Coursework Assignment

4CCS1CS1 Computer Systems

Introduction

This is a summative coursework for CS1. It counts for 15% of your final module grade. The assignment provides you with the opportunity to apply your knowledge and skills that you've gained from previous labs to some new tasks. You will need to write an assembly program, which will then be submitted to KEATS as a single `.s` file.

You have just under two weeks to complete the assignment. The deadline for submission is Friday 24th November 18:00. The suggested time to spend on the coursework is 8–10 hours.

In the labs following submission of the coursework, you will review each others programs and provide feedback to each other. This peer review is a mandatory part of the assessment, and non-participation in the review activity will result in your coursework mark being capped at 40%.

1 Display your k-number (15 marks)

Using the circuit from Lab 5, you should write a program to display the digits of your King's **K-number** on the LEDs.

Have your program write out each digit of your K-number separately, writing the left-most *numerical* digit first. For example, if your K-number is **K1070542**, then your program will first write out a **1**, followed by a **0**, then a **7**, then a **0**, then a **5**, then a **4**, and finally a **2**. Each number should be displayed for 1 second.

2 Display your initials (20 marks)

You should now modify your program so that it also displays a binary encoding of your initials after it has finished displaying your K-number.

There are many ways to encode alphanumeric characters in binary, the most common is ASCII. However, we will use our own encoding of alphanumeric characters. We will assume an 'A' is the decimal value 1, a 'B' is 2, a 'C' is 3 and so on. In this encoding, 'Z' would be 26. Again, you can use the look-up table later in this document to find the equivalent binary values that you will display, and accompanying hexadecimal values.

You should also display a full stop character '.', which we will assume is encoded as the value 27, between your initials.

For example, Ada Lovelace's program would first display her K-number. The program would then display the value 1 (00001, representing 'a'), then the value 27 (11011, representing '.'), and then the value 12 (01100, representing 'l').

Decimal Digit	Hexademical Equivalent	Binary Number Representation							
0	0x00	0	0	0	0	0	0	0	0
1	0x01	0	0	0	0	0	0	0	1
2	0x02	0	0	0	0	0	0	1	0
3	0x03	0	0	0	0	0	0	1	1
4	0x04	0	0	0	0	0	1	0	0
5	0x05	0	0	0	0	0	1	0	1
6	0x06	0	0	0	0	0	1	1	0
7	0x07	0	0	0	0	0	1	1	1
8	0x08	0	0	0	0	1	0	0	0
9	0x09	0	0	0	0	1	0	0	1
10	0x0A	0	0	0	0	1	0	1	0
11	0x0B	0	0	0	0	1	0	1	1
12	0x0C	0	0	0	0	1	1	0	0
13	0x0D	0	0	0	0	1	1	0	1
14	0x0E	0	0	0	0	1	1	1	0
15	0x0F	0	0	0	0	1	1	1	1
16	0x10	0	0	0	1	0	0	0	0
17	0x11	0	0	0	1	0	0	0	1
18	0x12	0	0	0	1	0	0	1	0
19	0x13	0	0	0	1	0	0	1	1
20	0x14	0	0	0	1	0	1	0	0
21	0x15	0	0	0	1	0	1	0	1
22	0x16	0	0	0	1	0	1	1	0
23	0x17	0	0	0	1	0	1	1	1
24	0x18	0	0	0	1	1	0	0	0
25	0x19	0	0	0	1	1	0	0	1
26	0x1A	0	0	0	1	1	0	1	0
27	0x1B	0	0	0	1	1	0	1	1
		7	*6*	*5*	*4*	*3*	*2*	*1*	*0*

