Computer Science 230
Computer Architecture and Assembly Language
Fall 2023

Assignment 2

Due: Tuesday, October 31, 11:55 pm by submission to Brightspace
(Sorry for screwing up Hallowe'en plans...)
(Late submissions **not** accepted)

Programming environment

For this assignment you must ensure your work is executed correctly on Arduino boards in ECS 249. If you have installed Microchip Studio on your own computer, then you are welcome to do some of the programming work on your machine. However, please plan to spend a significant amount of time in the lab. I would strongly recommend completing the assignment as soon as you are able, as I anticipate the lab to be busy towards October 31 with all the students who normally procrastinate.

Individual work

This assignment is to be completed by each individual student (i.e., no group work). Naturally you will want to discuss aspects of the problem with fellow students, and such discussion is encouraged. **However, sharing code fragments is strictly forbidden without the express written permission of the course instructor (Baharloo).** If you are still unsure regarding what is permitted or have other questions about what constitutes an appropriate collaboration, please contact me as soon as possible. (Code-similarity analysis tools will be used to examine submitted work.) The URLs of significant code fragments you have found and used in your solution must be cited in comments just before where such code has been used.

Objectives of this assignment

- Write and use functions.
- Implement parameter passing using registers and stack.
- Implement return values using registers.
- Use stack frames where needed for parameter passing.
- Use the Arduino mega2560 board in the course lab to implement a text signaling display.

Semaphore signaling

One form of visually communicating a message involves transforming each letter of the message into a visible signal. That signal may be in the form of flags (as is used in *naval flag signaling*). Another is to configure a set of lights into suitably different patterns, and this is something we will do for this assignment using the six LEDs on the lab's Arduino Mega boards.

A video has been prepared demonstrating some working code for all five parts of this assignment:

`https://youtu.be/_tRcKbYSZlY`

The video starts, however, with a demonstration of a complete working assignment that is flashing the message `HELLOWORLD`. For each of the ten letters in this message you will see is a pattern of six LEDs (some on, some off). Even more subtly, there is also the possibility of changing the duration of the light pattern. Breaking this down even further, and representing an LED that is on by using "`o`" and an LEDS that is off by using "`.`", you will notice the following in that opening sequence of the video:

- "H": `..oo..` (short duration)
- "E": `oooooo` (long duration)
- "L": `o.o.o.` (long duration)
- "L": `o.o.o.` (long duration)
- "O": `.oooo.` (long duration)
- "W": `o.o...` (short duration)
- "O": `.oooo.` (long duration)
- "R": `oo..oo` (long duration)
- "L": `o.o.o.` (long duration)
- "D": `.....o` (long duration)

Note again that letters "H" and "W" *have a short duration*. The encoding for all 26 letters of the alphabet can be found at the end of the starter file provided to you for this assignment (`a2-signalling.asm`).

There is only one file for this assignment. Your work will be in five parts made up of six functions, ordered from easy to more difficult, with the assumption that you will complete earlier parts before attempting later parts. Although you do not need to write any functions above and beyond those listed below, you are not forbidden from doing so.

a) Write the function `set_leds`
b) Write the functions `fast_leds` and `slow_leds`
c) Write the function `leds_with_speed`
d) Write the function `encode_letter`
e) Write the function `display_message`

Part (a): Write function `set_leds`

```
set_leds:
   parameters: one in r16, pass-by-value
   return value: none
```

The parameter to `set_leds` determines which of the Arduino board's six LEDs to turn on (i.e., the ones which you would have used during some lab exercises this semester). For some of what appears below, you may find it helpful to review once again the materials provided for Lab #3.

Interpretation of the parameter's value by the function is quite straightforward. You'll notice that the six-LED board itself on the Arduino has numbers underneath each of the LEDs.

- bit 5 of `r16` controls the leftmost LED (i.e. LED `01`) – if the bit is set, the light is to be turned on, otherwise turned off ...
- bit 4 of `r16` controls the second-to-left LED (i.e. LED `02`) in a manner similar to that above ...
- bit 3 of `r16` controls the third-to-left LED (i.e. LED `03`) ...
- bit 2 of `r16` controls the third-to-right LED (i.e. LED `04`) ...
- bit 1 of `r16` controls the second-to-right LED (i.e. LED `05`) ...
- bit 0 of `r16` controls the rightmost LED (i.e. LED `06`) ...
- and finally, bits 7 and 6 of `r16` is ignored, regardless of value.

For example, a value in `r16` that would turn on the leftmost and rightmost LEDs is `0x21` (i.e `0b00100001`). A value of `0x00` would turn off all LEDs.

Also please note that this function is meant to turn on/off the LEDs regardless of any duration for this action. LEDs will stay on or stay off until changed by another call to `set_leds` with a suitably different parameter value. To obtain effects such as turning LEDs on for a fixed period of time, you will need to use extra code, and this you will write written later.

Some code is provided for you at the label `test_a` which you may use to try out your work after writing code for the function. The video described earlier in this document shows expected behavior of `test_a`. (Note that the delay functions used in this code are provided to you already: `delay_long` is about one second long, and `delay_short` is about one-quarter of a second long.)

Part (b): Write the functions `fast_leds` and `slow_leds`

```
fast_leds:
   parameters: one in ~~r16~~ r17, pass-by-value
   return value: none
```

```
slow_leds:
   parameters: one in ~~r16~~ r17, pass-by-value
   return value: none
```

The `fast_leds` function will turn on LEDs using the same pattern as described in part (a) but will leave them on for about one-quarter of a second before turning them off. Amongst other things, you must call your code for `set_leds` and use the various delay functions in the provided code (e.g., `delay_short`).

