Assignment #4 Winter 2011
SYSC 2003 Computer Organization
Department of Systems and Computer Engineering

You must submit the files identified below <u>using the electronic Submit application</u> found in your SCE account on the undergraduate network. The submission process will be canceled at the deadline. No assignments will be accepted via email or on disk.

***<u>THIS ASSIGNMENT MUST BE DONE IN TEAMS OF 2 STUDENTS.</u>* <u>Only ONE student must submit the assignment, clearly stating who are the members of the team.**

You must use the Pair Programming technique (check the course webpage for details). There will be questions about the assignment on the Final exam; students not being able to respond them will lose the Assignment marks.

Important instructions about submissions:

- Submit ONLY ONE ASSIGNMENT PER TEAM.
- Each file in your Assignment should include a Header with the names/numbers of BOTH students
- Students submitting TWO copies of the assignment will receive the LOWEST of the two marks and will receive a demerit of 20% of the final mark.
- Students who, by mistake, submit more than one copy, should contact the instructor BEFORE the deadline to fix the problem.

Part A [BONUS – RECOUP for Assign 3. Do not spend time in this part of the assignment if you know the materials of Assignment 3 well]

<u>Due Date for Part A: March 4th, 2011 @ 4:00 PM</u>

1. [Bonus 5 marks] (**recoup41.asm, recoup41.s19**) Using the subroutines

Implement the following subroutines described below <u>in assembly</u>, but use <u>C subroutine programming policies</u> (they will be integrated with C code on Ex. 2).

```
void init(void)   // All hardware init, (eg. setting DDR*, other ...)
void delay(void)        // Delay APPROXIMATELY 1 second
void setLED( unsigned byte index, boolean on )
                        //  where false = 0 and true = non-zero
Void set7Segment( char number, boolean on ) // Display a number on the
                        // 7 Segment Display
```

Write a small main program to test your application, demonstrating the use of your subroutines. The Delay function should wait more or less 1 second (do trial-and-error tests until the function waits the desired amount of time; <u>do not worry about precision in</u>

timing here; 1.2 seconds; 1.4 seconds; 0.7 seconds would be OK. We'll time execution very precisely on Part B and Assign 5-6).

2. [Bonus 5 marks] **Required files: recoup42.prj, recoup42.c, recoup42asm.s, DP256reg.s (provided), RECOUP42.SRC, and recoup42.s19**

Suppose our needs to be used for an elevator system. You must write the subroutine that will lit the floors lights on a display by the elevator. Write a program in C will show where the elevator is. Test it with the following pattern:

- Floor 1 to Floor 4
- Floor 4 to Floor 2
- Floor 2 to Floor 3
- Floor 3 to Floor 1

The software will simulate the elevator movement and will assume it takes 1 second between floors. The lights should be set according to the pattern above, showing in which floor the elevator is. It must also show the floor on the 7-segment display.

To do so, write a main program that, using the pattern above, invokes the setLED subroutine, setting each of the LEDS. Stop 3 seconds on the destination floor before continuing to the next request (i.e, wait 1 second in between floors, and wait 3 seconds between each line in the pattern above).

Your main program will be written in C (you can also re-write *init()* in C if you want), but the setLED subroutine must be completely written in assembly, and it must be called from C to manipulate the LEDs. Among other things, you will now know whether you followed the C calling conventions correctly! Do not use any inline ASM code with perhaps the exception of the last statement (asm("bra *") to end your program).

The program shall perform ONCE and finish.

