Laboratory – Microprocessor Based Systems

Laboratory Assignment 3: Program design based on C and assembler for the 8086 microprocessor

For this laboratory practice you will write a main program coded in C language (pract3.c). This main program makes different calls to different functions. These functions must be developed on two different files, both coded on assembler. Every student must analyse the requirements showed down below and implementing them.

Task 1. Pract3a.asm

On pract3a.asm file will be developed in assembler a set of functions for mathematical calculations. This functions will be called from a main program (coded on C) accordingly with the following prototipes:

1.- int OddPositive (int num)

If 'num' is an odd and positive number, It gives back 1. 0, otherwise (num is an int number with 16 bits, sign included). It is not allow using IDIV nor DIV instructions.

2.- int DigitComputation (unsigned int num, unsigned int pos)

It gives back the digit on the 'pos' position from the 'num' number (num and pos are int numbers with 16 bits, sign included). For instance, if num=65432 and pos =1, the function outcome will be 3. Positions start at zero and from right to the left.

3.- unsigned int NextPrime (unsigned int n)

It gives back the first prime number coming after n (int number with 16bits, sign included). This function must supply rigth results for values on the range from 2 to 65520. For instance, if n=7, the outcome will be 11.

Task 2: pract3b.asm

On pract3b.asm file will be developed in assembler a set of functions related with strings. This functions will be called from a main program (coded on C) accordingly with the following prototipes:

1.- int SubstringFinder (char* str, char* substr)

It gives back the index since where 'substr' starts on the string 'str'. This functions gives back -1, if the substring is not content on the string. For instance, if str=' abcdefghijklmn' and 'substr'=defghijk, the outcome will be 3.

2.- unsigned int SecondComputDC (char* numCuenta)

The outcome from this functions will be the second control digit from a 10-digits bank account, given previously as an ASCII string (as an example: ‘ 0438853602’). To make it, this function multiplies every bank account digit by 1, 2, 4, 8, 5, 10, 9, 7, 3, 6 respectively, starting from left to the right. Once we get the results, we will add these ten numbers and compute mod-11 of that number for achiving the final result. The outcome will be 11 substracted by mod-11 number computed during the previous step. There will be one exception, the outcome will be 1 if the final result is 10. For instance, AccountNumb = ‘0438853602’, the outcome will be 9 (as an int number with 16 bits, sign included).

Notas:

- In order to work properly, the compiler needs underscore character ‘_’ at the begining of each developed function on assembler (as an example: `_SecondComputDC`).
- All developed functions on assembler must be declared as PUBLIC to be able to call them from the main program.
- It is not allow declaring global variables on assembler. In fact, every developed module on assembler will have exclusively only one code segment which will start as the following:

```
<module name> SEGMENT BYTE PUBLIC 'CODE'
ASSUME CS: <module name>
```

- The C program (pract3.c) must be compile on LARGE model. Functions developed on assembler will be FAR.
- pract3.c program will ask for different values to check every developed function on assembler in the different modules, showing on the screen an understandable result with the right format.
- All C-coded strings end with NULL character (last byte is zero).

Anexo: How to compile a project developed on C and assembler together

In this practice we have 3 files, a main program in C ((pract3.c) and 2 modules with the functions coded on assembler (pract3a.asm y pract3b.asm).

In order to compile this C-program we will be using TurboC (tcc) compiler. Execute tcc witout parameters inside DosBox, for finding out all the options. Assembler modules will be compiled as we have done until now, using tasm.

This C-program must be compiled using -ml (memory model large) option, while assembler modules must be assembled using /ml (case sensitivity on all symbols).

We can use the following makefile (be carefull with tabs) for obtaining the final executable ‘prac3.exe’:

```
all: pract3.exe

pract3.exe: pract3.obj pract3a.obj pract3b.obj

	tcc -v -ml -Lc:\compila\tc\lib pract3.obj pract3a.obj pract3b.obj

pract3.obj: pract3.c

	tcc -c -v -ml -Ic:\compila\tc\include pract3.c

pract3a.obj: pract3a.asm

	tasm /zi /ml pract3a,,pract3a

pract3b.obj: pract3b.asm

	tasm /zi /ml pract3b,,pract3b
```