigned char rc + unsigned char src
  * Gets the operand based on the RC field

RunMode Function = int debug
  * Runs the emulator in the specified mode (normal or debug)

DisplayRegisters Function = void
  * Displays the current values of all registers

ChangeRegister Function = void
  * Allows the user to change the value of a register

ChangeMemory Function = void
  * Allows the user to change the value of a memory location

SetBreakpoint Function = unsigned short address
  * Sets a breakpoint at the specified address

SaveRegisterInfoToFile Function = void

* Saves the current register information to a file

Implementation

The code has been provided in a zip file for implementation, as adding the code here would make the document too long.

Testing

Test ADD Instruction

```
; Test_ADD
;
; Student Name: Iftekhar Rafi
; ID: B00871031
;
; ECED 3403 - Computer Architecture

    org    #1000

Main    ; 1. MOVLZ: Load immediate value (lower byte) into a register
    movlz    #03,R0        ; R0 <- 0x0003

    ; 2. MOVL: Load immediate value (lower byte) into a register
    movl    #02,R1        ; R1 <- 0x0002

    ; 3. ADD: Add R0 to R1
    add    R0,R1        ; R1 <- R1 + R0

    ; 4. Halt program execution
    halt

    end    Main        ; End of program
```

Expected results:

R0: 0003
R1: 0005
R7: 1008

No PSW bits set.

Results:

```
Header (ASCII): Test_Add.asm
Header (Bytes): 54 65 73 74 5F 41 64 64 2E 61 73 6D 79
Starting address: 1000

Enter command (I for IMEM, D for DMEM, R to display registers, C to change
 register, M to change memory, B to set breakpoint, E for execute program,
 X for debugger mode): x
Clock  PC   Instruction      Fetch      Decode      Execute    Z N V C
-------------------------------------------------------------------------
  0   1000      6818       F0: 1000    D0: 6800
  1                        F1: 6818                E0: 6818    0 0 0 0
  2   1002      6011       F0: 1002    D0: 6818
  3                        F1: 6011                E0: 6011    0 0 0 0
  4   1004      4001       F0: 1004    D0: 6011
  5                        F1: 4001                E0: 4001    0 0 0 0
  6   1006      0          F0: 1006    D0: 4001
  7                        F1: 0                   E0: 0       0 0 0 0
End of program reached. Execution stopped.

Register Contents:
R0: 0003
R1: 0005
R2: 0000
R3: 0000
R4: 0000
R5: 0000
R6: 0000
R7: 1008

Do you want to enter another command? (Y for yes, N for no):
```

Result: The test is successful. The results are consistent with expectations.

Test ADDC Instruction

Input:

```
; Test_ADDC.asm
;
; Student Name: Iftekhar Rafi
; ID: B00871031
;
; ECED 3403 - Computer Architecture

    org    #1000

Main    ; 1. MOVLZ: Load immediate value (lower byte) into a register
    movlz    #FF,R0        ; R0 <- 0x00FF

    ; 2. MOVH: Load immediate value (higher byte) into a register
    movh    #FF00,R0        ; R0 <- 0xFFFF


    ; 3. MOVL: Load immediate value (lower byte) into a register
    movl    #01,R1        ; R1 <- 0x0001

    ; 4. ADD: Add R0 to R1 to set the carry bit
    add    R0,R1        ; R1 <- R1 + R0

    ; 5. ADDC: Add with carry R0 to R2
    addc    R1,R2        ; R2 <- R2 + R1 + Carry


    ; 4. Halt program execution
    halt

    end    Main        ; End of program
```

Expected results:

R0: FFFF
R1: 0000
R2: 0001
R7: 100C

Z, V and C bit set on executing the ADD instruction.

Results:

```
Enter command (I for IMEM, D for DMEM, R to display registers, C to chang
e register, M to change memory, B to set breakpoint, E for execute progra
m, X for debugger mode): x
Clock  PC   Instruction      Fetch      Decode      Execute    Z N V C
------------------------------------------------------------------------
-
  0   1000      6FF8       F0: 1000    D0: 6800
  1                        F1: 6FF8                E0: 6FF8    0 0 0 0
  2   1002      7FF8       F0: 1002    D0: 6FF8
  3                        F1: 7FF8                E0: 7FF8    0 0 0 0
  4   1004      6009       F0: 1004    D0: 7FF8
  5                        F1: 6009                E0: 6009    0 0 0 0
  6   1006      4001       F0: 1006    D0: 6009
  7                        F1: 4001                E0: 4001    0 0 0 0
  8   1008      410A       F0: 1008    D0: 4001
  9                        F1: 410A                E0: 410A    1 0 1 1
  10  100A      0          F0: 100A    D0: 410A
  11                       F1: 0                   E0: 0       0 0 0 0
End of program reached. Execution stopped.

Register Contents:
R0: FFFF
R1: 0000
R2: 0001
R3: 0000
R4: 0000
R5: 0000
R6: 0000
R7: 100C

Do you want to enter another command? (Y for yes, N for no):
```

Result: The test is successful. The psw bits are set successfully, and register values are consistent with expectations

Test SUB Instruction

```
; Test_SUB
;
; Student Name: Iftekhar Rafi
; ID: B00871031
;
; ECED 3403 - Computer Architecture

    org    #1000

Main    ; 1. MOVLZ: Load immediate value (lower byte) into a register
    movlz    #05,R0        ; R0 <- 0x0005

    ; 2. MOVL: Load immediate value (lower byte) into a register
    movl    #03,R1        ; R1 <- 0x0003

    ; 3. SUB: Subtract R1 from R0 (Normal subtraction)
    sub    R1,R0        ; R0 <- R0 - R1
    ; Check PSW bits after normal subtraction

    ; 4. MOVLZ: Load immediate value (lower byte) into a register
    movlz    #FF,R2        ; R2 <- 0x00FF (255 in decimal)

    ; 5. MOVL: Load immediate value (lower byte) into a register
    movl    #01,R3        ; R3 <- 0x0001

    ; 6. SUB: Subtract R3 from R2 (Subtraction causing borrow)
    sub    R2,R3        ; R3 <- R3 - R2
    ; Check PSW bits after subtraction causing borrow

    ; 7. Halt program execution
    halt

    end    Main        ; End of program
```

Expected Results:

Initial State

- **R0:** 0x0000
- **R1:** 0x0000
- **R2:** 0x0000
- **R3:** 0x0000
- **PSW:** {ZF: 0, SF: 0, OF: 0, CF: 0}

After `movlz #05,R0`

- **R0:** 0x0005
- **PSW:** Unchanged

After `movl #03,R1`

- **R1:** 0x0003
- **PSW:** Unchanged

After `sub R1,R0`

- **R0:** 0x0002 (0x0005 - 0x0003 = 0x0002)
- **PSW:**
    - **ZF:** 0 (Result is not zero)
    - **SF:** 0 (Result is positive)
    - **OF:** 0 (No signed overflow)
    - **CF:** 0 (No borrow)

After `movlz #FF,R2`

- **R2:** 0x00FF
- **PSW:** Unchanged

After `movl #01,R3`

- **R3:** 0x0001
- **PSW:** Unchanged

After `sub R2,R3`

- **R3:** 0xFF02 (0x0001 - 0x00FF = 0xFF02)
- **PSW:**
  - **ZF:** 0 (Result is not zero)
  - **SF:** 1 (Result is negative)
  - **OF:** 0 (No signed overflow)
  - **CF:** 1 (Borrow occurred)

Final Register Contents

- **R0:** 0x0002
- **R1:** 0x0003
- **R2:** 0x00FF
- **R3:** 0xFF02
- **R4:** 0x0000
- **R5:** 0x0000
- **R6:** 0x0000
- **R7:** 0x100C (Final PC value)
- **PSW:** {ZF: 0, SF: 1, OF: 0, CF: 1}

Results

```
Header (ASCII): Test_Sub.asm
Header (Bytes): 54 65 73 74 5F 53 75 62 2E 61 73 6D 58
Starting address: 1000

Enter command (I for IMEM, D for DMEM, R to display registers, C to change
 register, M to change memory, B to set breakpoint, E for execute program,
 X for debugger mode): x
Clock  PC   Instruction      Fetch      Decode      Execute    Z N V C
------------------------------------------------------------------------
  0   1000      6828       F0: 1000    D0: 6800
  1                        F1: 6828                E0: 6828    0 0 0 0
  2   1002      6019       F0: 1002    D0: 6828
  3                        F1: 6019                E0: 6019    0 0 0 0
  4   1004      4208       F0: 1004    D0: 6019
  5                        F1: 4208                E0: 4208    0 0 0 0
  6   1006      6FFA       F0: 1006    D0: 4208
  7                        F1: 6FFA                E0: 6FFA    0 0 0 0
  8   1008      600B       F0: 1008    D0: 6FFA
  9                        F1: 600B                E0: 600B    0 0 0 0
  10  100A      4213       F0: 100A    D0: 600B
  11                       F1: 4213                E0: 4213    0 0 0 0
  12  100C      0          F0: 100C    D0: 4213
  13                       F1: 0                   E0: 0       0 1 1 1
End of program reached. Execution stopped.

Register Contents:
R0: 0002
R1: 0003
R2: 00FF
R3: FF02
R4: 0000
R5: 0000
R6: 0000
R7: 100E

Do you want to enter another command? (Y for yes, N for no):
```

The test is successful and the values are consistent with the expectations.