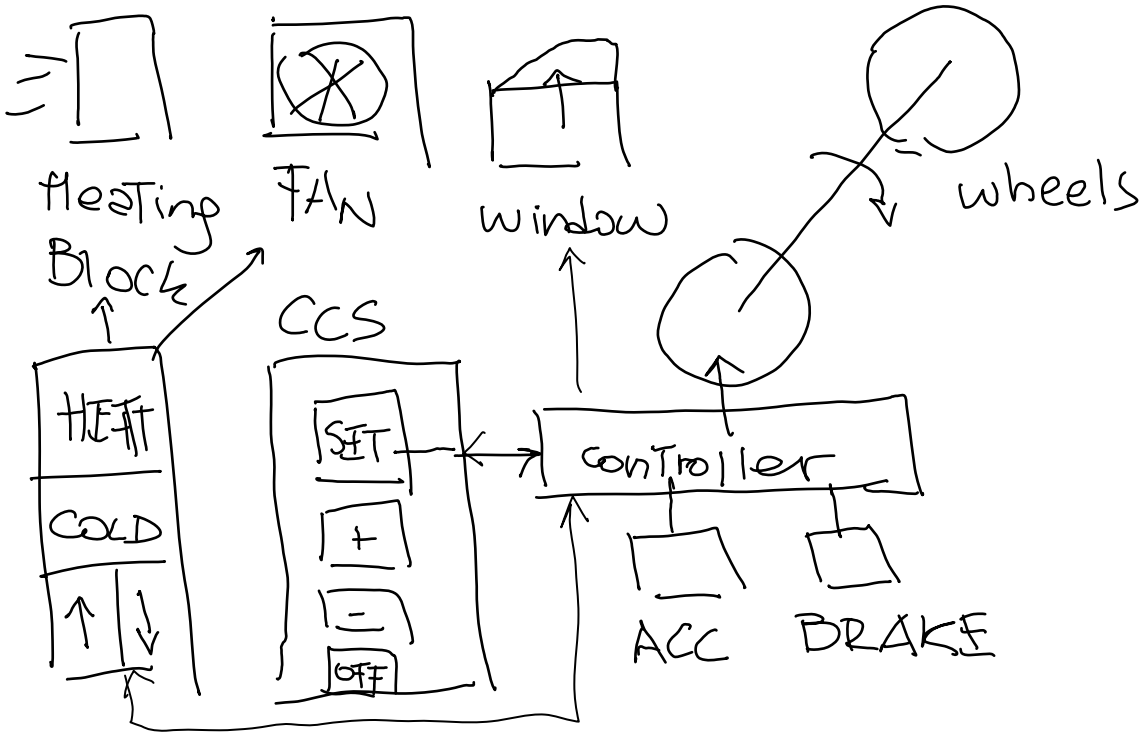
To access the I/O ports from C, you will have to *#include <hcs12dp256.h>*.

Tips:

1. In case you did not notice above, the assembly files must have the extension .s. Copy your *Recoup41.asm* to *Recoup42asm.s* and make any necessary changes or additions.
2. Do not put an ORG statement in your assembly file.
3. Do not put an END statement in your assembly file.
4. Remember that all public labels must have an underscore and two colons; any private label must simply have a colon.
5. To access I/O ports in your assembly file, copy *DP256reg.s* from the course's include directory and .include *DP256reg.s* (don't use #, use .).
6. In *.s* assembly files, variables are declared using *.blkb <value>* where value is the number of bytes (not *ds.b* or *rmb*)
7. Use *char* is used for any unsigned byte value. Furthermore, any no-argument functions must be declared *void* in ICC: *void* sample(*void*).

Part B. Objectives : Programming simple I/O devices in the Project Board. C programming of the HC12 with ICC12. Programming advanced I/O devices. Hardware interrupts.

In this assignment, we will continue building a Control system that we started discussing on Assignment 3. As a reminder, we are trying to build a Control system for an automobile. It will include software to control the heating, ventilation, a simple controller for opening/closing a window, and a simple cruise control system. We will start by creating very simple subroutines, but, in order to understand the whole idea of the system you will have built at the end of the course, we will start with a general description now. The following figure depicts a general description of the system to build.



All the systems in the vehicle are controlled using a single microcontroller. When the driver presses the accelerator, the controller will make the motor (and consequently, the wheels) spin faster. When the brake is pressed, the motor slows down up to the stop.

A Cruise Control System (CCS) is also available. When the user presses the SET button, the current speed is maintained by the controller. To do so, the current speed is first computed (in RPM, Revolutions Per Minute), and stored in a variable that keeps the current speed (we will call this variable the *Set Point* for this control variable). The OFF button turns the CCS system off. The + and - buttons change the Set Point variable (increasing or reducing the programmed speed). When this happens the motor accelerates or diminishes the speed accordingly. When the CCS is turned on, a yellow LED is turned

on to inform that (and the LED is turned off when the CCS is turned off). At all times the current speed is displayed on the panel.

