

## ELEC3730 - Assignment # 2

- To attempt questions 1 and 2, you need to use the nasm assembler. This can be achieved by the following steps.

- Download `nasm098p.zip` from

`http://www.eecs.newcastle.edu.au/brett/elec3730/software.html`

unzip it to get `nasm.exe` and place it either in your working directory (where you place all your source code) or in your executable search path.

- Set up DEV-C++ to invoke the nasm assembler as part of the build process by following these steps:

- \* Create the new project: (File→New→Project) Select: Empty C project;
- \* Add the C and assembly files to the project (set the file filter to "All files" instead of "all known files" to be able to see the `.asm` files);
- \* Under Project → Project Options:
  - Choose files tab;
  - Select the assembly file;
  - Tick "include in compilation" and "include in linking";
  - In the "override build command" box, type  
`nasm -f win32 -o filename.o filename.asm`  
where "filename" is the name of your assembly file.

These instructions assume that the project file is saved in the same folder as the source files.

- To attempt question 3, you need the file `a2.zip`. It is a zip compressed archive which you may download in conjunction with this assignment (it's link is next to the one you used to download this). It contains the following:

- A port of the  $\mu$ COS-II real time operating system that runs inside a command window of Microsoft XP or 2000;
- A program that, via using this port, demonstrates four tasks sharing a common message buffer which is printed to the screen;
- A project file definition which allows for compilation of this demonstration together with linking against the  $\mu$ COS-II port using the DEV-C++ development environment;
- A file called README which explains how to unpack this archive and perform the appropriate compile/build process.

The purpose of this material is to provide you with a basic framework that illustrates the mechanics of using  $\mu$ COS-II.

Finally, note that this port  $\mu$ COS-II includes some extra PC support functions, such as

```
PC_DispStr(INT8U x, INT8U y, INT8U * s, INT8U color)
```

which will be very useful to you. Their use is illustrated in the demo program supplied, but further documentation is available in the

```
ucos-dev/ucos-lib/win32-src/pc.c
```

file, and also at:

[http://www.it.fht-esslingen.de/~zimmerma/software/uCOS-II\\_WIN32.htm](http://www.it.fht-esslingen.de/~zimmerma/software/uCOS-II_WIN32.htm)

- Your answers to this assignment should consist of a zipped archive containing your source code (one or more .c files) and DEV-C++ project definition (a .dev file) **per question**.
- You should submit your work via the “Assignments” section of the blackboard homepage for elec3730. A link to this homepage is provided at:

<http://www.ee.newcastle.edu.au/brett/elec3730/notices.html>

- Your assessment for this assignment will be based on this source code. You should be aware that you may be required to demonstrate your solutions (compile and run them) should your tutor require it.

1. **(20 marks)** Mixing x86 assembly language and C code

Write an assembly language routine `regdump` that accepts a byte, and dependent on it, returns the value in a certain internal register. For example, with input of 01, the EBX register value is returned, for input 02, the ECX register is returned, for input 03, the stack pointer is returned, and so on.

Supply C code which tests the routine by calling it multiple times to print out as many of the internal register values as possible. The C prototype for calling the assembly language routine is

```
unsigned long int regdump(char);
```

Note in particular that in your assembly language, you should initialise the values of all the registers, so that you can check that you are passing their value out correctly.

2. **(30 marks)**

Write an assembly language routine with prototype

```
regdump(char *)
```

that accepts a pointer to a string specifying a file name, and then dumps the values of all the registers to an ascii text file of that name. Format the file to have one value per line, with the value represented in hexadecimal format.

Write a C code program to demonstrate the correct working of your routine. In your solution, you may call C standard library functions from within your assembly language code.

To provide a hint towards a solution to both the above problems, the following C and assembly code solves a slightly different problem involving simply printing the registers to the screen, but without passing the registers out to a calling function

C code:

```
#include<stdio.h>

void regdump(char*);

int main()
{
    char s[30]="Doin it from assembler: %ld\n";
    regdump(s);
    return 0 ;
}
```

Assembly code:

```
SECTION .text
ALIGN    16
BITS     32
EXTERN  _printf
GLOBAL  _regdump

_regdump:

; Function prototype:
; -----
; void regdump(char*) ;
;

; Load up register with known values
MOV EAX,123455

MOV EBX,[ESP+4] ; get pointer to format string
PUSH EAX        ; pass parameters to printf
PUSH EBX
CALL _printf    ; call printf
ADD ESP,8      ; don't forget to fix the stack!

RET            ; Return to calling program.
```

### 3. (50 marks) Use of the $\mu$ COS-II real time operating system

As mentioned above, the a2.zip file you can download provides an example of using the  $\mu$ COS-II real time operating system.

You are required to implement a digital clock and alarm using this operating system.

This solution should consist of the following components:

- (a) A highest priority thread which keeps track of tenth's of seconds. It should use the appropriate  $\mu$ COS-II function to implement a tenth of second delay within this highest priority thread in order to set the time base;
- (b) Other threads, one for each to track seconds, minutes, hours, days, weeks, months and years. They should update their counts of these quantities based on semaphores;
- (c) A main loop which monitors for keyboard commands, according to your own design, and which allows for setting alarms, clearing alarms, and listing currently pending alarms;
- (d) A thread or a component of the main loop which prints the current time on the screen;
- (e) A thread which determines when alarms should be activated.

**This is important:** The project .dev file for your solution to this question should be such that if your source file(s) are placed in the /src subdirectory of the a2.zip archive described below, then your assignment solution will build (compile and link) without error.