Exercise 1 – x86 Assembly

Deadline: 13.03.2013, 15:00

Part 1 - .COM programming (50 points)

We saw in class that a .COM file is the simplest, in terms of sections and infrastructure code. In this part you will create .COM files in order to brush up on your basic assembly programming skills.

Some notes:

- If you need a simple debugger for your assembly code, `debug.exe` might be of help. This file exists by default for every Windows installation. Run it and type `'?'` to get help.
- You must terminate your programs normally and set a return value using DOS interrupt `21h`, subroutine `4Ch` as seen in class. The return value of the program can be obtained by typing `echo %errorlevel%`.
- Your functions must not change register values except for the register which is defined as the return value. Use the stack to save and restore registers.
- Use NASM to assemble your code. Sample command line: `nasm code.asm –o code.com`.
- Assembly does not mean unreadable. Make sure your code is readable, formatted properly and well documented!

Question 1

In this question you will implement a Carmichael number detector. A Carmichael number is a composite positive integer n which satisfies: $b^{n-1} \equiv 1\ (mod\ n)\ \forall\ b : gcd(n, b) = 1$.

Use Korselt's criterion to test if a given number is a Carmichael number:

- n is square –free (is divisible by no perfect square except 1, meaning $n = p_1 * p_2 * \ldots * p_k$ where all p_i are prime and no prime appears more than once).
- For all prime divisors p of n: $(p-1)$ divides $(n-1)$.

Implementation steps:

1. Write the function `calc_mod` that receives two numbers in `ax` and `bx`, calculates `ax % bx` and returns the result in `ax`. Use only subtraction in order to implement this function (and not `div`).

2. Create the function `is_prime` that receives a single number in `ax`, and returns 1 in `ax` if the number is prime, 0 otherwise. Your function should use your `calc_mod` function (and not `mod` or `div`).
3. Finally, implement the function `is_carmichael` that receives a single number in `ax` and returns 1 in `ax` if the number is a Carmichael number or 0 otherwise, by using Korselt's criterion.

For example, while the following code is executed:

```
mov ax, 10
call is_carmichael

mov ax, 7
call is_carmichael

mov ax, 561
call is_carmichael
```

`ax` will contain 0 after the first call, 0 after the second call, and 1 after the third call.

You will also need to implement a test code for your function. The test code will a predefined value (use `dw` to define it) and test if it is a Carmichael number. The result will be the return value of the program. The program must terminate normally.

Some notes:

- All code in this question should be 16-bit (e.g. use `ax`, `bx` and not `eax`, `ebx`).
- All numbers in this question are unsigned.
- You can assume that the right operand of the mod function is not zero.
- The code for this question should reside in a file named `carmichael.asm`.

Question 2

In this question you'll create a "guess the number" game. In this game the computer guesses a random number, and you try to guess it. The computer will give you higher/lower hints until you reach the number. The game goes as follows:

- The string "`I selected a random number between 0 and 99.\r\n`" is displayed.
- For every game iteration:
    - The string "`Please enter your guess:<space>`" is displayed.
    - The user enters a 2-digit number. For numbers between 0 and 9 the user writes the number with a preceding '0'. Only two characters are given, no <enter> is needed.
    - If the number is lower than the computer's number, the string "`\r\nYour number is lower than mine.\r\n`" is displayed.

  - If the number is higher than the computer's number, the string "`\r\nYour number is higher than mine.\r\n`" is displayed.
  - If the number equals the computer's number, the string "`\r\nYou win! Try count: <number of guesses>.\r\n`".
- Game iterations end when the user guesses the number correctly.
- The program terminates normally with return value `0`.

Some notes:

- All parameters passed to functions in this question must be **passed through the stack**. Do not use registers for this purpose. Create a stack frame as seen in class.
- When a function has a return value it should be returned using `ax`.
- The code must support up to 255 guesses.
- When displaying the guess count, leading zeros must not be displayed.
- Please make sure your strings are exactly as defined in this exercise, including dots, newlines, spaces etc. Automatic testing will be used.
- This question requires generating a "random number". For this purpose you will use the centiseconds from the current system time. See `int 21h`, function 2Ch for details.
- Video of a typical run of the program can be found on the course's site.
- The code for this question should reside in a file named `game.asm`.