There is a heater is used to keep the vehicle temperature. To turn the heating on, the user pressed the HEAT button. When this happens, a red LED is turned on. When the button "up" is pressed, the temperature goes up. When the button "down" is pressed, the heating temperature is reduced. Also, when the button COLD is pressed, the fan is turned on. In this case, a green LED is set.

Finally, the controller is also used to move the window up/down.

We are a part of the Software Engineering team, which is in charge of building the software for this vehicle. This device is going to be built by a multidisciplinary team including electrical engineers (in charge of the electrical components), mechanical engineers (in charge of the mechanical components), and non-technical staff (user manual's writers, marketing personnel, etc.).

During the requirements analysis, it was decided to build the software on an embedded platform using an HC12 microcontroller (like the one available in the lab). Although most of the hardware is not still available (accelerators, motors, heater, vent, etc.), the Electrical Engineering team has already designed the interfaces to be used. We know that we can use the lab's Experimental Board to start doing some basic tests that will serve as the basis for the actual vehicle. In the car, the hardware will be interfaced as follows:

- The LEDs in the car will be connected to the same circuits used for the lab's LEDs interface;
- The heater will be connected to our lab's heating device interface;
- The panel display in the car is the LCD display that we have in the lab,
- The temperature sensor is connected to the AD interface (like in the lab)
- The accelerator, brake, buttons of the CCS, heating and vent use an interface exactly the same than the one used for the keypad in our lab
- The window will be controlled by a stepper motor, the fan will be connected to the fan in the lab, and the motor's vehicle will be connected to a DC motor (controlled by a PWM interface, like the one in the lab).

Therefore, we have all the equipment needed to start developing the software for this vehicle. In that way, as soon as the rest of the vehicle is built and ready for testing, we can integrate our board, software, connect the tool interfaces to the I/O lines in the car, and we are ready to do integration testing on the actual prototype of the vehicle.

In order to build such complex application, we will start with very simple tasks. At this point, you will:
a. Define the different subroutines discussed here, using the Policies included on each exercise
b. Write a Main program (to be delivered with your subroutine) with the simple purpose to test your subroutine
c. Write good documentation for your main program and your subroutine

In brief, you have to solve the following exercises:

Due Date for Part B: March 9th, 2011 @ 4:00 PM

1. [5 marks] (**assign41.prj, assign41.c, assign41asm.s, DP256reg.s (provided), assign41.SRC, and assign41.s19)** Write a program that will detect the keys pressed in the keypad and act accordingly. The keypad contains the following keys:

column / row	0	1	2	3
0	'1'	'2'	'3'	'A'
1	'4'	'5'	'6'	'B'
2	'7'	'8'	'9'	'C'
3	'E'	'0'	'F'	'D'

We will use the keypad for the operator's console. 'A' will be used to accelerate; 'B' will be used to brake. The first column will be use for the Cruise Control System: '1' will be used to turn it on/off (every time you press '1' the system toggles between on/off). '4' will be to increase speed, and '7' to reduce speed. 'E' is used to cancel cruise control.

When the CCS is on, the green led should be set (and it should be turned off when CCS if off).

The second column will be used for the heating system. '2' will be used to turn on/off the heating (every time you press '2' the heating system toggles between on/off). '5' will be used to increase the heat and '8' to reduce the heat. '0' will be used to turn on/off the vent (every time you press '0' the vent toggles between on/off). '5' will be used to increase the heat and '8' to reduce the heat.

When the heating is on, the red led should be set (and it should be turned off when heating if off). The yellow led is used for the vent.

The third column will be used to control the window. '3' will be used to move the window up, and '6' to move the window down.

As we are testing the software, we will have an 'emergency' button to turn off the system (i.e., to stop the program and finish). If at any time button 'F' is pressed, the system must stop.

- When the emergency button is pressed, the buzzer must sound for 1 second.

Build a **C application** that will check the keys pressed and will act accordingly.
The program runs forever until you PRESS '9'.

