## Experiment No. 8

**Aim:** To implement data structures of two pass MASM macro processor.

**Theory:**

In assembly language programming it is often that some set or block of statements get repeated every now. In this context the programmer uses the concept of macro instructions (often called as macro) where a single line abbreviation is used for a set of line. For every occurrence of that single line the whole block of statements gets expanded in the main source code. This is gives a high level feature to assembly language that makes it more convenient for the user to write code easily. Many computer gives use macro instructions to automate the writing of "tailored" operating systems in a process called as system generation. In its simplest form, a macro is an abbreviation for a sequence of operations. Consider the program given in example 1. In this example we see that the three set of statements (add instruction with DATA) occurs twice. The macro processor effectively constitutes a separate language processor with its own language. It provides a name to the sequence of statements (called macro name) that are repeated and ever needed it make use of the name in the main program. The macro definition consists of the following parts:

1. Start of definition MACRO
2. Macro name [ ]
3. Sequence of statements ---------- ----------
4. End of definition MEND

In the definition of the MACRO and MEND are the pseudo code that represents the start and the end of the macro definition. MACRO being the first statement, the second statement being the name of the macro instruction with is an identifier what will be used whenever the macro is called. The statements from the third till the MEND represent the body of the macro. Once the macro is defined then the name of macro instruction now acts as a mnemonic in assembly language that is equivalent to sequence of the statements. Now whenever the macro instruction is encountered, the whole of the macro gets substituted in place of the macro instruction. This process of substitution is called as macro expansion. This expansion takes place as the definition of the macro instruction is saved in the macro processer and is used whenever the macro instruction is encountered. A macro call represents the expansion of the macro in the source program.

SPECIFICATION OF DATA BASE FORMAT

The only data bases with non-trivial format are the MDT, MNT and ALA. Others are of less importance.

*ARGUMENT LIST ARRAY:* used during both pass1 and pass2 but some the functions are just the reverse. During pass 1 the ALA stores the arguments in the macro definition with positional indicator when the definition is stored in MDT i.e. the ith argument is stored in the ALA as #i. This arrangement is in according to the order of argument in which they appear in the MDT. Later on in the pass2, when there is macro expansion the ALA fills the arguments of the corresponding index with its appropriate argument in the call. This will clear from the example. Here when LOOP INCR DATA1, DATA2, DATA3 is executed the ALA fills the argument fields of the corresponding index #0, #1, #2, #3 with LOOP, DATA1, DATA2 and DATA3.

*Argument List Array*

8 bytes per entry

| *Index* | *Argument* |
|---|---|
| 0 | "LOOP1*bbb*" |
| 1 | "DATA1*bbb*" |
| 2 | "DATA2*bbb*" |
| 3 | "DATA3*bbb*" |

(*b* denotes the blank character)

*MACRO DEFINTION TABLE:* It is table of text lines. It consists of two fields, index that keep track of line numbers of the macro definition and the card that is 80 bytes of size and is responsible for storing the macro definition. Everything except the pseudo code MACRO is inserted into the MDT. MEND pseudo code marks the end of the macro definition.

*Macro Definition Table*

80 bytes per entry

| *Index* | *Card* | | |
|---|---|---|---|
| ⋮ | ⋮ | | |
| 15 | &LAB | INCR | &ARG1,&ARG2,&ARG3 |
| 16 | #0 | A | 1, #1 |
| 17 | | A | 2, #2 |
| 18 | | A | 3, #3 |
| 19 | | MEND | |
| ⋮ | ⋮ | | |

*MACRO NAME TABLE:* It is similar to out MOT and POT in assembler. It has got three field, Index field that keep track of various macro that are defined, the Name field that keep track of names of the macros and the MDT index is a pointer to the entry in MDT.

| | 8 bytes | 4 bytes |
|---|---|---|
| *Index* | *Name* | *MDT index* |
| ⋮ | ⋮ | ⋮ |
| 3 | "INCR*bbbb*" | 15 |
| ⋮ | ⋮ | ⋮ |

