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 DIASSEMBLER PROJECT

Write a program to simulate a Modified Simple Instruction Computer *(modi-SIC) diassembler*. A modi-SIC disassembler is a program that translates machine code (HTE record) into an assembly code for the Modified Simplified Instructional Computer (modi-SIC).

First: Lets introduce the modi-SIC assembler to understand the modification occurred to the instructions and instruction set:

The *modi-SIC* consists of

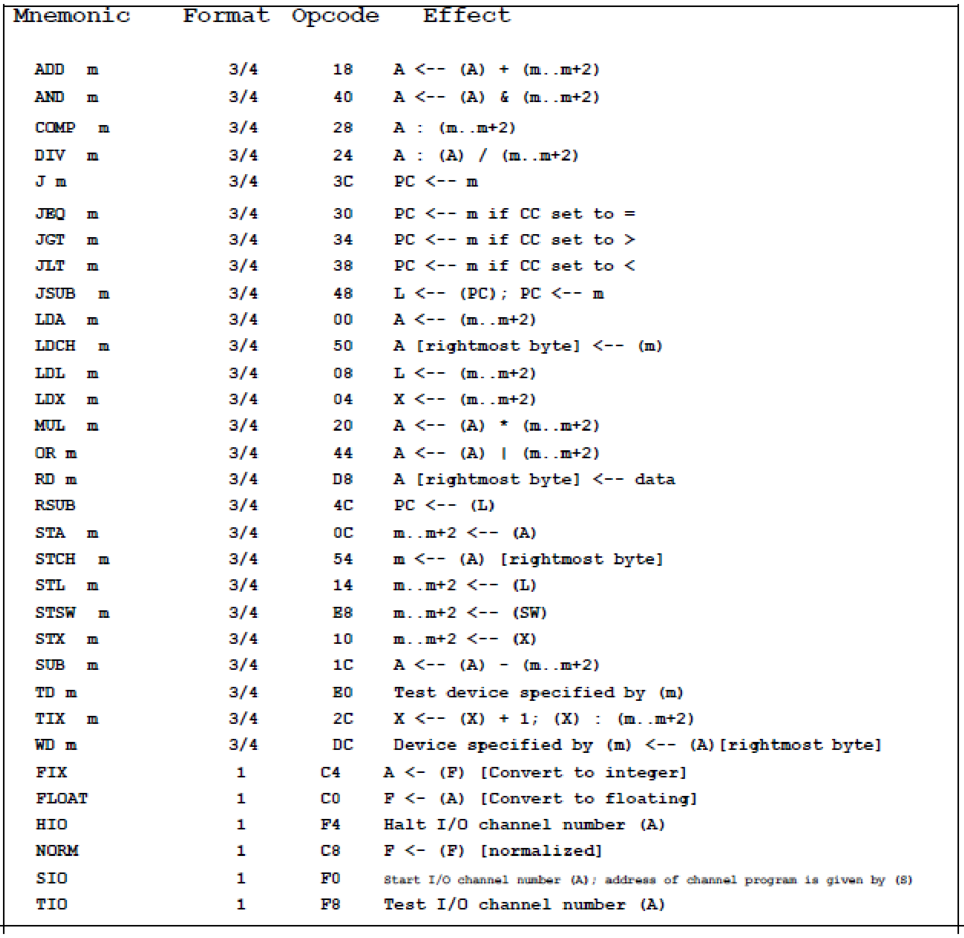
1. Same instructions set (Format 3) of SIC
2. Same idea of reservation of variables in memory using BYTE, WORD, RESB, RESW

***modi-SIC* is extened to include**

1. Format 1 instuctions
2. Immediate Instruction (Format 3) that deals with an immediate value passed to as integer.

For the sack of simplification a list of *modi-SIC* instructions is attached at the end of project description that have all instructions handled by *modi-SIC* + a full description of a new introduced bit that states if the instruction is dealing with immediate or not.

FULL INSTRUCTION SET OF *modi-SIC*



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 Diassembler implementation details

It takes as an input a text file (in.txt) that contains modi-SIC machine code (modi-SIC HTE record). Remember that The *modi-SIC HTE record* will be modified to accept also object code of Format 1 instruction of SIC/XE. So this case must be handled.

Generating Symbol Table File

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

Generating Assembly Code:

You will have to generate the assembly code file (assembly.txt). It must contain three columns ordered from left as location counter, the assembly code, and object code.

Assessment

A maximum of 2 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.

Student 1	**Student 2**
Input Parsing	Input Parsing
Format 1 and Format 3 (normal)	Format 1 and Format 3 (with immediate optio
Handling memory reservations (Byte, Word)	Handling memory reservations (RESB, RESW
Generating Symbol Table (symbolTable.txt)	Generating assembly file (assembly.txt)

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
- design issues and sample run