ARAB ACADEMY FOR SCIENCE & TECHNOLOGY & MARITIME TRANSPORT
COLLEGE OF ENGINEERING & TECHNOLOGY

Course	**CC410 System Programming**
Lecturer	Dr. Sherine Nagy Saleh – Dr. Noha Seddik Abdelsalam
TA	Eng. Esraa Khatab - Mohamed Ramzy - Eng. Esraa Ismail - Eng. Hoda Elkhodery

modi-SIC ONE-PASS ASSEMBLER PROJECT

Project Overview:

The Modi-SIC One-Pass Assembler Project (Type 2) is designed to create a program that performs one-pass assembly for the Modified Simple Instruction Computer (modi-SIC). The assembler will translate assembly code into machine code compatible with the Modified Simplified Instructional Computer.

Introduction to Modi-SIC:

The modi-SIC architecture retains the instruction set and Format 3 instructions from the original SIC (Simple Instruction Computer). It also includes the concept of reserving variables in memory using directives such as BYTE, WORD, RESB, and RESW.

Key Modifications and Extensions:

- Format 1 Instructions: Modi-SIC extends its capabilities by introducing Format 1 instructions, expanding the range of supported operations.
- *Immediate Instructions (Format 3):* Modi-SIC introduces immediate instructions, allowing for the handling of immediate values passed as integers. This provides greater flexibility in executing instructions.
- *Relocation:* The Modi-SIC also supports relocation by using the masking bits in the text records, where 1 denotes memory location that needs modification and 0 otherwise

Instructions Handling:

The assembler will process the assembly code in a single pass, generating machine code (HTE records) for execution on the modi-SIC. It will consider both the original SIC Format 3 instructions and the newly introduced Format 1 instructions and immediate instructions.

Project Scope:

This project aims to streamline the assembly process for modi-SIC, providing an efficient tool for converting assembly code into executable machine code in a single pass. **The one-pass assembler will read the assembly code, allocate memory, and generate the corresponding relocatable machine code for execution on modi-SIC.** It will handle symbol resolution, directive processing, and object code generation (HTE) in a single pass.

Project Deliverables:

1- Modi-SIC One-Pass Assembler Program: A fully functional program capable of converting assembly code into machine code in a single pass.
2- Documentation: Comprehensive documentation outlining the one-pass assembly process, key algorithms used, instructions handling, design issues and sample runs.

List of Modi-SIC Instructions:

Below is a list of modi-SIC instructions along with a new bit introduced to indicate whether the instruction deals with an immediate value:

Mnemonic	Format	Opcode	Effect
ADD m	3/4	18	A <-- (A) + (m..m+2)
AND m	3/4	40	A <-- (A) & (m..m+2)
COMP m	3/4	28	A : (m..m+2)
DIV m	3/4	24	A : (A) / (m..m+2)
J m	3/4	3C	PC <-- m
JEQ m	3/4	30	PC <-- m if CC set to =
JGT m	3/4	34	PC <-- m if CC set to >
JLT m	3/4	38	PC <-- m if CC set to <
JSUB m	3/4	48	L <-- (PC); PC <-- m
LDA m	3/4	00	A <-- (m..m+2)
LDCH m	3/4	50	A [rightmost byte] <-- (m)
LDL m	3/4	08	L <-- (m..m+2)
LDX m	3/4	04	X <-- (m..m+2)
MUL m	3/4	20	A <-- (A) * (m..m+2)
OR m	3/4	44	A <-- (A) | (m..m+2)
RD m	3/4	D8	A [rightmost byte] <-- data
RSUB	3/4	4C	PC <-- (L)
STA m	3/4	0C	m..m+2 <-- (A)
STCH m	3/4	54	m <-- (A) [rightmost byte]
STL m	3/4	14	m..m+2 <-- (L)
STSW m	3/4	E8	m..m+2 <-- (SW)
STX m	3/4	10	m..m+2 <-- (X)
SUB m	3/4	1C	A <-- (A) - (m..m+2)
TD m	3/4	E0	Test device specified by (m)
TIX m	3/4	2C	X <-- (X) + 1; (X) : (m..m+2)
WD m	3/4	DC	Device specified by (m) <-- (A)[rightmost byte]
FIX	1	C4	A <- (F) [Convert to integer]
FLOAT	1	C0	F <- (A) [Convert to floating]
HIO	1	F4	Halt I/O channel number (A)
NORM	1	C8	F <- (F) [normalized]
SIO	1	F0	Start I/O channel number (A); address of channel program is given by (S)
TIO	1	F8	Test I/O channel number (A)

New instruction formats and types:

- **instruction format 1 in *modi-SIC***

Opcode (8 bits)

- **Immediate Instruction format in *modi-SIC***

  All Type 3 instruction could be immediate instructions this is done by a new division of bits of instructions of Type 3 (Format 3) as shawn in following table.

Opcode (7 bits)	Immediate flag (i) (1 bit)	Indexing (x) (I bit)	Address (15 bits)

  The modification applied on the opcode as

  1. Only opcode is represented as 7 bits (not 8) as in SIC
  2. The 8th bit of the opcode represents the immediate flag (i) which has two value
     a. 0 if the instruction without immediate value (has an address)
     b. 1 if the instruction with immediate value

***Second: modi-SIC* one-pass assembler implementation details**

It takes as an input a text file (in.txt) that contains modi-SIC assembly program.
Remember that The *modi-SIC program* includes Format 1 instruction of SIC/XE. So this case must be handled.

Generating Symbol Table File

You will have to read the assembly program and generate a symbol table file (symbolTable.txt) for all the symbols extracted from the program.

Generating HTE records:

You will have to generate the complete HTE records (objectcode.txt). It must contain one header, one or more text records (including masking bits) and one end record.

Your code should support showing both symbol table and THE records should be available at any given line in the input file

Assessment

A maximum of 3 students per group is allowed, each team member will be assessed for his individual work as well as for the group work. Both members should agree on the programming languages and code structures to be used. The table below shows the tasks required from each team member.

Report **Overall code structure and output**		
Student 1	**Student 2**	**Student 3**
Input Parsing	Input Parsing	Input parsing
Location counter	Location counter	Location counter
Format 1	Format 3	Handling memory reservations (BYTE, WORD, RESW, RESB)
Symbol Table handling	Masking bit	HTE records generation

Each team should submit the merged source code and executable files and each student should submit the individual source code, executable files, and a report that includes:

- example statements on how to use your program (both individual and merged). E.g. for a python program: python3 code.py --data in.txt
- Data structures used and design issues
- sample run with input and output and error analysis

Academic Integrity Statement:

The development and submission of the Modi-SIC One-Pass Assembler Project adhere strictly to principles of academic integrity. **The code, documentation, and any associated materials will be submitted as a comprehensive PDF document through Turnitin.** This submission is made with the understanding that it will undergo plagiarism detection, and any instances of unauthorized collaboration or academic dishonesty will be subject to the institution's policies and procedures. We affirm our commitment to the ethical conduct of academic work, acknowledging that the originality and authenticity of this project are paramount.