Name ______________________________________ Email__________________________

ECE 461/561 Sample Final Quizzes w/ Solutions

Spring 2021

Quiz 1: Examining Object Code and Speed Optimizations

This code uses bit-mapped data for a printable symbol ("glyph") to render it on the LCD. Each bit of the bit-map was already initialized to 1 if the pixel should be the foreground color, or 0 if the pixel should be the background color. The object code was generated with –O3 optimization.
Assume:

- CHAR_HEIGHT = 13
- CHAR_WIDTH = 8
- 0 < glyph_width < 9

Source Code:

```
// Initialization code deleted
for (row = 0; row < CHAR_HEIGHT; row++) {
    x_bm = 0; // x position within glyph bitmap, can span bytes
    do {
        bitmap_byte = *glyph_data;
        col = 0;
        num_pixels = 0;
        for (; (x_bm < glyph_width) && (col < 8); col++) {
            if (bitmap_byte & 0x01) // if pixel is to be set
                pixel_color = &fg;
            else
                pixel_color = &bg;
            // Inlined LCD_Write_Rectangle_Pixel
            b1 = (pixel_color->R&0xf8) | ((pixel_color->G&0xe0)>>5);
            b2 = ((pixel_color->G&0x1c)<<3) | ((pixel_color->B&0xf8)>>3);
            GPIO_Write(b1);
            GPIO_ResetBit(LCD_NWR_POS);
            GPIO_SetBit(LCD_NWR_POS);
            GPIO_Write(b2);
            GPIO_ResetBit(LCD_NWR_POS);
            GPIO_SetBit(LCD_NWR_POS);

            bitmap_byte >>= 1;
            x_bm++;
        }
        glyph_data++;       // Advance to next byte of glyph data
    } while (x_bm < glyph_width);
    if (x_bm < CHAR_WIDTH) {
        // fill in rest of cell with background color for narrow glyphs
        LCD_Write_Rectangle_Pixel(&bg, CHAR_WIDTH - x_bm);
    }
}
```

Name ____ Email___________________________

Object Code:

```
126:     bitmap_byte = *glyph_data;
0x01F1E LDR  r0,[sp,#0x24]
0x01F20 LDRB r0,[r0,#0x00]
0x01F22 STR  r0,[sp,#0x18]
127:     col = 0;
0x01F24 MOVS r0,#0x00
0x01F26 STR  r0,[sp,#0x08]
128:     num_pixels = 0;
0x01F28 STR  r0,[sp,#0x04]
129:     for (; (x_bm < glyph_width) && (col < 8);
col++) {
0x01F2A B    0x01F9A

130:        if (bitmap_byte & 0x01) // if pixel is
set
0x01F2C LDR  r0,[sp,#0x18]
0x01F2E LSLS r0,r0,#31
0x01F30 LSRS r0,r0,#31
0x01F32 CMP  r0,#0x00
0x01F34 BEQ  0x01F3A

131:           pixel_color = &fg;
0x01F36 LDR  r5,[pc,#184]  ; @0x01FF0
0x01F38 B    0x01F3C

132:        else
133:           pixel_color = &bg;
0x01F3A LDR  r5,[pc,#184]  ; @0x01FF4

134:        b1 = (pixel_color->R&0xf8) |
((pixel_color->G&0xe0)>>5);
0x01F3C LDRB r0,[r5,#0x00]
; Code at 0x01F3E to 0x01F8B implementing source code
lines 135-141 deleted from listing. It is all part of
same basic block as instruction at 0x01F3C.
135:        b2 = ((pixel_color->G&0x1c)<<3) |
((pixel_color->B&0xf8)>>3);
136:        GPIO_Write(b1);
137:        GPIO_ResetBit(LCD_NWR_POS);
138:        GPIO_SetBit(LCD_NWR_POS);
139:        GPIO_Write(b2);
140:        GPIO_ResetBit(LCD_NWR_POS);
141:        GPIO_SetBit(LCD_NWR_POS);
143:        bitmap_byte >>= 1;
0x01F8A LDR  r0,[sp,#0x18]
0x01F8C ASRS r0,r0,#1
0x01F8E STR  r0,[sp,#0x18]
144:        x_bm++;
145:     }
0x01F90 ADDS r0,r4,#1
0x01F92 UXTB r4,r0
145:     }
0x01F94 LDR  r0,[sp,#0x08]
0x01F96 ADDS r0,r0,#1
0x01F98 STR  r0,[sp,#0x08]

0x01F9A LDR  r0,[sp,#0x14]
0x01F9C CMP  r4,r0
0x01F9E BGE  0x01FA6

0x01FA0 LDR  r0,[sp,#0x08]
0x01FA2 CMP r0,#0x08
0x01FA4 BCC 0x01F2C

146:     glyph_data++;   // Advance to next byte
0x01FA6 LDR  r0,[sp,#0x24]
0x01FA8 ADDS r0,r0,#1
0x01FAA STR  r0,[sp,#0x24]
147:   } while (x_bm < glyph_width);
0x01FAC LDR  r0,[sp,#0x14]
0x01FAE CMP  r4,r0
0x01FB0 BLT  0x01F1E

; Basic block 9 starts at 0x01FB2
```