Test SUBC Instruction

```
; Test_SUBC
;
; Student Name: Iftekhar Rafi
; ID: B00871031
;
; ECED 3403 - Computer Architecture

    org    #1000

Main    ; 1. MOVLZ: Load immediate value (lower byte) into a register
    movlz    #05,R0        ; R0 <- 0x0005

    ; 2. MOVL: Load immediate value (lower byte) into a register
    movl    #03,R1        ; R1 <- 0x0003

    ; 3. SUBC: Subtract R1 from R0 with carry
    subc    R1,R0        ; R0 <- R0 - R1 - Carry
    ; Check PSW bits after normal subtraction

    ; 4. SETCC: Set the carry bit
    setcc    C    ; Set CF

    ; 5. MOVLZ: Load immediate value (lower byte) into a register
    movlz    #FF,R2        ; R2 <- 0x00FF (255 in decimal)

    ; 6. MOVL: Load immediate value (lower byte) into a register
    movl    #01,R3        ; R3 <- 0x0001

    ; 7. SUBC: Subtract R2 from R3 with carry
    subc    R2,R3        ; R3 <- R3 - R2 - Carry
    ; Check PSW bits after subtraction causing borrow

    ; 8. MOVLZ: Load immediate value (lower byte) into a register
    movlz    #80,R4        ; R4 <- 0x0080 (-128 in decimal for signed overflow)

    ; 9. MOVL: Load immediate value (lower byte) into a register
    movl    #01,R5        ; R5 <- 0x0001

    ; 10. SUBC: Subtract R4 from R5 with carry
    subc    R4,R5        ; R5 <- R5 - R4 - Carry
    ; Check PSW bits after subtraction causing overflow

    ; 11. Halt program execution
    halt

    end    Main        ; End of program
```

Expected Results for `Test SUBC.asm`

Initial State

- **R0:** 0x0000
- **R1:** 0x0000
- **R2:** 0x0000
- **R3:** 0x0000
- **R4:** 0x0000
- **R5:** 0x0000
- **PSW:** {ZF: 0, SF: 0, OF: 0, CF: 0}

After `movlz #05,R0`

- **R0:** 0x0005
- **PSW:** Unchanged

After `movl #03,R1`

- **R1:** 0x0003
- **PSW:** Unchanged

After `subc R1,R0` (Normal subtraction with carry)

- **R0:** 0x0002 (0x0005 - 0x0003 - 0)
- **PSW:**
  - **ZF:** 0 (Result is not zero)
  - **SF:** 0 (Result is positive)
  - **OF:** 0 (No signed overflow)
  - **CF:** 0 (No borrow)

After `setcc C` (Set carry bit)

- **PSW:** {ZF: 0, SF: 0, OF: 0, CF: 1}

After `movlz #FF,R2`

- **R2:** 0x00FF
- **PSW:** Unchanged

After `movl #01,R3`

- **R3:** 0x0001
- **PSW:** Unchanged

After `subc R2,R3` (Subtraction causing borrow with carry)

- **R3:** 0xFF01 (0x0001 - 0x00FF - 1)
- **PSW:**
  - **ZF:** 0 (Result is not zero)
  - **SF:** 1 (Result is negative)
  - **OF:** 0 (No signed overflow)
  - **CF:** 1 (Borrow occurred)

After `movlz #80,R4`

- **R4:** 0x0080
- **PSW:** Unchanged

After `movl #01,R5`

- **R5:** 0x0001
- **PSW:** Unchanged

After `subc R4,R5` (Subtraction causing overflow with carry)

- **R5:** 0xFF80 (0x0001 - 0x0080 - 1)
- **PSW:**
  - **ZF:** 0 (Result is not zero)
  - **SF:** 1 (Result is negative)
  - **OF:** 1 (Signed overflow)
  - **CF:** 1 (Borrow occurred)

Final Register Contents

- **R0:** 0x0002
- **R1:** 0x0003
- **R2:** 0x00FF
- **R3:** 0xFF01
- **R4:** 0x0080
- **R5:** 0xFF80
- **R6:** 0x0000
- **R7:** 0x1016 (Final PC value)
- **PSW:** {ZF: 0, SF: 1, OF: 1, CF: 1}

Actual Output for SUBC

```
Enter command (I for IMEM, D for DMEM, R to display registers, C to ch
ange register, M to change memory, B to set breakpoint, E for execute
program, X for debugger mode): x
Clock  PC   Instruction      Fetch      Decode      Execute    Z N V C
------------------------------------------------------------------------
----
  0   1000      6828       F0: 1000    D0: 6800
  1                        F1: 6828                E0: 6800    0 0 0 0
  2   1002      6019       F0: 1002    D0: 6828
  3                        F1: 6019                E0: 6828    0 0 0 0
  4   1004      4308       F0: 1004    D0: 6019
  5                        F1: 4308                E0: 6019    0 0 0 0
  6   1006      4DA1       F0: 1006    D0: 4308
  7                        F1: 4DA1                E0: 4308    0 0 0 0
  8   1008      6FFA       F0: 1008    D0: 4DA1
  9                        F1: 6FFA                E0: 4DA1    0 0 0 1
  10  100A      600B       F0: 100A    D0: 6FFA
  11                       F1: 600B                E0: 6FFA    0 0 0 1
  12  100C      4313       F0: 100C    D0: 600B
  13                       F1: 4313                E0: 600B    0 0 0 1
  14  100E      6C04       F0: 100E    D0: 4313
  15                       F1: 6C04                E0: 4313    0 1 1 1
  16  1010      600D       F0: 1010    D0: 6C04
  17                       F1: 600D                E0: 6C04    0 1 1 1
  18  1012      4325       F0: 1012    D0: 600D
  19                       F1: 4325                E0: 600D    0 1 1 1
  20  1014      0          F0: 1014    D0: 4325
  21                       F1: 0                   E0: 4325    0 1 1 1
End of program reached. Execution stopped.

Register Contents:
R0: 0002
R1: 0003
R2: 00FF
R3: FF01
R4: 0080
R5: FF80
R6: 0000
R7: 1016

Do you want to enter another command? (Y for yes, N for no):
```

The test is successful!

Test DADD Instruction

```
; Test_DADD_Carry
;
; Student Name: Iftekhar Rafi
; ID: B00871031
;
; ECED 3403 - Computer Architecture

    org    #1000

Main    ; 1. MOVLZ: Load immediate value (lower byte) into a register
    movlz    #99,R0        ; R0 <- 0x0099

    ; 2. MOVL: Load immediate value (lower byte) into a register
    movl    #99,R1        ; R1 <- 0x0099

    ; 3. MOVH: Load immediate value (higher byte) into R0 and R1
    movh    #9900,R0      ; R0 <- 0x9999
    movh    #9900,R1      ; R1 <- 0x9999

    ; 4. DADD: Decimal Add R1 to R0 (should set carry bit)
    dadd    R1,R0         ; R0 <- R0 + R1
    ; Check PSW bits after DADD (Decimal addition with carry)

    ; 5. MOVLZ: Load immediate value (lower byte) into a register
    movlz    #45,R2       ; R2 <- 0x0045

    ; 6. MOVL: Load immediate value (lower byte) into a register
    movl    #55,R3        ; R3 <- 0x0055

    ; 7. MOVH: Load immediate value (higher byte) into R2 and R3
    movh    #4500,R2      ; R2 <- 0x4545
    movh    #5500,R3      ; R3 <- 0x5555

    ; 8. DADD: Decimal Add R3 to R2 using the carry bit set by previous DADD
    dadd    R3,R2         ; R2 <- R2 + R3 + Carry
    ; Check PSW bits after DADD (Decimal addition with carry from previous DADD)

    ; 9. Halt program execution
    halt

    end    Main        ; End of program
```

Assembly Code and Expected Results:

1. **MOVLZ #99,R0**:
    - R0 <- 0x0099
    - **Expected R0:** 0x0099
2. **MOVL #99,R1**:
    - R1 <- 0x0099
    - **Expected R1:** 0x0099
3. **MOVH #9900,R0**:
    - R0 <- 0x9999 (high byte of R0 set to 0x99)
    - **Expected R0:** 0x9999
4. **MOVH #9900,R1**:
    - R1 <- 0x9999 (high byte of R1 set to 0x99)
    - **Expected R1:** 0x9999
5. **DADD R1,R0**:
    - R0 <- 0x9999 + 0x9999 (BCD addition, should set carry bit)
    - **Expected R0:** 0x3332 (0x13332, with carry bit set)
    - **Expected Carry Bit:** 1
6. **MOVLZ #45,R2**:
    - R2 <- 0x0045
    - **Expected R2:** 0x0045
7. **MOVL #55,R3**:
    - R3 <- 0x0055
    - **Expected R3:** 0x0055
8. **MOVH #4500,R2**:
    - R2 <- 0x4545 (high byte of R2 set to 0x45)
    - **Expected R2:** 0x4545
9. **MOVH #5500,R3**:
    - R3 <- 0x5555 (high byte of R3 set to 0x55)
    - **Expected R3:** 0x5555
10. **DADD R3,R2**:
    - R2 <- 0x4545 + 0x5555 + carry (BCD addition, uses carry bit from previous DADD)
    - **Expected R2:** 0x9A9B (BCD addition with carry)
    - **Expected Carry Bit:** 1