**File:**

```
RDBUFF MACRO &INDEV,&BUFADR,&RECLTH
CLEAR X
CLEAR A
CLEAR S
+LDT #4096
TD =X'&INDEV'
JEQ *-3
RD =X'&INDEV'
COMPR A,S
JEQ *+11
STCH &BUFADR,X
TIXR T
JLT *-19
STX &RECLTH
MEND
WRBUFF MACRO &OUTDEV,&BUFADR,&RECLTH
CLEAR X
LDT &RECLTH
LDCH &BUFADR,X
TD =X'&OUTDEV'
JEQ *-3
WD =X'&OUTDEV'
TIXR T
JLT *-14
MEND
```

**Code:**

```
ass_code= open('code.txt', 'r').read()
MNT=[]
MDT=[]
ALA=[]
print(ass_code)
ass_code_lines=ass_code.split('\n')
index=0
start=False
record=""
for line in ass_code_lines:
line_=line.split('\t')
print(line_)
if "MACRO" in line_:
MNT.append(" "+line_[0]+" : "+str(index+1))
record=record+str(index+1)+" = "
```

```
start=True
ALA.append(" "+line_[0]+" : "+line_[2])
elif "MEND" in line_:
start=False
MDT.append(record)
record=""
elif start:
record+=line
index+=1
print("MNT :",MNT)
print("MDT :",MDT)
print("ALA :",ALA)

#--------------------OUTPUT-----------------------------
# Python 3.6.1 (default, Dec 2015, 13:05:11)
# [GCC 4.8.2] on linux
# RDBUFF MACRO &INDEV,&BUFADR,&RECLTH
# CLEAR X
# CLEAR A
# CLEAR S
# +LDT #4096
# TD =X'&INDEV'
# JEQ *-3
# RD =X'&INDEV'
# COMPR A,S
# JEQ *+11
# STCH &BUFADR,X
# TIXR T
# JLT *-19
# STX &RECLTH
# MEND
# WRBUFF MACRO &OUTDEV,&BUFADR,&RECLTH
# CLEAR X
# LDT &RECLTH
# LDCH &BUFADR,X
# TD =X'&OUTDEV'
# JEQ *-3
# WD =X'&OUTDEV'
# TIXR T
# JLT *-14
# MEND

# ['RDBUFF', 'MACRO', '&INDEV,&BUFADR,&RECLTH']
# ['', 'CLEAR', 'X']
# ['', 'CLEAR', 'A']
# ['', 'CLEAR', 'S']
```

```
# ['', '+LDT', '#4096']
# ['', 'TD', "=X'&INDEV'"]
# ['', 'JEQ', '*-3']
# ['', 'RD', "=X'&INDEV'"]
# ['', 'COMPR', 'A,S']
# ['', 'JEQ', '*+11']
# ['', 'STCH', '&BUFADR,X']
# ['', 'TIXR', 'T']
# ['', 'JLT', '*-19']
# ['', 'STX', '&RECLTH']
# ['', 'MEND']
# ['WRBUFF', 'MACRO', '&OUTDEV,&BUFADR,&RECLTH']
# ['', 'CLEAR', 'X']
# ['', 'LDT', '&RECLTH']
# ['', 'LDCH', '&BUFADR,X']
# ['', 'TD', "=X'&OUTDEV'"]
# ['', 'JEQ', '*-3']
# ['', 'WD', "=X'&OUTDEV'"]
# ['', 'TIXR', 'T']
# ['', 'JLT', '*-14']
# ['', 'MEND']


# MNT : [' RDBUFF : 1', ' WRBUFF : 16']
# MDT : ["1 =
\tCLEAR\tX\tCLEAR\tA\tCLEAR\tS\t+LDT\t#4096\tTD\t=X'&INDEV'\tJEQ\t
*-3\tRD\t=X'&INDEV'\tCOMPR\tA,S\tJEQ\t*+11\tSTCH\t&BUFADR,X\tTIXR\
tT\tJLT\t*-19\tSTX\t&RECLTH", "16 =
\tCLEAR\tX\tLDT\t&RECLTH\tLDCH\t&BUFADR,X\tTD\t=X'&OUTDEV'\tJEQ\t*
-3\tWD\t=X'&OUTDEV'\tTIXR\tT\tJLT\t*-14"]
# ALA : [' RDBUFF : &INDEV,&BUFADR,&RECLTH', ' WRBUFF :
&OUTDEV,&BUFADR,&RECLTH']
```

**Conclusion:**

From this experiment I got to know about the working of MACRO’s, assembler etc.