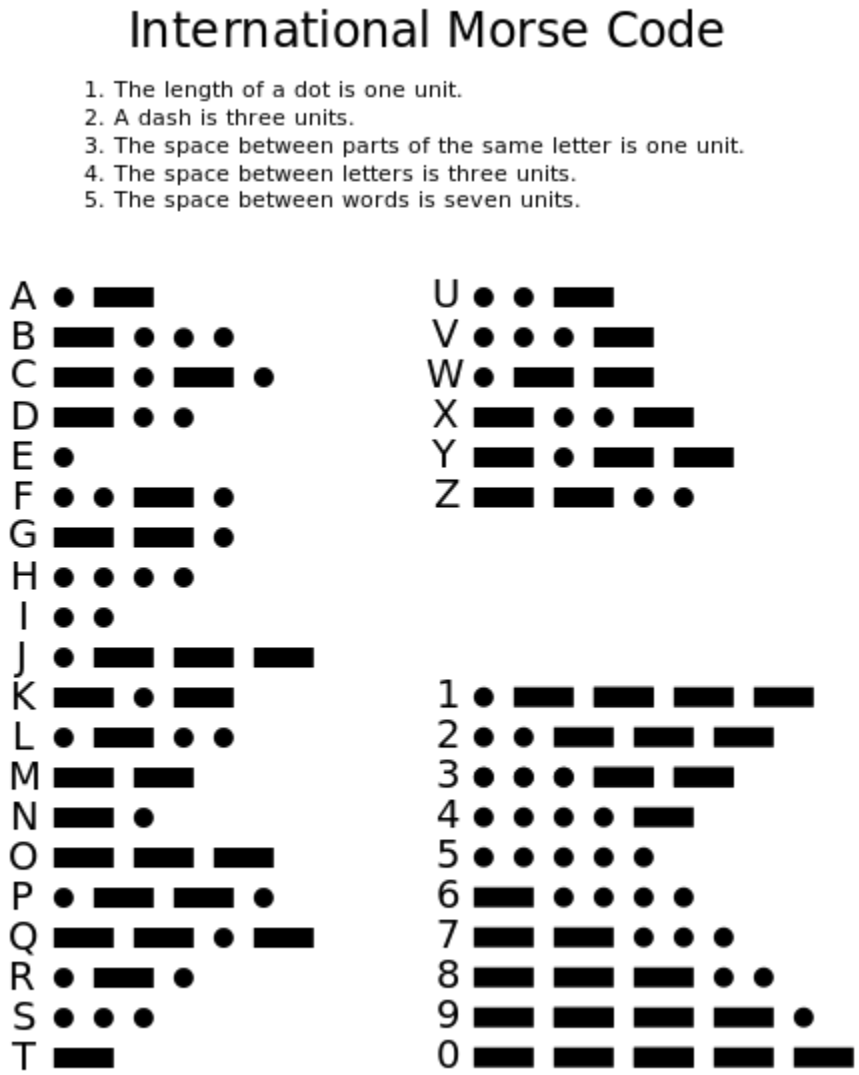
<bit position

Display Morse Code (10 marks)

You will now extend your program to communicate Morse code on the LEDs.

What is Morse code? Morse code is a method of transmitting text information as a series of on-off tones, lights, or clicks that can be directly understood by a skilled listener or observer without special equipment. (`https://en.wikipedia.org/wiki/Morse_code`).

Below is the International Morse Code Roman alphabet.



Your base program will blink a 3 letter sequence in Morse code on the LEDs. You three letter sequence is the first three letters of your first name.

- For example, Charles Babbage's code would be `CHA`.
- If your first name is less than three characters, you should use the first 3 characters of your first name concatenated with your surname. For example, Jo Rowling's code would be `JOR`.

So that we can perceive the Morse code, we will use a unit length of **200 milliseconds (ms)**. This means the duration of a *dot* is **200 ms**, and a *dash* is **600 ms**.

For example, if your sequence was `ABC`, then your program would run as follows:

1. Turn ON the LED for **200 ms** for the first *dot* of the letter `A`
2. Turn OFF the LED for **200 ms** for the inter-part space of the letter `A`
3. Turn ON the LED for **600 ms** for the first *dash* of the letter `A`
4. Turn OFF the LED for **600 ms** for the inter-letter space between the letters `A` and `B`
5. Turn ON the LED for **600 ms** for the first *dash* of the letter `B`
6. ... and so forth
7. Until the last *dot* of letter `C`
8. Turn OFF the LED for **1400 ms** for the inter-word space.
9. *Loop back to the beginning of the Morse code sequence*

3 Odd, Even, modulo 5 (15 marks)

Extend your program as follows.

- The Morse code sequence should loop 50 times (1–50).
- On odd iterations (1, 3, 5, ..., 49) your three characters should be displayed in their normal order.
  - *e.g.* `ABC`
- On even iterations (2, 4, 6, ..., 50) your three characters should be displayed in reverse order.
  - *e.g.* `CBA`

Using comments, you should explain how you have implemented the check of whether the iteration is even or odd.

Once you have this behaviour working, you should again extend your program as follows.

- On iterations that are divisible by 5 (5, 10, 15, ..., 50) your program should display a '5' after what would normally be displayed on that iteration.
  - *e.g.* `ABC5` or `CBA5`

4 Ping-pong (20 marks)

Once the Morse code sequence has terminated, your LEDs should display a repeating pattern.

You should use the LEDs to display a ping-pong like pattern, where only a single LED is on at a time, and it appears to move back and forth across the LEDS.

- $1000 \rightarrow 0100 \rightarrow 0010 \rightarrow 0001 \rightarrow 0010 \rightarrow 0100 \rightarrow 1000 \rightarrow ...$

It is left up to you to determine a suitable time to display each pattern for.

Submission instructions

- You should submit a single `.s` file named `assignment.s` via KEATS.
- **DO NOT** put your program in a .zip, .7z, or any other archive, **DO NOT** submit your program as a `.doc`, `.pdf`, or any other format other than `.s`.
- This coursework uses 'Model 4' for use of generative AI. That is, you can use generative AI to help you with the coursework. However:
  - I have tried to use it myself for completing the coursework, it was not very helpful. Do not expect generative AI to spit out complete assembly programs for you.
  - You should make it clear in the comments where and how you have used generative AI.
  - It is your responsibility to ensure any code produced by generative AI is 'fair use'.

Mark scheme

- There are 100 marks available in total.
- Marks for correctness are awarded according to the number of marks for that task (80 marks).
- Marks for readability and style are also awarded (20 marks).
  - Is your code structured and neat?
  - Is your code commented well?
  - Have you made use good use of the constructs we've covered in previous labs, e.g. functions, conditionals, and loops.