Note these very important points:

- On Vula you will find a template which you **must** use.
- Along with your assembly code you **must** submit a **Makefile**.
- When ‘make’ is run in your submission directory a .elf file should be produced.
- Do not launch GDB in a recipe.
- Submitting **code that does not compile will get a zero mark.**

Some later parts have dependencies on the earlier parts. Hence, the earlier parts need to be correctly solved before the later parts can be marked. Assume that all of the values given in the literals section will be modified (this includes PATTERNS, DATA, and SPECIAL_VALUE)

Part 1: (2)
Iterate through each individual byte in the DATA block, sign extend the byte and copy the word to the start of RAM. I.e:
- the first byte in the DATA block must be read sign extended, and stored to 0x20000000
- the second byte in the DATA block must be read sign extended, and stored to 0x20000004
- the third byte in the DATA block must be read sign extended, and stored to 0x20000008
…

When the automaker hits the label copy_to_RAM_complete it will verify contents of RAM
(Hint: See Resources> toolchain_guide.pdf, section 7.1.7 for a reminder on how to examine memory)

Part 2: (2)
Iterate through each word in the block of data in RAM and calculate the two values. The first value (x) represents how many words occurred in the data block whose value is numerically less than the word labeled SPECIAL_VALUE. The second value (y) represents how many words occurred in the data block whose value is equal to or numerically larger than the word labeled SPECIAL_VALUE. Find RESULT such that:

$$RESULT = x * y$$

After solving you should find 0xEB. Display RESULT on the LEDs. The marker will verify this when it hits the label display_result_done.

Part 3: (1)
TIM6 should be made to generate an interrupt every 0.8 seconds. The interrupt handler should increment the value displayed on the LEDs by 1. It should start counting from the value found and displayed in Part 2.

Part 4: (2)
When SW0 is held, held cycle the patterns in the PATTERNS block given in the template and display on the LEDs at a rate of 0.8 seconds. When the SW0 is released, the LEDs should go back to incrementing as in part 2. The first time SW0 is held it doesn't matter from which value the patterns transition starts, however if later when SW0 is held again, the value that was previously being cycled should display (i.e. it should not reset). Similarly, when SW0 is released, the value that was incrementing should be restored.

Part 5: (2)
When SW1 is held **immediately** sample POT0 and display a value on the LEDs proportional to what the POT is outputting. However, only display this value if it is numerically less than or equal to the value you found in part 2 (RESULT) otherwise display the value found in part 2. I.e display 0 if the pot output is 0 V. Display 0xFF if 3.3 V (except if what you will display is larger than RESULT, then display RESULT). Linear in between.

If SW1 == held:
    Sample POT0
    If POT0 value < RESULT:
        Display POT0 value
    Else if POT0 value >= RESULT:
        Display RESULT

NOTE: immediately means this action should not take longer than 0.1 seconds to occur, otherwise it is implemented incorrectly.

Part 6: (3)
When SW2 is held vary the delay generated by the interrupt handler such that:
- If no other switch is held, the interrupt handler should increment the value displayed on the LEDs at a rate proportional to the value coming out of POT1, linear between 0.8 and 1.6 seconds
- If SW0 is also held, the interrupt handler should cycle the patterns in the PATTERNS block given in the template on the LEDs at a rate proportional to the value coming out of POT1, linear between 0.8 and 1.6 seconds

Marked out of: 11

Available marks: 12