1. Control Flow Analysis: Identify each basic block in the object code above by listing the hexadecimal start address (shown in the listing) of its first and last instructions. Indicate the number of instructions per basic block in "Instruction Count". Then identify the immediate successor(s) of each basic block by number. Assume basic block 9 starts at address 0x01FB2.

Basic Block Number	Starting Address	Ending Address	Instruction Count	Basic Block Number of Immediate Successor(s)
1	0x00001F1E	0x00001F2A	7	6
2	0x00001F2C	0x00001F34	5	3, 4
3	0x00001F36	0x00001F38	2	5
4	0x00001F3A	0x00001F3A	1	5
5	0x00001F3C	0x00001F98	47	6
6	0x00001F9A	0x00001F9E	3	7, 8
7	0x00001FA0	0x00001FA4	3	2, 8
8	0x00001FA6	0x00001FB0	6	1,9

Name ______________________________________ Email__________________________

2. Basic block characteristics:
   a. Which basic block will be executed the most times?

   6

   b. Which basic blocks will execute the fewest times?

   1, 8

   c. Which basic block will dominate overall execution time? Assume each instruction takes one clock cycle to execute.

   Basic block 5, because it is by far the longest basic block in the innermost loop (47 instructions).

3. There are usually “runs” of multiple pixels of the same value (foreground or background) in most glyph bitmaps. Explain a way to optimize for these runs.

Precompute two versions of b1 and b2: one for the foreground color (fg_b1, fg_b2) and one for the background color (bg_b1 and bg_b2) . This code could be simplified further using two two-element arrays to hold the precomputed b1 and b2 values, though the speed would likely be similar.

```
uint8_t fg_b1, fg_b2, bg_b1, bg_b2;
fg_b1 = (fg.R&0xf8) | ((fg.G&0xe0)>>5);
fg_b2 = ((fg.G&0x1c)<<3) | ((fg.B&0xf8)>>3);
bg_b1 = (bg.R&0xf8) | ((bg.G&0xe0)>>5);
bg_b2 = ((bg.G&0x1c)<<3) | ((bg.B&0xf8)>>3);

for (row = 0; row < CHAR_HEIGHT; row++) {
   x_bm = 0; // x position within glyph bitmap, can span bytes
   do {
      bitmap_byte = *glyph_data;
      col = 0;
      num_pixels = 0;
      for (; (x_bm < glyph_width) && (col < 8); col++) {
         // Inlined LCD_Write_Rectangle_Pixel
         if (bitmap_byte & 0x01) { // if pixel is to be set
            GPIO_Write(fg_b1);
         }  else {
            GPIO_Write(bg_b1);
         }
         GPIO_ResetBit(LCD_NWR_POS);
         GPIO_SetBit(LCD_NWR_POS);
         if (bitmap_byte & 0x01) { // if pixel is to be set
            GPIO_Write(fg_b2);
         } else {
            GPIO_Write(bg_b2);
         }
         GPIO_ResetBit(LCD_NWR_POS);
         GPIO_SetBit(LCD_NWR_POS);

         bitmap_byte >>= 1;
         x_bm++;
      }
      glyph_data++;        // Advance to next byte of glyph data
   } while (x_bm < glyph_width);
```

Name ____

Quiz 2: Responsiveness

Consider a real-time system consisting of the following set of independent periodic tasks. Assume each task's deadline is equal to its period.

Task	Worst Case Execution Time C (ms)	Period T (ms)	Priority
Fee	3	31	B
Fi	1	41	C
Fo	4	59	E
Fum	11	26	A
Foo	15	53	D

4. Complete the table above by assigning a priority (high (A) to low (E)) to each task using the rate-monotonic approach.
5. Calculate the utilization of the task set.