Results:

```
Header (ASCII): Test_DADD.asm
Header (Bytes): 54 65 73 74 5F 44 41 44 44 2E 61 73 6D 74
Starting address: 1000

Enter command (I for IMEM, D for DMEM, R to display registers, C to change
 register, M to change memory, B to set breakpoint, E for execute program,
 X for debugger mode): x
Clock  PC   Instruction      Fetch      Decode      Execute    Z N V C
-------------------------------------------------------------------------
  0   1000      6CC8       F0: 1000    D0: 6800
  1                        F1: 6CC8                E0: 6800    0 0 0 0
  2   1002      64C9       F0: 1002    D0: 6CC8
  3                        F1: 64C9                E0: 6CC8    0 0 0 0
  4   1004      7CC8       F0: 1004    D0: 64C9
  5                        F1: 7CC8                E0: 64C9    0 0 0 0
  6   1006      7CC9       F0: 1006    D0: 7CC8
  7                        F1: 7CC9                E0: 7CC8    0 0 0 0
  8   1008      4408       F0: 1008    D0: 7CC9
  9                        F1: 4408                E0: 7CC9    0 0 0 0
  10  100A      6A2A       F0: 100A    D0: 4408
  11                       F1: 6A2A                E0: 4408    0 0 0 1
  12  100C      62AB       F0: 100C    D0: 6A2A
  13                       F1: 62AB                E0: 6A2A    0 0 0 1
  14  100E      7A2A       F0: 100E    D0: 62AB
  15                       F1: 7A2A                E0: 62AB    0 0 0 1
  16  1010      7AAB       F0: 1010    D0: 7A2A
  17                       F1: 7AAB                E0: 7A2A    0 0 0 1
  18  1012      441A       F0: 1012    D0: 7AAB
  19                       F1: 441A                E0: 7AAB    0 0 0 1
  20  1014      0          F0: 1014    D0: 441A
  21                       F1: 0                   E0: 441A    0 0 0 0
End of program reached. Execution stopped.

Register Contents:
R0: 3332
R1: 9999
R2: 9A9B
R3: 5555
R4: 0000
R5: 0000
R6: 0000
R7: 1016

Do you want to enter another command? (Y for yes, N for no):
```

The test is successful!

Test CMP Instruction

```
; Test_CMP
;
; Student Name: Iftekhar Rafi
; ID: B00871031
;
; ECED 3403 - Computer Architecture

    org    #1000

Main    ; 1. MOVLZ: Load immediate value (lower byte) into a register
    movlz    #05,R0        ; R0 <- 0x0005

    ; 2. MOVL: Load immediate value (lower byte) into a register
    movl    #05,R1        ; R1 <- 0x0005

    ; 3. CMP: Compare R0 with R1 (should be equal, Z flag set)
    cmp    R1,R0        ; Compare R0 with R1

    ; 4. MOVLZ: Load immediate value (lower byte) into a register
    movlz    #FF,R2        ; R2 <- 0x00FF (255 in decimal)

    ; 5. MOVL: Load immediate value (lower byte) into a register
    movl    #01,R3        ; R3 <- 0x0001

    ; 6. CMP: Compare R2 with R3 (R2 > R3, N flag set)
    cmp    R3,R2        ; Compare R2 with R3

    ; 7. MOVLZ: Load immediate value (lower byte) into a register
    movlz    #01,R4        ; R4 <- 0x0001

    ; 8. MOVL: Load immediate value (lower byte) into a register
    movl    #02,R5        ; R5 <- 0x0002

    ; 9. CMP: Compare R4 with R5 (R4 < R5, no flags set)
    cmp    R5,R4        ; Compare R4 with R5

    ; 10. Halt program execution
    halt

    end    Main        ; End of program
```

Expected Results:

1. **MOVLZ #05,R0**:
    - R0 <- 0x0005
    - **Expected R0:** 0x0005
2. **MOVL #05,R1**:
    - R1 <- 0x0005
    - **Expected R1:** 0x0005
3. **CMP R1,R0**:
    - Compare R0 with R1 (both are 0x0005)
    - **Expected Flags:** Z = 1 (because they are equal), N = 0, V = 0, C = 0
4. **MOVLZ #FF,R2**:
    - R2 <- 0x00FF (255 in decimal)
    - **Expected R2:** 0x00FF
5. **MOVL #01,R3**:
    - R3 <- 0x0001
    - **Expected R3:** 0x0001
6. **CMP R3,R2**:
    - Compare R2 (0x00FF) with R3 (0x0001)
    - **Expected Flags:** Z = 0, N = 1 (R2 > R3), V = 1, C = 1
7. **MOVLZ #01,R4**:
    - R4 <- 0x0001
    - **Expected R4:** 0x0001
8. **MOVL #02,R5**:
    - R5 <- 0x0002
    - **Expected R5:** 0x0002
9. **CMP R5,R4**:
    - Compare R4 (0x0001) with R5 (0x0002)
    - **Expected Flags:** Z = 0, N = 0 (R4 < R5), V = 0, C = 0

Results:

```
Header (ASCII): Test_CMP.asm
Header (Bytes): 54 65 73 74 5F 43 4D 50 2E 61 73 6D A2
Starting address: 1000

Enter command (I for IMEM, D for DMEM, R to display registers, C to change regis
ter, M to change memory, B to set breakpoint, E for execute program, X for debug
ger mode): x
Clock  PC   Instruction      Fetch      Decode      Execute    Z N V C
------------------------------------------------------------------------
  0   1000      6828       F0: 1000    D0: 6800
  1                        F1: 6828                E0: 6800    0 0 0 0
  2   1002      6029       F0: 1002    D0: 6828
  3                        F1: 6029                E0: 6828    0 0 0 0
  4   1004      4508       F0: 1004    D0: 6029
  5                        F1: 4508                E0: 6029    0 0 0 0
  6   1006      6FFA       F0: 1006    D0: 4508
  7                        F1: 6FFA                E0: 4508    1 0 0 0
  8   1008      600B       F0: 1008    D0: 6FFA
  9                        F1: 600B                E0: 6FFA    1 0 0 0
  10  100A      4513       F0: 100A    D0: 600B
  11                       F1: 4513                E0: 600B    1 0 0 0
  12  100C      680C       F0: 100C    D0: 4513
  13                       F1: 680C                E0: 4513    0 1 1 1
  14  100E      6015       F0: 100E    D0: 680C
  15                       F1: 6015                E0: 680C    0 1 1 1
  16  1010      4525       F0: 1010    D0: 6015
  17                       F1: 4525                E0: 6015    0 1 1 1
  18  1012      0          F0: 1012    D0: 4525
  19                       F1: 0                   E0: 4525    0 0 0 0
End of program reached. Execution stopped.

Register Contents:
R0: 0005
R1: 0005
R2: 00FF
R3: 0001
R4: 0001
R5: 0002
R6: 0000
R7: 1014

Do you want to enter another command? (Y for yes, N for no):
```

The test is successful!

Test XOR Instruction

```
; Test_XOR
;
; Student Name: Iftekhar Rafi
; ID: B00871031
;
; ECED 3403 - Computer Architecture

    org    #1000

Main    ; 1. MOVLZ: Load immediate value (lower byte) into a register
    movlz    #AA,R0        ; R0 <- 0x00AA

    ; 2. MOVL: Load immediate value (lower byte) into a register
    movl    #55,R1        ; R1 <- 0x0055

    ; 3. XOR: XOR R0 and R1
    xor    #55,R0        ; R0 <- R0 ^ #55

    ; 4. Halt program execution
    halt

    end    Main        ; End of program
```

Expected Results:

1. **PC 1000**:
    - **Instruction**: 0x6878
    - **Fetch**: F0: 1000
    - **Decode**: D0: 6800
    - **Execute**: E0: 6800
    - **ZNVC**: 0 0 0 0
2. **PC 1002**:
    - **Instruction**: 0x6781
    - **Fetch**: F0: 1002
    - **Decode**: D0: 6878
    - **Execute**: E0: 6878
    - **ZNVC**: 0 0 0 0
3. **PC 1004**:
    - **Instruction**: 0x4601
    - **Fetch**: F0: 1004
    - **Decode**: D0: 6781
    - **Execute**: E0: 6781
    - **ZNVC**: 0 0 0 0
4. **PC 1006**:
    - **Instruction**: 0x6D52
    - **Fetch**: F0: 1006
    - **Decode**: D0: 4601
    - **Execute**: E0: 4601
    - **ZNVC**: 0 0 0 0
5. **PC 1008**:
    - **Instruction**: 0x4612
    - **Fetch**: F0: 1008
    - **Decode**: D0: 6D52
    - **Execute**: E0: 6D52
    - **ZNVC**: 0 0 0 0
6. **PC 100A**:
    - **Instruction**: 0x6AAB
    - **Fetch**: F0: 100A
    - **Decode**: D0: 4612
    - **Execute**: E0: 4612
    - **ZNVC**: 1 0 0 0
7. **PC 100C**:
    - **Instruction**: 0x6554
    - **Fetch**: F0: 100C
    - **Decode**: D0: 6AAB
    - **Execute**: E0: 6AAB
    - **ZNVC**: 1 0 0 0
8. **PC 100E**:
    - **Instruction**: 0x461C
    - **Fetch**: F0: 100E
    - **Decode**: D0: 6554
    - **Execute**: E0: 6554

    - **ZNVC**: 1 0 0 0
9. **PC 1010**:
    - **Instruction**: 0x0
    - **Fetch**: F0: 1010
    - **Decode**: D0: 461C
    - **Execute**: E0: 461C
    - **ZNVC**: 0 0 0 0

Register Contents:

- **R0**: 000F
- **R1**: 00FF
- **R2**: 0000
- **R3**: 0055
- **R4**: 00FF
- **R5**: 0000
- **R6**: 0000
- **R7**: 1012

ZNVC bits:

