Microprocessor-Based Systems Laboratory

Lab Session 3: Implementation of programs using C and 80x86 assembler

The videogame company Feic SL is currently in the development stage of a game based in the popular desk game "MasterMind". In this game, the player has to guess a 4-digit secret number in several consecutive attempts (possible values of each digit between 0 and 9). Each digit of the secret number has to be generated randomly, and no repeated digits are allowed in the secret number.

The game developed has to compare each player attempt with the secret number, and provide the following information on screen: how many digits of the attempt match and are located in the same position as in the secret number (number of matches) and how many digits of the attempt match but are located in a different position with respect to the secret number (number of semi-matches).

Due to restrictions in game execution speed and memory constraints, the game is to be developed in two different programming languages: C language for the main program and 80x86 assembler for internal functions. The development of the game was assigned to Zote Cenutrio Pazguato, due to his vast experience and programming speediness in both languages. Unfortunately for the company, Zote won the Euro-million jackpot leaving the project unfinished: he managed to finish all the C-programming but left the assembler internal functions undone.

In this lab assignment the students are required to complete the game development by programming all the internal functions of the game in 80x86 assembler. These functions will be called from the main program already developed in C language (pract3.c, provided in Annex II). These internal functions have to be coded in two different files (modules). Students shall analyze the requirements for each function and do their implementation. Each function contributes with the same weight to the final assignment grade.

Module 1: pract3a.asm

In this module (file pract3a.asm) student has to implement two functions in assembler. The functions will be called from the C main program according to the following prototypes:

1.- unsigned int checkSecretNumber(unsigned char* number);

This function shall return the value 1 if "number[]" contains any repeated digit, otherwise it shall return the value 0 (number[]" is a 4-element array, each element contains one digit of the secret number).

2.- void fillUpAttempt(unsigned int attempt, unsigned char* attemptDigits);

This function fills the 4-element array "attemptDigits[]", storing at each array position the corresponding digit of the 4-digit number "attempt". For example, if attempt equals 4567, this function shall fill the array in this fashion: attemptDigits[0] = 4; attemptDigits[1] = 5; attemptDigits[2] = 6; attemptDigits[3] = 7.

Module 2: pract3b.asm

In this module (file pract3b.asm) student has to implement two functions in assembler. The functions will be called from the C main program according to the following prototypes:

1.- unsigned int computeMatches(unsigned char* secretNum, unsigned char* attemptDigits);

This function shall return the number of matches in the 4-element array "attemptDigits[]" with respect to the 4-element array "secretNum[]". An element of "attemptDigits[]" is considered a match if it matches in value and position with an element of "secretNum[]". For example, if secretNum[] = {1,2,3,4} and attemptDigits[] = {5,6,3,4}, this function shall return the value 2.

2.- unsigned int computeSemiMatches(unsigned char* secretNum, unsigned char* attemptDigits);

This function shall return the number of semi-matches in the 4-element array "attemptDigits[]" with respect to the 4-element array "secretNum[]". An element of "attemptDigits[]" is considered a semi-match if it matches in value with an element of "secretNum[]", but it is placed in a different position. For example, if secretNum[] = {1,2,3,4} and attemptDigits[] = {3,1,7,4}, this function shall return the value 2.

Notes:

- As the compiler precedes all extern references with the character '_', it is necessary that all the assembler functions start with this character
  (for example: `_computeMatches`)
- All the assembler functions have to be declared as PUBLIC in order to be called from the C main program
- Declaration of global variables is not allowed in the assembler modules. Each assembler module will **exclusively** contain a segment code, starting/ending with the following lines:

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

END
```

- The C program developed (pract3.c) has to be compiled in LARGE model (proper makefile is provided in Annex I). Therefore, assembler functions will be declared as FAR.

DELIVERY: Date and contents

Students shall upload to Moodle a ZIP file containing the source files (only the .asm) of the exercises and the makefile. Only one member of the team may upload the file. The files shall contain the authors' name and the team number in the header.

The source files shall be tabulated and commented correctly. The lack of comments or poor quality comments will be qualified negatively.

The date limit to upload the files is April 19th at 23:55h

Annex I: Compilation of a project containing C and assembler files

In this assignment we will have three different source files, one main program written in C (pract3.c) and two files containing the functions written in assembler (pract3a.asm y pract3b.asm).

The TurboC (tcc) compiler will be used to compile C files. The different options provided by TurboC can be shown by typing "tcc" at the DosBox prompt. The assembler modules will be assembled as in the previous practices.

The C program shall be compiled with the option –ml (memory model large), whereas the assembler modules shall be assembled with the option /ml (case sensitivity on all symbols).

In order to generate the executable file "pract3.exe" the following makefile can be used (including tabulations):

```
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
```

Annex II: Main program developed in C language

```
#include <stdio.h>
#include <stdlib.h>

#define TRUE  1
#define FALSE 0
#define NUMBERATTEMPTS 5

unsigned int checkSecretNumber(unsigned char* number);
void fillUpAttempt(unsigned int attempt, unsigned char* attemptDigits);
unsigned int computeMatches(unsigned char* secretNum, unsigned char* attemptDigits);
unsigned int computeSemiMatches(unsigned char* secretNum, unsigned char* attemptDigits);

////////////////////////////////////////////////////////////////////////
///// ------------------------- MAIN ----------------------------- /////
////////////////////////////////////////////////////////////////////////

int main( void )
{

	int t;
	unsigned char secretNum[4];
	unsigned char attemptDigits[4];
	unsigned int numAttempts, attempt, matches, semimatches, repeated, i;

	srand((unsigned) time(&t));

	do {
		for (i=0; i<4; i++)
			secretNum[i] = rand() % 10;
		repeated = checkSecretNumber(secretNum);
	} while (repeated == TRUE);

	numAttempts = 0;

	do {
		numAttempts++;
		do
		{
			printf("Please enter attempt %u [0000 - 9999]: ", numAttempts );
			scanf("%u", &attempt);
		}
		while ( attempt > 9999);

		fillUpAttempt( attempt, attemptDigits );
		matches = computeMatches(secretNum, attemptDigits);
		semimatches = computeSemiMatches(secretNum, attemptDigits);
		printf("Number of matches: %u\t", matches);
		printf("Number of semi-matches: %u\n", semimatches );

	} while ((matches != 4) && (numAttempts != NUMBERATTEMPTS));

	if (matches == 4)
		printf("Secret number guessed: YOU WIN!!!\n");
	else
		printf("Number of attempts exceeded: YOU LOSE :(\n");

	printf("Secret number: %u%u%u%u\n", secretNum[0], secretNum[1], secretNum[2],
secretNum[3]);

	return 0;
}
```