$U = \frac{3}{31} + \frac{1}{41} + \frac{4}{59} + \frac{11}{26} + \frac{15}{53} = 0.89505$

6. Find the worst-case response time of the …
   a. **highest** priority task when using **preemptive** fixed-priority scheduling.
   11 ms

   b. **highest** priority task when using **non-preemptive** fixed-priority scheduling.
   11 ms + 15 ms = 26 ms

   c. **lowest** priority task when using **preemptive** fixed-priority scheduling.

$$R_i^0 = C_i + \sum_{j \in hp(i)} C_j = 4 + 3 + 1 + 11 + 15 = 34\ ms$$

$$R_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j = 4 + \left\lceil \frac{34}{31} \right\rceil 3 + \left\lceil \frac{34}{41} \right\rceil 1 + \left\lceil \frac{34}{26} \right\rceil 11 + \left\lceil \frac{34}{53} \right\rceil 15 = 48\ ms$$

$$R_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j = 4 + \left\lceil \frac{48}{31} \right\rceil 3 + \left\lceil \frac{48}{41} \right\rceil 1 + \left\lceil \frac{48}{26} \right\rceil 11 + \left\lceil \frac{48}{53} \right\rceil 15 = 49\ ms$$

$$R_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j = 4 + \left\lceil \frac{49}{31} \right\rceil 3 + \left\lceil \frac{49}{41} \right\rceil 1 + \left\lceil \frac{49}{26} \right\rceil 11 + \left\lceil \frac{49}{53} \right\rceil 15 = 49\ ms$$

Name ____

7. Does the utilization bound test show that this task set is **always schedulable** using fixed-priority **preemptive** scheduling with these priorities? Why or why not?

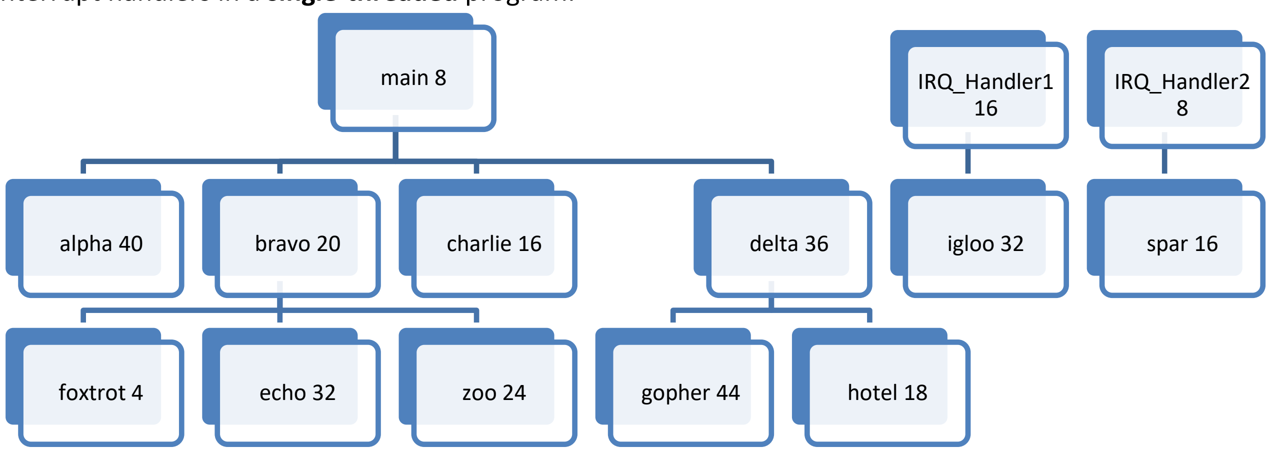
No, because $U_{max}(5) = 5(2^{1/5}-1)=0.74349$, and the task set utilization is $U = 0.89505$.

Name ____

Quiz 3: Memory Size

8. Some functions can be compiled to object code with a stack frame size of zero bytes. For this to happen, certain conditions are **necessary**. Note that these conditions are not **sufficient**; other conditions will also have to be met to get a zero stack frame size.
   a. A necessary condition for arguments to take no space on the stack is …
      that the arguments fit into registers r0 through r3.
   b. A necessary condition for automatic variables to take no space on the stack is …
      that the automatic variables fit into registers r0 through r3.
   c. A necessary condition for the link register to take no space on the stack is …
      that the function cannot call any other functions. As a result, the link register doesn't need to be pushed onto the stack.
   d. A necessary condition for the return value to take no space on the stack is …
      that the return value fits into registers r0 through r3.

9. The diagram below shows the maximum stack memory use in bytes for each function and two interrupt handlers in a **single-threaded** program.



   a. What is the sequence of function calls (ignoring interrupts) which causes the maximum stack depth, and what is that stack depth?
      main->delta->gopher, 88 bytes

   b. What is the maximum possible stack depth considering interrupts? Draw a picture of the call stack and label the stack frames (activation records) and their sizes. Assume interrupt handlers are not interruptable. State any other assumptions for full credit.