- **PC 1006**: Z: 1, N: 0, V: 0, C: 0
- **PC 100A**: Z: 1, N: 0, V: 0, C: 0
- **PC 100C**: Z: 1, N: 0, V: 0, C: 0

Results:

```
Header (ASCII): Test_XOR.asm
Header (Bytes): 54 65 73 74 5F 58 4F 52 2E 61 73 6D 89
Starting address: 1000

Enter command (I for IMEM, D for DMEM, R to display registers, C to change
 register, M to change memory, B to set breakpoint, E for execute program,
 X for debugger mode): x
Clock  PC   Instruction      Fetch      Decode      Execute    Z N V C
-------------------------------------------------------------------------
  0   1000      6878       F0: 1000    D0: 6800
  1                        F1: 6878                E0: 6800    0 0 0 0
  2   1002      6781       F0: 1002    D0: 6878
  3                        F1: 6781                E0: 6878    0 0 0 0
  4   1004      4601       F0: 1004    D0: 6781
  5                        F1: 4601                E0: 6781    0 0 0 0
  6   1006      6D52       F0: 1006    D0: 4601
  7                        F1: 6D52                E0: 4601    0 0 0 0
  8   1008      4612       F0: 1008    D0: 6D52
  9                        F1: 4612                E0: 6D52    0 0 0 0
  10  100A      6AAB       F0: 100A    D0: 4612
  11                       F1: 6AAB                E0: 4612    1 0 0 0
  12  100C      6554       F0: 100C    D0: 6AAB
  13                       F1: 6554                E0: 6AAB    1 0 0 0
  14  100E      461C       F0: 100E    D0: 6554
  15                       F1: 461C                E0: 6554    1 0 0 0
  16  1010      0          F0: 1010    D0: 461C
  17                       F1: 0                   E0: 461C    0 0 0 0
End of program reached. Execution stopped.

Register Contents:
R0: 000F
R1: 00FF
R2: 0000
R3: 0055
R4: 00FF
R5: 0000
R6: 0000
R7: 1012

Do you want to enter another command? (Y for yes, N for no):
```

The test is successful!

Test AND Instruction

```
; Test_AND.asm
;
; Student Name: Iftekhar Rafi
; ID: B00871031
;
; ECED 3403 - Computer Architecture

    org    #1000

Main    ; 1. MOVLZ: Load immediate value (lower byte) into a register
    movlz    #AA,R0        ; R0 <- 0x00AA

    ; 2. MOVL: Load immediate value (lower byte) into a register
    movlz    #55,R1        ; R1 <- 0x0055

    ; 3. AND: AND R0 and R1
    and    R1,R0        ; R0 <- R0 & R1

    ; 4. Halt program execution
    halt

    end    Main        ; End of program
```

Expected Results

1. **Instruction: MOVLZ #0F, R0**
    - **Action**: Load immediate value 0x0F into the low byte of R0.
    - **Register R0**: 0x000F
    - **PSW**: Z = 0, N = 0, V = 0, C = 0
2. **Instruction: MOVL #F0, R1**
    - **Action**: Load immediate value 0xF0 into the low byte of R1.
    - **Register R1**: 0x00F0
    - **PSW**: Z = 0, N = 0, V = 0, C = 0
3. **Instruction: AND R1, R0**
    - **Action**: Perform bitwise AND between R0 (0x000F) and R1 (0x00F0). Result = 0x0000.
    - **Register R0**: 0x0000
    - **PSW**: Z = 1, N = 0, V = 0, C = 0
4. **Instruction: MOVLZ #FF, R2**
    - **Action**: Load immediate value 0xFF into the low byte of R2.
    - **Register R2**: 0x00FF
    - **PSW**: Z = 1, N = 0, V = 0, C = 0
5. **Instruction: MOVL #0F, R3**
    - **Action**: Load immediate value 0x0F into the low byte of R3.
    - **Register R3**: 0x000F
    - **PSW**: Z = 1, N = 0, V = 0, C = 0
6. **Instruction: AND R3, R2**
    - **Action**: Perform bitwise AND between R2 (0x00FF) and R3 (0x000F). Result = 0x000F.
    - **Register R2**: 0x000F
    - **PSW**: Z = 0, N = 0, V = 0, C = 0
7. **Instruction: MOVLZ #AA, R4**
    - **Action**: Load immediate value 0xAA into the low byte of R4.
    - **Register R4**: 0x00AA
    - **PSW**: Z = 0, N = 0, V = 0, C = 0
8. **Instruction: MOVL #55, R5**
    - **Action**: Load immediate value 0x55 into the low byte of R5.
    - **Register R5**: 0x0055
    - **PSW**: Z = 0, N = 0, V = 0, C = 0
9. **Instruction: AND R5, R4**
    - **Action**: Perform bitwise AND between R4 (0x00AA) and R5 (0x0055). Result = 0x0000.
    - **Register R4**: 0x0000
    - **PSW**: Z = 1, N = 0, V = 0, C = 0

Summary of Final Register Values

- **R0**: 0x0000
- **R1**: 0x00F0
- **R2**: 0x000F
- **R3**: 0x000F
- **R4**: 0x0000
- **R5**: 0x0055
- **R6**: 0x0000
- **R7**: 0x1014

Summary of Final PSW Values

- **Z**: 1
- **N**: 0
- **V**: 0
- **C**: 0

Results:

```
Header (ASCII): Test_AND.asm
Header (Bytes): 54 65 73 74 5F 41 4E 44 2E 61 73 6D AF
Starting address: 1000

Enter command (I for IMEM, D for DMEM, R to display registers, C to chan
ge register, M to change memory, B to set breakpoint, E for execute prog
ram, X for debugger mode): x
Clock  PC   Instruction      Fetch      Decode      Execute    Z N V C
------------------------------------------------------------------------
--
  0   1000      6878       F0: 1000    D0: 6800
  1                        F1: 6878                E0: 6800    0 0 0 0
  2   1002      6781       F0: 1002    D0: 6878
  3                        F1: 6781                E0: 6878    0 0 0 0
  4   1004      4708       F0: 1004    D0: 6781
  5                        F1: 4708                E0: 6781    0 0 0 0
  6   1006      6FFA       F0: 1006    D0: 4708
  7                        F1: 6FFA                E0: 4708    1 0 0 0
  8   1008      607B       F0: 1008    D0: 6FFA
  9                        F1: 607B                E0: 6FFA    1 0 0 0
  10  100A      471A       F0: 100A    D0: 607B
  11                       F1: 471A                E0: 607B    1 0 0 0
  12  100C      6D54       F0: 100C    D0: 471A
  13                       F1: 6D54                E0: 471A    0 0 0 0
  14  100E      62AD       F0: 100E    D0: 6D54
  15                       F1: 62AD                E0: 6D54    0 0 0 0
  16  1010      472C       F0: 1010    D0: 62AD
  17                       F1: 472C                E0: 62AD    0 0 0 0
  18  1012      0          F0: 1012    D0: 472C
  19                       F1: 0                   E0: 472C    1 0 0 0
End of program reached. Execution stopped.

Register Contents:
R0: 0000
R1: 00F0
R2: 000F
R3: 000F
R4: 0000
R5: 0055
R6: 0000
R7: 1014

Do you want to enter another command? (Y for yes, N for no):
```

The test is successful!

Test OR Instruction

```
; Test_OR
;
; Student Name: Iftekhar Rafi
; ID: B00871031
;
; ECED 3403 - Computer Architecture

    org    #1000

Main    ; 1. MOVLZ: Load immediate value (lower byte) into a register
    movlz    #00,R0        ; R0 <- 0x0000

    ; 2. MOVL: Load immediate value (lower byte) into a register
    movl    #00,R1        ; R1 <- 0x0000

    ; 3. OR: OR R1 with R0 (should set ZF since the result is zero)
    or    R1,R0        ; R0 <- R0 | R1
    ; Check PSW bits after OR

    ; 4. MOVH: Load immediate value (higher byte) into a register
    movh    #8000,R2        ; R2 <- 0x8000

    ; 5. OR: OR R2 with R0 (should set SF since the result has a high bit set)
    or    R2,R0        ; R0 <- R0 | R2
    ; Check PSW bits after OR

    ; 6. MOVLZ: Load immediate value (lower byte) into a register
    movlz    #7F,R3        ; R3 <- 0x007F

    ; 7. OR: OR R3 with R0 (should keep SF set)
    or    R3,R0        ; R0 <- R0 | R3
    ; Check PSW bits after OR

    ; 8. MOVLZ: Load immediate value (lower byte) into a register
    movlz    #00,R4        ; R4 <- 0x0000

    ; 9. OR: OR R4 with R0 (should keep SF set)
    or    R4,R0        ; R0 <- R0 | R4
    ; Check PSW bits after OR

    ; 10. Halt program execution
    halt

    end    Main        ; End of program
```

Expected Results:

1. **After step 1:**
   - **Instruction:** `movlz #00,R0`
   - **Registers:** `R0 = 0x0000`
   - **PSW:** No change.
2. **After step 2:**
   - **Instruction:** `movl #00,R1`
   - **Registers:** `R1 = 0x0000`
   - **PSW:** No change.
3. **After step 3:**
   - **Instruction:** `or R1,R0`
   - **Registers:** `R0 = 0x0000`
   - **PSW:** `ZF = 1, SF = 0, CF = 0, OF = 0.`
4. **After step 4:**
   - **Instruction:** `movh #8000,R2`
   - **Registers:** `R2 = 0x8000`
   - **PSW:** No change.
5. **After step 5:**
   - **Instruction:** `or R2,R0`
   - **Registers:** `R0 = 0x8000`
   - **PSW:** `ZF = 0, SF = 1, CF = 0, OF = 0.`
6. **After step 6:**
   - **Instruction:** `movlz #7F,R3`
   - **Registers:** `R3 = 0x007F`
   - **PSW:** No change.
7. **After step 7:**
   - **Instruction:** `or R3,R0`
   - **Registers:** `R0 = 0x807F`
   - **PSW:** `ZF = 0, SF = 1, CF = 0, OF = 0.`
8. **After step 8:**
   - **Instruction:** `movlz #00,R4`
   - **Registers:** `R4 = 0x0000`
   - **PSW:** No change.
9. **After step 9:**
   - **Instruction:** `or R4,R0`
   - **Registers:** `R0 = 0x807F`
   - **PSW:** `ZF = 0, SF = 1, CF = 0, OF = 0.`