Part 2 - .EXE programming (50 points)

Question 3

In this question you'll write a very simple login program. In order to do so you'll use 32-bit assembly and standard C functions. The flow of the program is as follows:

- Read 8 bytes from a file called "`login.txt`"
- Compare the string to a predefined string in the code, named password. The password should be "`13371337`"
- If the two match, display the string "`Welcome!\r\n`", otherwise display the string "`No match.\r\n`"

All errors should be reported to the user, and an appropriate return value must be set as follows:

Error	Message	Return value
Unable to open file	"`Cannot open password file\r\n`"	1
Cannot read file or not enough bytes in file	"`Error reading password file\r\n`"	2
Password does not match	"`No match.\r\n`"	3

If no error has occurred, the return value should be 0.

You may find the following standard C library functions useful: `fopen, fread, fclose, memset, printf`. Don't forget that the standard library uses the `cdecl` calling convention.

In your code you will need to compare 2 strings. In order to do so you may not use standard library functions such as `strcmp` and you must implement this functionality yourself. In your implementation you'll need to use the repeat string operation prefix and the string functions. Read more about `rep/repe/repz/repne/repnz` and `movs/lods/stos/cmps/scas` and use the appropriate ones for you. Your function should be named `my_strcmp`.

Some notes:

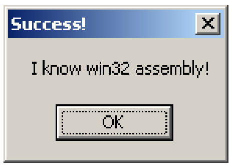
- As before, use the stack to pass parameters to functions.
- As before, your functions must not change register values except for the register which is defined as the return value. Use the stack to save and restore registers.
- You may assume that the file "`login.txt`" contains exactly 8 characters.
- In your code you'll need to store the data read from the file and a FILE*. Both of these should be stored as locals on the stack using a stack frame as taught in class.
- Your file should be named `login.asm`.
- In the lecture you've seen the commands used to assemble and link under Mac OSX. Under Windows use `nasm -f win32 login.asm` to assemble and `gcc login.obj -o login.exe` to link.

Question 4

In this question you'll take your first steps in Win32 API assembly programming. In order to do so you'll need to get acquainted with the functions `LoadLibraryA` and `GetProcAddress`.

In windows all API function reside in dynamic link libraries (DLLs). This will be discussed further in class. For now, all you need to know is that code can be dynamically loaded from these files (much like .so files in Linux). In order to load a specific DLL file into your address space you'll need to use the `LoadLibraryA` function. In order to obtain a pointer to a specific function within the DLL you'll need to use `GetProcAddress`. These two functions themselves are API functions which are available to you (the usage is similar to how you call the standard C functions in the previous question).

`MessageBoxA` is a function used to display a standard Windows message box. All your program needs to do is display a message box like this:



I know win32 assembly!

You'll get the pointer to `MessageBoxA` by dynamically loading `User32.dll` as previously explained.

Some notes:

- As before, all error conditions should be checked and handled appropriately. You may choose your own error messages and return values as long as they are informative.
- The first parameter to `MessageBoxA`, `hWnd`, can be `NULL` (=0).
- Due to name decoration you'll need to use `_LoadLibraryA@4` and `_GetProcAddress@8` instead of the original function names. You can read more about it here.
- Remember that WinAPI calls use the `stdcall` calling convention (what does this mean about stack unwinding?)
- Your file should be named `messagebox.asm`.

Part 3 – Bonus (25 points)

Question 5

Write a .COM file which will print your full names, by using **only** opcodes and operands represented by printable ASCII characters, while trying to use as few different characters as possible. There is no limit on the size of your solution (aside from the .COM file size limit – 64K). Full credit will be given for using 3 different characters or less (hard!), partial credit for using more characters (fewer characters == more credit). Submit this part as `bonus.com` and describe in the README exactly what you did.

Submission

All solutions should assemble and link under the Windows machines in Ross 19.

The following files should be included in the submitted tarball file:

- `carmichael.asm`
- `game.asm`
- `login.asm`
- `messagebox.asm`
- README – Formatted according to the course guidelines. Please add a bonus section if you decided to solve the bonus part.
- `bonus.com` (if you chose to submit the bonus)

A single TAR file named `ex1.tar` should be submitted. Make sure you run the pre-submit script before submitting the exercise.

Good luck!