Description	RAM Used (Bytes)
igloo stack frame	32
IRQ_Handler1 stack frame	16
hardware-stacked register context	32
gopher	44
delta	36
main	8

168 bytes total

Name ____

10. Consider the following source code fragment which defines several static (not automatic) variables. Determine the amount of ROM and RAM used by each variable. Assume the compiler does not pack data structures or compress initialization data.

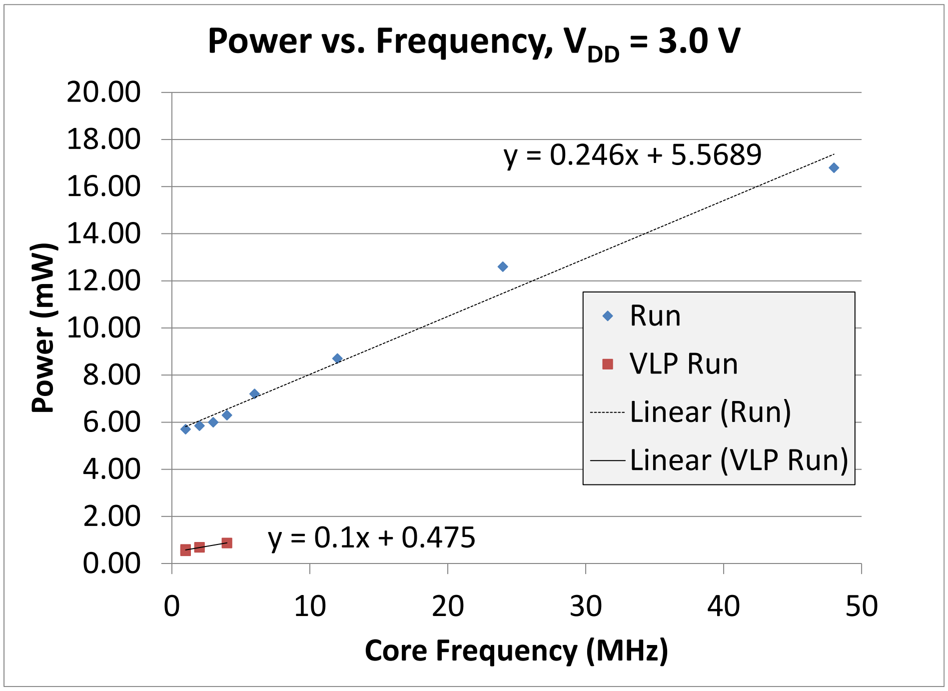
```
typedef struct {
     uint8_t R, G, B; // note: using 5-6-5 color mode for LCD.
} COLOR_T;
typedef struct {
     uint8_t FontID;
     uint8_t Orientation;
     uint16_t FirstChar;
     uint16_t LastChar;
     uint8_t Height;
     uint8_t Reserved;
} FONT_HEADER_T;
typedef struct {
     uint32_t Width:8; // pixels
     uint32_t Offset:24; // Offset from start of table
} GLYPH_INDEX_T;
uint8_t * font;
FONT_HEADER_T * font_header;
GLYPH_INDEX_T * glyph_index;
COLOR_T fg=BLACK, bg=GRAY;
uint8_t G_LCD_char_width, G_LCD_char_height;
const uint8_t * fonts[] = {Lucida_Console8x13, Lucida_Console12x19,
Lucida_Console20x31};
const uint8_t char_widths[] = {8, 12, 20};
const uint8_t char_heights[] = {13, 19, 31};
```

Variable	ROM Used (Bytes)	RAM Used (Bytes)
font	0	4
font_header	0	4
glyph_index	0	4
fg	4	4
bg	4	4
G_LCD_char_width	0	1
G_LCD_char_height	0	1
fonts	12	12
char_widths	3	0
char_heights	3	0

Name ____

Quiz 4: Power and Energy

Selected power vs. MCU core frequency characteristics of the KL25Z128 MCU are shown below.



11. Consider the MCU operating at 36 MHz in Run mode at 1.9 V.
    a. What is the power consumption?

$$P = \left(\frac{1.9}{3.0}\right)\left(36\ MHz * 0.246\frac{mW}{MHz} + 5.5689\ mW\right) = 9.13577\ mW$$

    b. How much energy is used per clock cycle?

$$E = \frac{9.13577\ mW}{36\ MHz} = 253.77\ pJ$$

12. Consider the MCU operating at 3 MHz in VLP Run mode at 3 V.
    a. What is the power consumption?

$$P = \left(3\ MHz * 0.1\frac{mW}{MHz} + 0.475\ mW\right) = 0.775\ mW$$

    b. How much energy is used per clock cycle?

$$E = \frac{0.775\ mW}{3\ MHz} = 258.33\ pJ$$