Results:

```
Header (ASCII): Test_OR.asm
Header (Bytes): 54 65 73 74 5F 4F 52 2E 61 73 6D E2
Starting address: 1000

Enter command (I for IMEM, D for DMEM, R to display registers, C to chang
e register, M to change memory, B to set breakpoint, E for execute progra
m, X for debugger mode): x
Clock  PC   Instruction      Fetch      Decode      Execute    Z N V C
------------------------------------------------------------------------
-
  0   1000      6800       F0: 1000    D0: 6800
  1                        F1: 6800                E0: 6800    0 0 0 0
  2   1002      6001       F0: 1002    D0: 6800
  3                        F1: 6001                E0: 6800    0 0 0 0
  4   1004      4808       F0: 1004    D0: 6001
  5                        F1: 4808                E0: 6001    0 0 0 0
  6   1006      7C02       F0: 1006    D0: 4808
  7                        F1: 7C02                E0: 4808    1 0 0 0
  8   1008      4810       F0: 1008    D0: 7C02
  9                        F1: 4810                E0: 7C02    1 0 0 0
  10  100A      6BFB       F0: 100A    D0: 4810
  11                       F1: 6BFB                E0: 4810    0 1 0 0
  12  100C      4818       F0: 100C    D0: 6BFB
  13                       F1: 4818                E0: 6BFB    0 1 0 0
  14  100E      6804       F0: 100E    D0: 4818
  15                       F1: 6804                E0: 4818    0 1 0 0
  16  1010      4820       F0: 1010    D0: 6804
  17                       F1: 4820                E0: 6804    0 1 0 0
  18  1012      0          F0: 1012    D0: 4820
  19                       F1: 0                   E0: 4820    0 1 0 0
End of program reached. Execution stopped.

Register Contents:
R0: 807F
R1: 0000
R2: 8000
R3: 007F
R4: 0000
R5: 0000
R6: 0000
R7: 1014

Do you want to enter another command? (Y for yes, N for no):
```

The test is successful!

Test BIT Instruction

```
; Test_BIT
;
; Student Name: Iftekhar Rafi
; ID: B00871031
;
; ECED 3403 - Computer Architecture

    org    #1000

Main    ; 1. MOVLZ: Load immediate value (lower byte) into a register
    movlz    #F0,R0        ; R0 <- 0x00F0

    ; 2. MOVL: Load immediate value (lower byte) into a register
    movl    #0F,R1        ; R1 <- 0x000F

    ; 3. BIT: Test bits in R0 with a constant
    bit    #0,R0        ; Test bits in R0 with #0

    ; 4. Halt program execution
    halt

    end    Main        ; End of program
```

Instructions Breakdown:

1. **Instruction:** `movlz #F0,R0`
   - **Operation:** `R0 = 0x00F0`
   - **Expected Register State:** `R0 = 0x00F0`
   - **PSW:** No change.
2. **Instruction:** `movl #01,R1`
   - **Operation:** `R1 = 0x0001`
   - **Expected Register State:** `R1 = 0x0001`
   - **PSW:** No change.
3. **Instruction:** `bit R1,R0`
   - **Operation:** `R0 & (1 << R1)`
   - **Calculations:**
     - `R1 = 0x0001` -> `1 << R1 = 1 << 1 = 0x0002`
     - `R0 = 0x00F0`
     - `0x00F0 & 0x0002 = 0x0000`
   - **Expected PSW:**
     - `ZF = 1` (since the result is zero)
     - `SF = 0` (no negative result)
4. **Instruction:** `halt`
   - **Expected Register State:** No change.
   - **PSW:** No change.

Expected Output:

1. **Registers:**
   - `R0 = 0x00F0`
   - `R1 = 0x0001`
   - `R2 = 0x0000`
   - `R3 = 0x0000`
   - `R4 = 0x0000`
   - `R5 = 0x0000`
   - `R6 = 0x0000`
   - `R7 = 0x1004` (PC after halt)
2. **PSW (after BIT instruction):**
   - `ZF = 1` (result is zero)
   - `SF = 0` (result is not negative)
   - `OF = 0` (no overflow in bit test)
   - `CF = 0` (carry flag not affected by bit test)

Results:

```
Header (ASCII): Test_BIT.asm
Header (Bytes): 54 65 73 74 5F 42 49 54 2E 61 73 6D A3
Starting address: 1000

Enter command (I for IMEM, D for DMEM, R to display registers, C to change
 register, M to change memory, B to set breakpoint, E for execute program,
 X for debugger mode):
x
Clock  PC   Instruction      Fetch      Decode      Execute    Z N V C
-------------------------------------------------------------------------
  0   1000      6F80       F0: 1000    D0: 6800
  1                        F1: 6F80                E0: 6800    0 0 0 0
  2   1002      6009       F0: 1002    D0: 6F80
  3                        F1: 6009                E0: 6F80    0 0 0 0
  4   1004      4908       F0: 1004    D0: 6009
  5                        F1: 4908                E0: 6009    0 0 0 0
  6   1006      0          F0: 1006    D0: 4908
  7                        F1: 0                   E0: 4908    1 0 0 0
End of program reached. Execution stopped.

Register Contents:
R0: 00F0
R1: 0001
R2: 0000
R3: 0000
R4: 0000
R5: 0000
R6: 0000
R7: 1008

Do you want to enter another command? (Y for yes, N for no):
```

The test is successful!

Test BIC Instruction

```
; Test_BIC
;
; Student Name: Iftekhar Rafi
; ID: B00871031
;
; ECED 3403 - Computer Architecture

    org    #1000

Main    ; 1. MOVLZ: Load immediate value (lower byte) into a register
    movlz    #FF,R0        ; R0 <- 0x00FF

    ; 2. MOVL: Load immediate value (lower byte) into a register
    movl    #F0,R1        ; R1 <- 0x00F0

    ; 3. BIC: Clear bits of R0 in R1 (should clear lower nibble)
    bic    R1,R0        ; R0 <- R0 & ~R1

    ; 4. MOVLZ: Load immediate value (lower byte) into a register
    movlz    #0F,R2        ; R2 <- 0x000F

    ; 5. MOVL: Load immediate value (lower byte) into a register
    movl    #F0,R3        ; R3 <- 0x00F0

    ; 6. BIC: Clear bits of R2 in R3 (should clear upper nibble)
    bic    R2,R3        ; R3 <- R3 & ~R2

    ; 7. MOVLZ: Load immediate value (lower byte) into a register
    movlz    #80,R4        ; R4 <- 0x0080

    ; 8. BIC: Clear bits of R4 in R0 (should clear bit 7)
    bic    R4,R0        ; R0 <- R0 & ~R4

    ; 9. Halt program execution
    halt

    end    Main        ; End of program
```

Expected Results:

1. **Instruction:** `movlz #FF,R0`
   - **Operation:** `R0 = 0x00FF`
   - **Expected Register State:** `R0 = 0x00FF`
   - **PSW:** No change.
2. **Instruction:** `movl #F0,R1`
   - **Operation:** `R1 = 0x00F0`
   - **Expected Register State:** `R1 = 0x00F0`
   - **PSW:** No change.
3. **Instruction:** `bic R1,R0`
   - **Operation:** `R0 = R0 & ~R1`
   - **Calculations:**
     - `R0 = 0x00FF`
     - `R1 = 0x00F0`
     - `R0 = 0x00FF & ~0x00F0 = 0x00FF & 0xFF0F = 0x000F`
   - **Expected Register State:** `R0 = 0x000F`
   - **PSW:** `ZF = 0`, `SF = 0` (result is not zero and not negative).
4. **Instruction:** `movlz #0F,R2`
   - **Operation:** `R2 = 0x000F`
   - **Expected Register State:** `R2 = 0x000F`
   - **PSW:** No change.
5. **Instruction:** `movl #F0,R3`
   - **Operation:** `R3 = 0x00F0`
   - **Expected Register State:** `R3 = 0x00F0`
   - **PSW:** No change.
6. **Instruction:** `bic R2,R3`
   - **Operation:** `R3 = R3 & ~R2`
   - **Calculations:**
     - `R3 = 0x00F0`
     - `R2 = 0x000F`
     - `R3 = 0x00F0 & ~0x000F = 0x00F0 & 0xFFF0 = 0x00F0`
   - **Expected Register State:** `R3 = 0x00F0`
   - **PSW:** `ZF = 0`, `SF = 0` (result is not zero and not negative).
7. **Instruction:** `movlz #80,R4`
   - **Operation:** `R4 = 0x0080`
   - **Expected Register State:** `R4 = 0x0080`
   - **PSW:** No change.
8. **Instruction:** `bic R4,R0`
   - **Operation:** `R0 = R0 & ~R4`
   - **Calculations:**
     - `R0 = 0x000F`
     - `R4 = 0x0080`
     - `R0 = 0x000F & ~0x0080 = 0x000F & 0xFF7F = 0x000F`
   - **Expected Register State:** `R0 = 0x000F`
   - **PSW:** `ZF = 0`, `SF = 0` (result is not zero and not negative).
9. **Instruction:** `halt`
   - **Expected Register State:** No change.
   - **PSW:** No change.

