Yeditepe University
Department of Computer Engineering

CSE 232 Systems Programming
Spring 2023

Term Project

Macro-processor for Assembly Language

Due to: 28 May 2023

5 Students in a Group

In this project, you will implement a macro-processor for M6800 assembly language.

Write your macro-processor in C. Use gcc compiler on Linux.

The macro-processor must perform the following functions:
- Read the macro definitions from a file and store them in a memory buffer
- Check the condition for macro expansion
- Create the parameter table
- Macro expansion with parameters

Your macro-processor, named `my_mproc`, must read the macro definitions and the program from a file (e.g. filename) and writes the expanded code to a file with .asm extension (e.g. filename.asm). Macro definitions are found at the beginning of the file and they are in the following format:

```
#MNAME: MACRO P1,P2
         instructions
         ...
        #ENDM
```

where `P1,P2,...` are the dummy parameters. There can be maximum 3 parameters.

Following is an example program with macro calls:

```
instructions
  ...
#MNAME 3,5     //3,5 are the actual parameters
  ...
instructions
  ...
#if ($1='2') MNAME 7,8   // if condition parameter $1 is '2' then call macro with actual parameters 7,8
  ...
instructions
  ...
```

Your macro-processor must take the input file name and condition parameters as arguments, using argc, argv.
Ex: `./my_mproc filename, 5,6`
argv[1] is the `filename`, argv[2], argv[3]... are condition parameters (5 and 6 in this example).

The macro-processor uses a memory buffer to keep the macro definitions. You must use the following data structures:

```
struct mac {
       char   mname[8];     //macro name
       char   param[10][4]; //max. 10 parameters, each parameter max. 3 characters
       char   macro[256];   //macro body
}
struct mac buffer[10];     // memory buffer for 10 macro definitions
int    m_count;      // counts the number of macro definitions
```

Write the following functions to implement the macro-processor:

`int read(char* filename)`
Reads macro definitions from the file and stores them in the memory buffer. It returns the number of macro definitions.

`parse(char* filename)`
The program with macro calls follows the macro definitions. The program is read line by line. `parse()` `function` parses the current program line and stores its fields to the following data structure:

```
char   field[10][7];       //max 10 fields in a line, each field max 6 characters
```

Examples:

```
L1: LDAA #20H        Fields are:      L1:    LDAA   #20H
    STAA 2000H       Fields are:      STAA   2000H
    #M1        // calls macro M1                        Fields are:      M1
    #M2 2,3    // calls macro M2 with parameters 2 and 3 Fields are:      M2     2      3
    #if ($2='ABC') M3 200, 300     // calls M3 with parameter 200 if the condition is true
                                Fields are:      #if    $2     ABC    M3     200    300
```

`is_macro()`
It checks if there is a macro call in the current line. (Macro calls start with a macro name or `#if`). If so, it calls `expand()` to expand the macro. Otherwise writes the line into the .asm file.

`is_macro()` function must also check if the macro call has a condition. Conditional calls have the following format:

```
#if ($2='4') MNAME 3,5
```

`$2, $3, $4, $5` are condition parameters that are given as arguments to the macro-processor such that `$2` is `argv[2], $3` is `argv[3],` etc.
`is_macro()` function must evaluate the condition and call `expand()` if it is true.

`expand()`
It first calls `createPT()`. Then it takes a line from the macro body and writes it to the expanded code. If there are any dummy parameters in that line, it also substitutes them by the actual parameters using the parameter table.

`createPT()`
It fills the parameter table for the current macro call. You must use the following data structure:

```
struct pt {
       char   mname[8];     //macro name
       int    nparams;      //number of parameters
       char   dummy[10][4]; //max. 10 parameters, each parameter max. 3 characters
       char   actual[10][4];
}
struct pt PT;
```

Main program:

```
call read()
for all lines in filename
{
       call parse()
       call is_macro()
       call expand()
}
```

You must also write the following macro definitions using M6800 instruction set:

1. ADD macro adds the numbers at memory locations defined by the second and third parameters and stores the result to the location defined by the first parameter.
   Ex: Macro definition:
   ```
   #ADD:  MACRO D, A1, A2
   ...
   #ENDM
   ```
   called as `#ADD 100, 200, 300` for M[100]←M[200]+M[300]

2. SUB macro, similar to ADD, subtracts the numbers at memory locations defined by the second and third parameters and stores the result to the location defined by the first parameter.

3. MULT macro multiplies the number at memory location defined by the second parameter by the power of 2 given in the third parameter and stores the result to the location defined by the first parameter.
   Ex: Macro definition:
   ```
   #MULT:  MACRO D, A1, P
   ...
   #ENDM
   ```
   called as `#MULT 100, 200, 3` for M[100]←M[200]*2^3

4. DIV macro divides the number at memory location defined by the second parameter by the power of 2 given in the third parameter and stores the result to the location defined by the first parameter.
   Ex: Macro definition:
   ```
   #DIV:  MACRO D, A1, P
   ...
   #ENDM
   ```
   called as `#DIV 100, 200, 3` for M[100]←M[200]/2^3

5. REM macro divides the number at memory location defined by the second parameter by the power of 2 given in the third parameter and stores the remainder to the location defined by the first parameter.
   Ex: Macro definition:
   ```
   #REM:  MACRO D, A1, P
          ...
   #ENDM
   ```
   called as `#REM 100, 200, 3` for M[100]←remainder of M[200]/2^3

Example:

Input file `f1`:

```
#ADD: MACRO P1,P2,P3
        ...
      #ENDM
#DIV: MACRO D,A,P
        ...
      #ENDM
#NEG: MACRO D, A
      LDAB A
      NEGB
      STAB D
      #ENDM
```

```
      PROG
      LDAA   #3
      #NEG 100H,200H
      INCA
      #if ($2='x') NEG 300H,400H
      STAA 500H
      END
```

Macro-processor command and condition parameters:

```
./my_mproc f1 x
```

Output file `f1.asm`:

```
      PROG
      LDAA   #3
      LDAB 200H
      NEGB
      STAB 100H
      INCA
      LDAB 400H
      NEGB
      STAB 300H
      STAA 500H
      END
```