DISPLAY EACH KEY PRESSED ON THE TERMINAL.

2. [5 marks] (**assign42.prj, assign42.c, assign42asm.s, DP256reg.s (provided), assign42.SRC, and assign42.s19)**

Add an interface with the LCD display. This routine must:

. Display: “SPEED YYY CCS ON/OFF” on the first line (YY is the current speed in km/h).
. Display “Heating ON/OFF. Temperature XXC” (where XX is the current temperature) on the second line.
. When you press the STOP button, clear the screen and display the message:“Emergency stop”.

You can use Assembly or mixing C and Assembly, but your code built in Part 1 must be reused.

The program runs forever until you reset the board.

Hint:

- First build a routine that only takes care of the display with “dummy” values, and ensure they are displayed properly.
- When interfacing with the keypad subroutines start with values YYY=0, CCS OFF, Heating OFF, Temperature 22C. Every time you push the accelerator, increase YYY (and when the brake is pressed, YYY=0 again). When you press the corresponding buttons, turn on/off the heating/CCS. Initial temperature is 22C; whenever you press the “+” button for heating, increase XX (and decrease it when you press the “-“ heating button.
- Display every button pressed on the terminal for testing.

3. [BONUS - 5 marks] (**assign43.prj, assign43.c, assign43asm.s, DP256reg.s (provided), assign43.SRC, and assign43.s19)**

Convert the Keypad routine into an interrupt-based version.

The main() program should initialize the ISR and then continuously wait until an event is received, acting as in Part 2.

4. [10 marks] Build a simple sequential controller for the window. We will do this incrementally.

a. [5 marks] **assign44.prj, assign44.c, assign44asm.s, DP256reg.s (provided), assign44.SRC, and assign44.s19**

We will use the RTI (Real-Time interrupt) to time the execution of this sequential controller.

The window motor should be moved up/down for 3 seconds whenever the corresponding button is pressed.

We will building the timing controller for the window as follows:

1) Whenever you press the “window up” button, display a message (“window going up” on the terminal screen, wait exactly 3 seconds, and display a second message (“window is now up”)
2) Whenever you press the “window down” button, display a message (“window going down” on the terminal screen, wait exactly 3 seconds, and display a second message (“window is now down”)
3) Beep for 1 second.

The timing control must be driven by the RTI. All the activities are done by the interrupt service routine, while the main program sets up the system. You have to reset the board to finish.

b. [5 marks] **assign4b.prj, assign4b.c, assign4basm.s, DP256reg.s (provided), assign4b.SRC, and assign4b.s19**

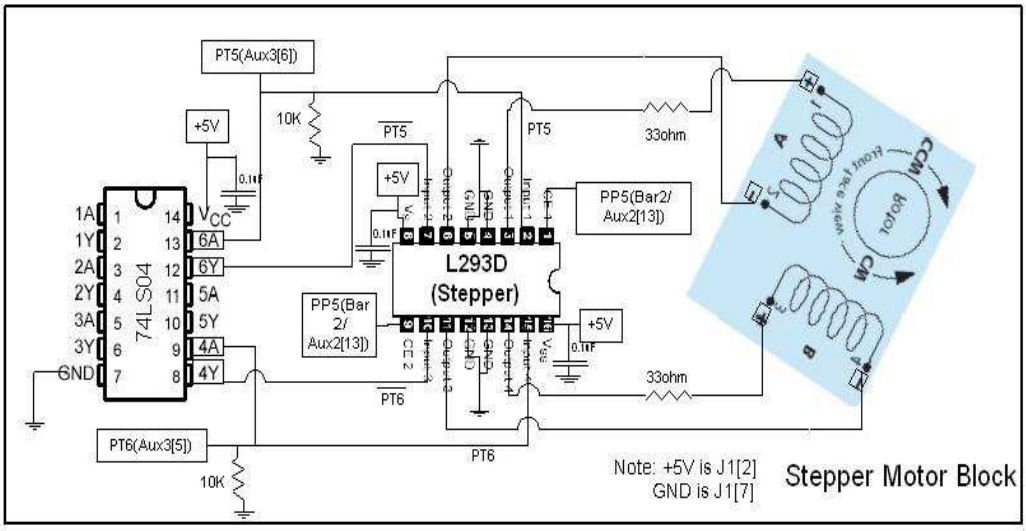
In reality, we are planning to connect a Stepper Motor to control the window.