Results:

```
Header (ASCII): Test_BIC.asm
Header (Bytes): 54 65 73 74 5F 42 49 43 2E 61 73 6D B4
Starting address: 1000

Enter command (I for IMEM, D for DMEM, R to display registers, C to change reg
ister, M to change memory, B to set breakpoint, E for execute program, X for d
ebugger mode): x
Clock  PC   Instruction      Fetch      Decode      Execute    Z N V C
------------------------------------------------------------------------
  0   1000      6FF8       F0: 1000    D0: 6800
  1                        F1: 6FF8                E0: 6800    0 0 0 0
  2   1002      6781       F0: 1002    D0: 6FF8
  3                        F1: 6781                E0: 6FF8    0 0 0 0
  4   1004      4A08       F0: 1004    D0: 6781
  5                        F1: 4A08                E0: 6781    0 0 0 0
  6   1006      687A       F0: 1006    D0: 4A08
  7                        F1: 687A                E0: 4A08    0 0 0 0
  8   1008      6783       F0: 1008    D0: 687A
  9                        F1: 6783                E0: 687A    0 0 0 0
  10  100A      4A13       F0: 100A    D0: 6783
  11                       F1: 4A13                E0: 6783    0 0 0 0
  12  100C      6C04       F0: 100C    D0: 4A13
  13                       F1: 6C04                E0: 4A13    0 0 0 0
  14  100E      4A20       F0: 100E    D0: 6C04
  15                       F1: 4A20                E0: 6C04    0 0 0 0
  16  1010      0          F0: 1010    D0: 4A20
  17                       F1: 0                   E0: 4A20    0 0 0 0
End of program reached. Execution stopped.

Register Contents:
R0: 000F
R1: 00F0
R2: 000F
R3: 00F0
R4: 0080
R5: 0000
R6: 0000
R7: 1012

Do you want to enter another command? (Y for yes, N for no):
```

The test is successful!

Test BIS Instruction

```
; Test_BIS
;
; Student Name: Iftekhar Rafi
; ID: B00871031
;
; ECED 3403 - Computer Architecture

    org    #1000

Main    ; 1. MOVLZ: Load immediate value (lower byte) into a register
    movlz    #00,R0        ; R0 <- 0x0000

    ; 2. MOVL: Load immediate value (lower byte) into a register
    movl    #F0,R1        ; R1 <- 0x00F0

    ; 3. BIS: OR R1 with R0 (should set ZF, as R0 was 0)
    bis    R1,R0        ; R0 <- R0 | R1 (R0 = 0x00F0)

    ; 4. MOVLZ: Load immediate value (lower byte) into a register
    movlz    #80,R2        ; R2 <- 0x0080 (128 in decimal)

    ; 5. MOVL: Load immediate value (lower byte) into a register
    movl    #01,R3        ; R3 <- 0x0001

    ; 6. BIS: OR R2 with R3 (should set SF, as result will be 0x0081)
    bis    R2,R3        ; R3 <- R3 | R2 (R3 = 0x0081)

    ; 7. BIS: OR #FFFF with R4 (should set SF, result 0xFFFF)
    movl    #FF,R4        ; R4 <- 0x00FF
    movh    #FF00,R4      ; R4 <- 0xFFFF
    bis    #FFFF,R4        ; R4 <- R4 | 0xFFFF

    ; 8. Halt program execution
    halt

    end    Main        ; End of program
```

Expected Results for BIS Instruction Test

Instruction: `movlz #00,R0`

- **Operation:** R0 = 0x0000
- **Expected Register State:**
  - R0 = 0x0000

Instruction: `movl #F0,R1`

- **Operation:** R1 = 0x00F0
- **Expected Register State:**
  - R1 = 0x00F0

Instruction: `movlz #01,R5`

- **Operation:** R5 = 0x0001
- **Expected Register State:**
  - R5 = 0x0001

Instruction: `bis R5,R0`

- **Operation:** R0 = R0 | (1 << 0)
- **Calculations:**
  - R0 = 0x0000 | (1 << 0) = 0x0001
- **Expected Register State:**
  - R0 = 0x0001
- **Zero Flag (ZF):** 0

Instruction: `movlz #80,R2`

- **Operation:** R2 = 0x0080
- **Expected Register State:**
  - R2 = 0x0080

Instruction: `movl #01,R3`

- **Operation:** R3 = 0x0001
- **Expected Register State:**
  - R3 = 0x0001

Instruction: `movlz #07,R5`

- **Operation:** R5 = 0x0007
- **Expected Register State:**

- R5 = 0x0007

Instruction: `bis R5,R3`

- **Operation:** R3 = R3 | (1 << 7)
- **Calculations:**
  - R3 = 0x0001 | (1 << 7) = 0x0081
- **Expected Register State:**
  - R3 = 0x0081
- **Sign Flag (SF):** 0

Instruction: `movlz #0F,R5`

- **Operation:** R5 = 0x000F
- **Expected Register State:**
  - R5 = 0x000F

Instruction: `movl #00,R4`

- **Operation:** R4 = 0x0000
- **Expected Register State:**
  - R4 = 0x0000

Instruction: `bis R5,R4`

- **Operation:** R4 = R4 | (1 << 15)
- **Calculations:**
  - R4 = 0x0000 | (1 << 15) = 0x8000
- **Expected Register State:**
  - R4 = 0x8000
- **Sign Flag (SF):** 1

Final Register State:

- **R0:** 0x0001
- **R1:** 0x00F0
- **R2:** 0x0080
- **R3:** 0x0081
- **R4:** 0x8000
- **R5:** 0x0000
- **R6:** 0x0000
- **R7:** 0x1018

Final PSW Flags:

- **Zero Flag (ZF):** 0

- **Sign Flag (SF):** 1
- **Overflow Flag (OF):** 0
- **Carry Flag (CF):** 0

Results:

```
Header (ASCII): Test_BIS.asm
Header (Bytes): 54 65 73 74 5F 42 49 53 2E 61 73 6D A4
Starting address: 1000

Enter command (I for IMEM, D for DMEM, R to display registers, C to change regis
ter, M to change memory, B to set breakpoint, E for execute program, X for debug
ger mode):
x
Clock  PC   Instruction      Fetch      Decode      Execute    Z N V C
-------------------------------------------------------------------------
  0   1000      6800       F0: 1000    D0: 6800
  1                        F1: 6800                E0: 6800    0 0 0 0
  2   1002      6781       F0: 1002    D0: 6800
  3                        F1: 6781                E0: 6800    0 0 0 0
  4   1004      4B08       F0: 1004    D0: 6781
  5                        F1: 4B08                E0: 6781    0 0 0 0
  6   1006      6C02       F0: 1006    D0: 4B08
  7                        F1: 6C02                E0: 4B08    0 0 0 0
  8   1008      600B       F0: 1008    D0: 6C02
  9                        F1: 600B                E0: 6C02    0 0 0 0
  10  100A      4B13       F0: 100A    D0: 600B
  11                       F1: 4B13                E0: 600B    0 0 0 0
  12  100C      67FC       F0: 100C    D0: 4B13
  13                       F1: 67FC                E0: 4B13    0 0 0 0
  14  100E      7FFC       F0: 100E    D0: 67FC
  15                       F1: 7FFC                E0: 67FC    0 0 0 0
  16  1010      4BBC       F0: 1010    D0: 7FFC
  17                       F1: 4BBC                E0: 7FFC    0 0 0 0
  18  1012      0          F0: 1012    D0: 4BBC
  19                       F1: 0                   E0: 4BBC    0 1 0 0
End of program reached. Execution stopped.

Register Contents:
R0: 00F0
R1: 00F0
R2: 0080
R3: 0081
R4: FFFF
R5: 0000
R6: 0000
R7: 1014

Do you want to enter another command? (Y for yes, N for no):
```

The test is successful!

Test SWAP Instruction

```
; Test_SWAP
;
; Student Name: Iftekhar Rafi
; ID: B00871031
;
; ECED 3403 - Computer Architecture

    org    #1000

Main    ; 1. MOVLZ: Load immediate value (lower byte) into a register
    movlz    #F0,R0        ; R0 <- 0x00F0

    ; 2. MOVL: Load immediate value (lower byte) into a register
    movl    #0F,R1        ; R1 <- 0x000F

    ; 3. SWAP: Swap bytes between R0 and R1
    swap    R0,R1        ; Swap bytes of R0 and R1

    ; 4. Halt program execution
    halt

    end    Main        ; End of program
```

Expected Results

Instruction Execution:

1. **Instruction:** `movlz #12,R0`
    - **Operation:** R0 = 0x0012
    - **Expected Register State:** R0 = 0x0012
2. **Instruction:** `movlz #34,R1`
    - **Operation:** R1 = 0x0034
    - **Expected Register State:** R1 = 0x0034
3. **Instruction:** `swap R0,R1`
    - **Operation:** Swap values of R0 and R1
    - **Expected Register State:**
        - R0 = 0x0034
        - R1 = 0x0012
4. **Instruction:** `movlz #56,R2`
    - **Operation:** R2 = 0x0038
    - **Expected Register State:** R2 = 0x0038
5. **Instruction:** `movlz #78,R3`
    - **Operation:** R3 = 0x0078
    - **Expected Register State:** R3 = 0x0078
6. **Instruction:** `swap R2,R3`
    - **Operation:** Swap values of R2 and R3
    - **Expected Register State:**
        - R2 = 0x0078
        - R3 = 0x0038
7. **Instruction:** `halt`
    - **Operation:** Halt the program
    - **Expected Register State:** No change to registers
    - **Expected PSW State:** No change to PSW