The `slow_leds` function will turn on the LEDs using the same pattern as described in part (b) but will leave them on for about one second before turning them off. Amongst other things, you must call your code for `set_leds` and use the various delay functions in the provided code (e.g., `delay_long`).

Some code is provided for you at the label `test_b` which you may use to try out your work after writing code for these two functions. The video described earlier in this document shows expected behavior of `test_b`.

Part (c): Write the function `leds_with_speed`

```
leds_with_speed:
   parameters: one byte, pushed onto the stack by the caller, pass-by-value
   return value: none
```

In the functions described within the previous two parts of the assignment, the byte provided as a parameter encoded the LEDs to be turned on or off. This is indicated by the way bits 5 through 0 of the parameter are set or unset. Bits 7 and 6 have been ignored until now. In this function, we'll begin to make use of those two left-over bits.

- If the two top-most bits are set, the LED pattern is to be on for about one second.
- If the two top-most bits are unset, the LED pattern is to be on for about one-quarter of a second.

As an example, to turn on all LEDs for one second, the value pushed onto the stack would be `0xFF` (i.e., `0b11111111`). However, to turn them all on for one-quarter of a second, the value pushed would be `0x3F` (`0b00111111`). *Do not worry about the other two possible patterns for the left-most two bits as those patterns may be ignored.*

Call your code for `fast_leds` and `slow_leds` as part of the implementation of this function.

Some code is provided for you at the label `test_c` which you may use to try out your work after writing code for these two functions. The video described earlier in this document shows the expected behavior of `test_c`.

Part (d): Write the function `encode_letter`

```
encode_letter:
   parameters: one byte, pushed onto the stack by the caller, pass-by-value
   return value: r25
```

This document began by describing a way of encoding letters as LED light patterns, with the example of "`HELLOWORLD`" given. Although you're not yet ready to display the whole message, your task for this part is to take a single upper-case letter and to convert it into the correct pattern (lights and duration).

Towards the bottom of `a2-signalling.asm` you will find the label `PATTERNS`, followed by an upper-case alphabet – one letter per line – with information on LED/duration encodings. For example, the first non-comment line below `PATTERNS` is:

```
.db "A", "..oo..", 1
```

which is to be interpreted as follows:

- This line is for the letter "A";
- LEDS 01, 02, 05 and 06 are off in this pattern;
- LEDS 03 and 04 are on in this pattern;
- The lighting pattern is to appear for about one second.

Right below appears this line:

```
.db "B", ".o..o.", 2
```

which is to be interpreted as follows:

- This line is for the letter "B";
- LEDS 01, 03, 04, and 06 are off in this pattern;
- LEDS 02 and ~~04~~ 05 are on in this pattern;
- The lighting pattern is to appear for about one-quarter of a second.

So on and so forth.

The only other entry that requires some special explanation is the very last. It appears here as something which you may choose to check – i.e. if you are looking for a letter,

but find the dash ("-") then something has gone wrong with the parameter value. (When testing your code, the evaluators will only use upper-case letters. However, you cannot be sure how you yourself will accidentally test your code!)

Given the letter pushed onto the stack, your function is to determine the encoding for the letter as would be given to `leds_with_speed` (i.e. a byte value), and this returned in register `r25`. (That is, *you are not* to call `leds_with_speed` from within `encode_letter`) Make use the functions you have written earlier in this assignment to complete the function, as well as other operations. For example, the value returned in `r25` for "A" would be `0b11001100` or `0xcc`; the value returned in `r25` for "B" would be `0b00010010` or `0x12`.

Some code is provided for you at the label `test_d` which you may use to try out your work after writing code for these two functions. The video described earlier in this document shows the expected behavior of `test_d`.

Part (e): Write the function `display_message`

```
display_message:
   parameters: byte address of message in program memory which is to be
         displayed; high byte of address in r25; low byte of address in r24;
         messages have no spaces and contain only upper-case letters;
         messages are terminated with null (ASCII code 0).

   return value: none
```

This last function of the assignment is meant to tie together all your work so far.

Again, towards the bottom of the assembly file provided to you are a few other labels. For example, there is this line:

```
WORD07: .db "THE", 0
```

where `WORD07` can be provided to the assembler as the address within program memory to the string "`THE`". Your function will move from letter to letter, encoding each in turn by calling `encode_letter` and using the return value as the parameter to call `leds_with_speed`.

Some code is provided for you at the label `test_e` which you may use to try out your work after writing code for these two functions. The video described earlier in this document shows the expected behavior of `test_e`.

What you must submit

- Your completed work in the single source code file (`a2-signalling.asm`); **do not change the name of this file!**
- Your work must use the provided skeleton `a2-signalling.asm`. Any other kinds of solutions will not be accepted.
- Nothing else is to be submitted – no ZIP files, no project files, no manifestos, no video rants, nothing else!

Evaluation

- 3 marks: Solution for `set_leds`
- 2 marks: Solution for `fast_leds` for `slow_leds`
- 4 marks: Solution for `leds_with_speed`
- 6 marks: Solution for `encode_letter`
- 5 marks: Solution for `display_message`

Therefore, the total mark for this assignment is 20.

Some of the evaluations above will also take into account whether or not submitted code is properly formatted (i.e., indenting and commenting are suitably used), and the file correctly named.