Write a program that will turn the stepper motor on each of the 2 possible directions for approximately 3 seconds. IN THIS EXERCISE, DO NOT WORRY ABOUT TIMING. DO NOT USE PART 4A. DO NOT USE THE TIMER ISR.

The main program should take care of the keypad, using the routines you wrote for Exercise 2, that is, the whole functionality of the keypad should be available. The program ends when YOU PRESS ‘9’ (as in Exercise 2).

HINT: first write a set of subroutines to program the Stepper Motor. Then, combine it with Exercise 2.

The basic behavior of stepper motors is described in the book. The following figure shows the actual schematic of the stepper circuit on our Project Board in the lab.



You do not have to necessarily understand the whole schematic, but you should be able to discern the signals involved in the stepper motor circuit, namely PT5, PT6 and PP5. Hence the programmer's model for our circuit involves two ports:

Port T

7	6	5	4	3	2	1	0	Bit
x	B	L	x	x	x	x	x	

where PT6 is an output sent to the Bottom Inductor coil of the stepper
where PT5 is an output sent to the Left inductor coil of the stepper
where x is not used in this circuit (but may be used in other circuits)

Port P

7	6	5	4	3	2	1	0	Bit
x	x	E	x	x	x	x	x	

where PP5 is the enable for the stepper
where x is not used in this circuit (but may be used in other circuits)

But how do we get the motor to move? You need to establish a sequence of values to be written to PT6 and PT5 by your program so as to induce magnetic field in a particular direction. The first thing you need to know is that the 74LS04 is an inverter. This means that we also have !PT5 and !PT6 (the inverted signals). If we now compare this circuit to the one in the textbook, we find that PT6 and !PT6 are the inputs to the left inductor coil (corresponding to PP3 and PP2 in the textbook) while PT5 and !PT5 are the inputs to the bottom inductor coil (corresponding to PP1 and PP0) in the textbook. Using these two inverted pairs (ie. two pins instead of four separate pins) means that control of our stepper motor is limited to full-step rotations, shown in Table 7.7. You must put all this information together and formulate the values required to send our stepper motor through a (counter-) clockwise rotation. More precisely, make your own version of Table 7.7 but with only two columns of PT6 and PT5.

With your output sequence for PT6/5 in hand, there are these further considerations:

1. You must allow some time between writing out the next number of the rotation sequence for the stepper motor to physically move. Put delays of about 10 ms between writes.
2. Each number in the rotation sequence cause a *step* movement but one step is not necessarily equal to ¼ turn. In fact, on our motor, there are about four-five steps per ¼ turn. Just use four – close enough.
3. ***Beware: Both the 7-segment and the stepper use Port T.***

You can program this exercise in Assembly OR C (or a mix of both).

5. [Bonus – 10 marks]
assign45.prj, assign45.c, assign45asm.s, DP256reg.s (provided), assign45.SRC, and assign45.s19

Put together Exercises 4a. and 4.b.

All the functions should be included now in the RTI and Keypad ISRs. You have to react to the user's request, set the LEDS through the timer, react to the commands, and integrate the opening/closing of the window to the system (in this case you must use the RTI to time the execution of the opening/closing of the window).

DO NOT WORRY ABOUT INTERRUPT INTERFERENCE.

Optional: use key "7" to restart the system. This will make easier to test different conditions without reloading the program.

Due Date for Part A: March 4th, 2011 @ 4:00 PM
Due Date for Part B: March 9th, 2011 @ 4:00 PM

Assignment 4 Marking Criteria:
ASSIGNMENT: 20 + 15 BONUS
RECOUP: 10 bonus
An assignment will receive **UNS** (UNSatisfactory 0) if it is late **OR** any of the directions are not followed (including late submission or printing rude messages).
Assignments will receive the marks specified on each exercise if it assembles and runs cleanly when run by the TA, AND subroutine policies have been followed.