Summary of Expected Register State:

- R0: 0x0034
- R1: 0x0012
- R2: 0x0078
- R3: 0x0038
- R4: 0x0000 (no change)
- R5: 0x0000 (no change)
- R6: 0x0000 (no change)
- R7: 0x100E (program counter points to the address after `halt`)

Results:

```
Enter command (I for IMEM, D for DMEM, R to display registers, C to change
register, M to change memory, B to set breakpoint, E for execute program, X
 for debugger mode): x
Clock  PC   Instruction      Fetch      Decode      Execute    Z N V C
------------------------------------------------------------------------
  0   1000      6890       F0: 1000    D0: 6800
  1                        F1: 6890                E0: 6800    0 0 0 0
  2   1002      69A1       F0: 1002    D0: 6890
  3                        F1: 69A1                E0: 6890    0 0 0 0
  4   1004      4C81       F0: 1004    D0: 69A1
  5                        F1: 4C81                E0: 69A1    0 0 0 0
  6   1006      6AB2       F0: 1006    D0: 4C81
  7                        F1: 6AB2                E0: 4C81    0 0 0 0
  8   1008      6BC3       F0: 1008    D0: 6AB2
  9                        F1: 6BC3                E0: 6AB2    0 0 0 0
  10  100A      4C93       F0: 100A    D0: 6BC3
  11                       F1: 4C93                E0: 6BC3    0 0 0 0
  12  100C      0          F0: 100C    D0: 4C93
  13                       F1: 0                   E0: 4C93    0 0 0 0
End of program reached. Execution stopped.

Register Contents:
R0: 0034
R1: 0012
R2: 0078
R3: 0056
R4: 0000
R5: 0000
R6: 0000
R7: 100E

Do you want to enter another command? (Y for yes, N for no):
```

The test is successful!

Test SRA Instruction

```
; Test_SRA
;
; Student Name: Iftekhar Rafi
; ID: B00871031
;
; ECED 3403 - Computer Architecture

    org    #1000

Main    ; 1. MOVLZ: Load immediate value (lower byte) into a register
    movlz    #80,R0        ; R0 <- 0x0080 (128 in decimal, 0b10000000)

    ; 2. SRA: Shift R0 right arithmetically (should preserve the sign bit)
    sra    R0              ; R0 >> 1

    ; 3. MOVLZ: Load immediate value (lower byte) into a register
    movlz    #01,R1        ; R1 <- 0x0001 (1 in decimal, 0b00000001)

    ; 4. MOVH: Load immediate value (higher byte) into R1
    movh    #FF01,R1       ; R1 <- 0xFF01 (-255 in decimal, 0b1111111100000001)

    ; 5. SRA: Shift R1 right arithmetically (should preserve the sign bit)
    sra    R1              ; R1 >> 1

    ; 6. MOVLZ: Load immediate value (lower byte) into a register
    movlz    #01,R2        ; R2 <- 0x0001 (1 in decimal, 0b00000001)

    ; 7. MOVH: Load immediate value (higher byte) into R2
    movh    #7F01,R2       ; R2 <- 0x7F01 (32513 in decimal, 0b0111111100000001)

    ; 8. SRA: Shift R2 right arithmetically (should not preserve the sign bit)
    sra    R2              ; R2 >> 1

    ; 9. Halt program execution
    halt

    end    Main            ; End of program
```

Expected Results

Instruction Execution:

1. **Instruction:** `movlz #80,R0`
   - **Operation:** R0 = 0x0080
   - **Expected Register State:** R0 = 0x0080
2. **Instruction:** `sra R0`
   - **Operation:** R0 = 0x0080 >> 1 = 0x0040 (binary: 0b10000000 -> 0b01000000)
   - **Expected Register State:** R0 = 0x0040
3. **Instruction:** `movlz #01,R1`
   - **Operation:** R1 = 0x0001
   - **Expected Register State:** R1 = 0x0001
4. **Instruction:** `movh #FF01,R1`
   - **Operation:** R1 = 0xFF01
   - **Expected Register State:** R1 = 0xFF01
5. **Instruction:** `sra R1`
   - **Operation:** R1 = 0xFF01 >> 1 = 0xFF80 (binary: 0b1111111100000001 -> 0b1111111110000000, preserving sign bit)
   - **Expected Register State:** R1 = 0xFF80
6. **Instruction:** `movlz #01,R2`
   - **Operation:** R2 = 0x0001
   - **Expected Register State:** R2 = 0x0001
7. **Instruction:** `movh #7F01,R2`
   - **Operation:** R2 = 0x7F01
   - **Expected Register State:** R2 = 0x7F01
8. **Instruction:** `sra R2`
   - **Operation:** R2 = 0x7F01 >> 1 = 0x3F80 (binary: 0b0111111100000001 -> 0b0011111110000000, not preserving sign bit)
   - **Expected Register State:** R2 = 0x3F80
9. **Instruction:** `halt`
   - **Operation:** Halt the program
   - **Expected Register State:** No change to registers
   - **Expected PSW State:** No change to PSW

Summary of Expected Register State:

- R0: 0x0040
- R1: 0xFF80
- R2: 0x3F80
- R3: 0x0000 (no change)
- R4: 0x0000 (no change)
- R5: 0x0000 (no change)
- R6: 0x0000 (no change)
- R7: 0x1012 (program counter points to the address after `halt`)

Results:

```
Header (ASCII): Test_SRA.asm
Header (Bytes): 54 65 73 74 5F 53 52 41 2E 61 73 6D 9C
Starting address: 1000

Enter command (I for IMEM, D for DMEM, R to display registers, C to change
register, M to change memory, B to set breakpoint, E for execute program, X
 for debugger mode): x
Clock  PC   Instruction      Fetch      Decode      Execute    Z N V C
-------------------------------------------------------------------------
  0   1000      6C00       F0: 1000    D0: 6800
  1                        F1: 6C00                E0: 6800    0 0 0 0
  2   1002      4D00       F0: 1002    D0: 6C00
  3                        F1: 4D00                E0: 6C00    0 0 0 0
  4   1004      6809       F0: 1004    D0: 4D00
  5                        F1: 6809                E0: 4D00    0 0 0 0
  6   1006      7FF9       F0: 1006    D0: 6809
  7                        F1: 7FF9                E0: 6809    0 0 0 0
  8   1008      4D01       F0: 1008    D0: 7FF9
  9                        F1: 4D01                E0: 7FF9    0 0 0 0
  10  100A      680A       F0: 100A    D0: 4D01
  11                       F1: 680A                E0: 4D01    0 1 0 0
  12  100C      7BFA       F0: 100C    D0: 680A
  13                       F1: 7BFA                E0: 680A    0 1 0 0
  14  100E      4D02       F0: 100E    D0: 7BFA
  15                       F1: 4D02                E0: 7BFA    0 1 0 0
  16  1010      0          F0: 1010    D0: 4D02
  17                       F1: 0                   E0: 4D02    0 0 0 0
End of program reached. Execution stopped.

Register Contents:
R0: 0040
R1: FF80
R2: 3F80
R3: 0000
R4: 0000
R5: 0000
R6: 0000
R7: 1012

Do you want to enter another command? (Y for yes, N for no):
```

The test is successful!

Test RRC Instruction

```
; Test_RRC
;
; Student Name: Iftekhar Rafi
; ID: B00871031
;
; ECED 3403 - Computer Architecture

    org    #1000

Main    ; 1. MOVLZ: Load immediate value (lower byte) into a register
    movlz    #F0,R0        ; R0 <- 0x00F0

    ; 2. RRC: Rotate right through carry
    rrc    R0            ; R0 <- Rotate R0 right through carry

    ; 3. Halt program execution
    halt

    end    Main        ; End of program
```

Expected Results

1. **Instruction:** `movlz #80,R0`
    - **Operation:** R0 = 0x0080
    - **Expected Register State:** R0 = 0x0080
2. **Instruction:** `rrc R0`
    - **Operation:** Rotate 0x0080 right through carry (initial CF=0):
        - Original: 0x0080 -> Binary: 0000 1000 0000 0000
        - Rotated: 0x0040, CF set to 0
    - **Expected Register State:** R0 = 0x0040, CF = 0
3. **Instruction:** `movlz #01,R1`
    - **Operation:** R1 = 0x0001
    - **Expected Register State:** R1 = 0x0001
4. **Instruction:** `movh #FF01,R1`
    - **Operation:** R1 = 0xFF01
    - **Expected Register State:** R1 = 0xFF01
5. **Instruction:** `rrc R1`
    - **Operation:** Rotate 0xFF01 right through carry (CF=0):
        - Original: 0xFF01 -> Binary: 1111 1111 0000 0001
        - Rotated: 0x7F80, CF set to 1
    - **Expected Register State:** R1 = 0x7F80, CF = 1
6. **Instruction:** `movlz #01,R2`
    - **Operation:** R2 = 0x0001
    - **Expected Register State:** R2 = 0x0001
7. **Instruction:** `movh #7F01,R2`
    - **Operation:** R2 = 0x7F01
    - **Expected Register State:** R2 = 0x7F01
8. **Instruction:** `rrc R2`
    - **Operation:** Rotate 0x7F01 right through carry (CF=1):
        - Original: 0x7F01 -> Binary: 0111 1111 0000 0001
        - Rotated: 0xBF80, CF set to 1
    - **Expected Register State:** R2 = 0xBF80, CF = 1

Summary of Register State

- **R0:** 0x0040
- **R1:** 0x7F80
- **R2:** 0xBF80
- **R3, R4, R5, R6, R7:** No change expected, matches 0x0000 and 0x1012 respectively.

PSW State Check

- **Zero Flag (ZF):** 0 (none of the operations resulted in zero)
- **Negative Flag (NF):** 1 (R1 and R2 became negative after RRC)
- **Overflow Flag (VF):** 0 (no overflow expected in rotation)
- **Carry Flag (CF):** 1 (after R1 and R2 RRC)

Results:

```
Header (ASCII): Test_RRC.asm
Header (Bytes): 54 65 73 74 5F 52 52 43 2E 61 73 6D 9B
Starting address: 1000

Enter command (I for IMEM, D for DMEM, R to display registers, C to change
register, M to change memory, B to set breakpoint, E for execute program, X
 for debugger mode): x
Clock  PC   Instruction      Fetch      Decode      Execute    Z N V C
-------------------------------------------------------------------------
  0   1000      6C00       F0: 1000    D0: 6800
  1                        F1: 6C00                E0: 6800    0 0 0 0
  2   1002      4D08       F0: 1002    D0: 6C00
  3                        F1: 4D08                E0: 6C00    0 0 0 0
  4   1004      6809       F0: 1004    D0: 4D08
  5                        F1: 6809                E0: 4D08    0 0 0 0
  6   1006      7FF9       F0: 1006    D0: 6809
  7                        F1: 7FF9                E0: 6809    0 0 0 0
  8   1008      4D09       F0: 1008    D0: 7FF9
  9                        F1: 4D09                E0: 7FF9    0 0 0 0
  10  100A      680A       F0: 100A    D0: 4D09
  11                       F1: 680A                E0: 4D09    0 0 0 1
  12  100C      7BFA       F0: 100C    D0: 680A
  13                       F1: 7BFA                E0: 680A    0 0 0 1
  14  100E      4D0A       F0: 100E    D0: 7BFA
  15                       F1: 4D0A                E0: 7BFA    0 0 0 1
  16  1010      0          F0: 1010    D0: 4D0A
  17                       F1: 0                   E0: 4D0A    0 1 0 1
End of program reached. Execution stopped.

Register Contents:
R0: 0040
R1: 7F80
R2: BF80
R3: 0000
R4: 0000
R5: 0000
R6: 0000
R7: 1012

Do you want to enter another command? (Y for yes, N for no):
```

The test is successful!

Test SWPB Instruction

```
; Test_SWPB
;
; Student Name: Iftekhar Rafi
; ID: B00871031
;
; ECED 3403 - Computer Architecture

    org    #1000

Main
    ; 1. MOVLZ: Load immediate value (lower byte) into a register
    movlz    #12,R0         ; R0 <- 0x0012

    ; 2. MOVH: Load immediate value (higher byte) into a register
    movh    #3400,R0        ; R0 <- 0x3412

    ; 3. SWPB: Swap bytes in R0
    swpb    R0              ; R0 <- 0x1234

    ; 4. MOVLZ: Load immediate value (lower byte) into a register
    movlz    #56,R1         ; R1 <- 0x0056

    ; 5. MOVH: Load immediate value (higher byte) into a register
    movh    #7800,R1        ; R1 <- 0x7856

    ; 6. SWPB: Swap bytes in R1
    swpb    R1              ; R1 <- 0x3878

    ; 7. Halt program execution
    halt

    end    Main             ; End of program
```

Expected Results:

1. **Instruction:** `movlz #12,R0`
    - **Operation:** R0 = 0x0012
    - **Expected Register State:** R0 = 0x0012
2. **Instruction:** `movh #3400,R0`
    - **Operation:** R0 = 0x3412
    - **Expected Register State:** R0 = 0x3412
3. **Instruction:** `swpb R0`
    - **Operation:** Swap bytes in R0 (0x3412):
        - Original: 0x3412 -> Swap bytes: 0x1234
    - **Expected Register State:** R0 = 0x1234
4. **Instruction:** `movlz #56,R1`
    - **Operation:** R1 = 0x0056
    - **Expected Register State:** R1 = 0x0038
5. **Instruction:** `movh #7800,R1`
    - **Operation:** R1 = 0x7856
    - **Expected Register State:** R1 = 0x7856
6. **Instruction:** `swpb R1`
    - **Operation:** Swap bytes in R1 (0x7856):
        - Original: 0x7856 -> Swap bytes: 0x5678
    - **Expected Register State:** R1 = 0x5678

Results:

```
Header (ASCII): Test_SWPB.asm
Header (Bytes): 54 65 73 74 5F 53 57 50 42 2E 61 73 6D 45
Starting address: 1000

Enter command (I for IMEM, D for DMEM, R to display registers, C to chan
ge register, M to change memory, B to set breakpoint, E for execute prog
ram, X for debugger mode): x
Clock  PC   Instruction      Fetch      Decode      Execute    Z N V C
------------------------------------------------------------------------
--
  0   1000      6890       F0: 1000    D0: 6800
  1                        F1: 6890                E0: 6800    0 0 0 0
  2   1002      79A0       F0: 1002    D0: 6890
  3                        F1: 79A0                E0: 6890    0 0 0 0
  4   1004      4D18       F0: 1004    D0: 79A0
  5                        F1: 4D18                E0: 79A0    0 0 0 0
  6   1006      6AB1       F0: 1006    D0: 4D18
  7                        F1: 6AB1                E0: 4D18    0 0 0 0
  8   1008      7BC1       F0: 1008    D0: 6AB1
  9                        F1: 7BC1                E0: 6AB1    0 0 0 0
  10  100A      4D19       F0: 100A    D0: 7BC1
  11                       F1: 4D19                E0: 7BC1    0 0 0 0
  12  100C      0          F0: 100C    D0: 4D19
  13                       F1: 0                   E0: 4D19    0 0 0 0
End of program reached. Execution stopped.

Register Contents:
R0: 1234
R1: 5678
R2: 0000
R3: 0000
R4: 0000
R5: 0000
R6: 0000
R7: 100E

Do you want to enter another command? (Y for yes, N for no):
```

The test is successful!

Test SXT Instruction

```
; Test_SXT
;
; Student Name: Iftekhar Rafi
; ID: B00871031
;
; ECED 3403 - Computer Architecture

    org    #1000

Main    ; 1. MOVLZ: Load immediate value (lower byte) into a register
    movlz    #80,R0        ; R0 <- 0x0080 (-128 in decimal, signed byte)

    ; 2. MOVLZ: Load immediate value (lower byte) into a register
    movlz    #7F,R1        ; R1 <- 0x007F (127 in decimal, signed byte)

    ; 3. SXT: Sign-extend byte to word
    sxt    R0          ; R0 <- Sign-extend R0 (R0 = 0xFF80)

    ; 4. SXT: Sign-extend byte to word
    sxt    R1          ; R1 <- Sign-extend R1 (R1 = 0x007F)

    ; 5. MOVLZ: Load immediate value (lower byte) into a register
    movlz    #00,R2        ; R2 <- 0x0000 (0 in decimal, signed byte)

    ; 6. SXT: Sign-extend byte to word
    sxt    R2          ; R2 <- Sign-extend R2 (R2 = 0x0000)

    ; 7. Halt program execution
    halt

    end    Main        ; End of program
```

Expected Results:

1. `movlz #80,R0`
    - Operation: R0 = 0x0080
    - Expected Register State: R0 = 0x0080
2. `movlz #7F,R1`
    - Operation: R1 = 0x007F
    - Expected Register State: R1 = 0x007F
3. `sxt R0`
    - Operation: R0 = 0xFF80 (Sign-extend 0x80 to 0xFF80)
    - Expected Register State: R0 = 0xFF80
4. `sxt R1`
    - Operation: R1 = 0x007F (Sign-extend 0x7F to 0x007F)
    - Expected Register State: R1 = 0x007F
5. `movlz #00,R2`
    - Operation: R2 = 0x0000
    - Expected Register State: R2 = 0x0000
6. `sxt R2`
    - Operation: R2 = 0x0000 (Sign-extend 0x00 to 0x0000)
    - Expected Register State: R2 = 0x0000

Results:

```
Header (ASCII): Test_SXT.asm
Header (Bytes): 54 65 73 74 5F 53 58 54 2E 61 73 6D 83
Starting address: 1000

Enter command (I for IMEM, D for DMEM, R to display registers, C to change
register, M to change memory, B to set breakpoint, E for execute program, X
 for debugger mode): x
Clock  PC   Instruction      Fetch      Decode      Execute    Z N V C
-------------------------------------------------------------------------
  0   1000      6C00       F0: 1000    D0: 6800
  1                        F1: 6C00                E0: 6800    0 0 0 0
  2   1002      6BF9       F0: 1002    D0: 6C00
  3                        F1: 6BF9                E0: 6C00    0 0 0 0
  4   1004      4D20       F0: 1004    D0: 6BF9
  5                        F1: 4D20                E0: 6BF9    0 0 0 0
  6   1006      4D21       F0: 1006    D0: 4D20
  7                        F1: 4D21                E0: 4D20    0 1 0 0
  8   1008      6802       F0: 1008    D0: 4D21
  9                        F1: 6802                E0: 4D21    0 0 0 0
  10  100A      4D22       F0: 100A    D0: 6802
  11                       F1: 4D22                E0: 6802    0 0 0 0
  12  100C      0          F0: 100C    D0: 4D22
  13                       F1: 0                   E0: 4D22    1 0 0 0
End of program reached. Execution stopped.

Register Contents:
R0: FF80
R1: 007F
R2: 0000
R3: 0000
R4: 0000
R5: 0000
R6: 0000
R7: 100E

Do you want to enter another command? (Y for yes, N for no):
```

The test is successful!