CMPE-250
Assembly and Embedded Programming
Spring 2021
Laboratory Exercise Seven:
Circular FIFO Queue Operations

This exercise investigates circular FIFO queue operations using the KL05. It requires polled serial I/O with the NXP Freedom Board KL05Z using the universal asynchronous receiver/transmitter (UART) module along with the character and string I/O subroutines developed in previous exercises. The objective of this exercise is to write subroutines to enqueue a character to a queue and to dequeue a character from a queue. An assembly language program is created to test these subroutines, and it is executed on the KL05Z board.

Prelab Work

Following the specifications below and the notes you have taken in lecture, write subroutines for circular FIFO queue initialization and operations. Use the queue management record structure in the class notes—including the field order and field offset names—so that the subroutines will work with any compatible circular FIFO queue. The subroutines must not use any temporary variables; the stack is the only area of memory that may be accessed other than the queue. Additionally, you must write the subroutines so that the no registers other than PSR and any output parameter(s) have changed values after return, (e.g., Dequeue uses R0 for an output parameter).

- InitQueue: Initializes the queue record structure at the address in R1 for the empty queue buffer at the address in R0 of size given in R2, (i.e., character capacity).
- Dequeue: Attempts to get a character from the queue whose record structure's address is in R1: if the queue is not empty, dequeues a single character from the queue to R0, and returns with the PSR C bit cleared, (i.e., 0), to report dequeue success; otherwise, returns with the PSR C bit set, (i.e., 1), to report dequeue failure.
- Enqueue: Attempts to put a character in the queue whose queue record structure's address is in R1: if the queue is not full, enqueues the single character from R0 to the queue, and returns with the PSR C bit cleared to report enqueue success; otherwise, returns with the PSR C bit set to report enqueue failure.
- PutNumHex: Prints to the terminal screen the text hexadecimal representation of the unsigned word value in R0. (For example, if R0 contains 0x000012FF, then 000012FF should print on the terminal. Note: 12FF would *not* be acceptable. *Do not use division to determine the hexadecimal digit values*—use bit masks and shifts.)
- PutNumUB: Prints to the terminal screen the text decimal representation of the unsigned byte value in R0. (For example, if R0 contains 0x003021101, then 1 should print on the terminal. Note: 001 would also be acceptable, as the text decimal representation of 0x01.) (Hint: use a mask to preserve only the least byte of the word in R0, and call your PutNumU subroutine from Lab Exercise Six.)

Application

This assignment tests queue enqueue and dequeue operations on a single circular FIFO queue and prints queue state information. A command menu allows the user to specify which queue operation will be performed, as described in the table below.

Command	Description
D or d	Dequeue a character from the queue.
E or e	Enqueue a character to the queue. (Prompt to enter a character, and then enqueue it.)
H or h	Help: list the commands.
P or p	Print the queued characters from the queue buffer to the terminal screen, in order from first in to last in.
S or s	Status: print the queue's current *InPointer*, *OutPointer*, and *NumberEnqueued*.

Program Specification

Write a program to implement the queue test command menu application that calls the subroutines from prelab work. The program must perform the following sequence of operations and must meet the requirements that follow.

1. Use Init_UART0_Polling from Lab Exercise Five to initialize the KL05 for UART0 polled serial I/O at 9600 baud using a format of eight data bits, no parity, and one stop bit—as in the previous lab exercises.
2. Use InitQueue from prelab work to initialize the program queue record structure (QRecord) for a queue buffer (QBuffer) of 4 characters, (i.e., bytes).
3. Output the character string below to the terminal.
```
Type a queue command (D,E,H,P,S):
```
4. Read a character typed on the terminal keyboard.
5. If the character is a lowercase alphabetic character, save the original character for possible output in step 7, and convert a copy to its uppercase equivalent as the command. For example, if the user typed d, save it and change a copy to D. (Having only one case of command characters requires fewer lines of code in the next step.)
6. If the character typed is not a valid command, (i.e., D, d, E, e, H, h, P, p, S, or s), ignore the character and repeat from step 4.
7. Otherwise, if the character is a valid command, print the character from step 4 to the terminal screen.
```
Type a queue command (D,E,H,P,S):p
```

8. Next, output a carriage return character (0x0D) and a line feed character (0x0A) to move the cursor to the next line of the screen, and perform the command, as follows.
   - `D` or `d`: Use `Dequeue` from prelab work to attempt to dequeue a character from the queue. If the queue was not empty, print the dequeued character followed by a colon, as shown below where "A" was dequeued.
     ```
     Type a queue command (D,E,H,P,S):d
     A:
     ```
     Otherwise, print "Failure:" if the queue was empty, as shown below
     ```
     Type a queue command (D,E,H,P,S):d
     Failure:
     ```
     In either case continue with step 9.
   - `E` or `e`: Prompt the user to enter a character to enqueue, as shown below.
     ```
     Type a queue command (D,E,H,P,S):e
     Character to enqueue:
     ```
     Read a character typed on the terminal keyboard, and then print it to the screen followed by a carriage return character (0x0D) and a line feed character (0x0A) to move the cursor to the next line of the screen, as shown below for a typed "A."
     ```
     Type a queue command (D,E,H,P,S):e
     Character to enqueue:A

     ```
     Next, use `Enqueue` from prelab work to attempt to enqueue the typed character to the queue. If the queue was not full, print "Success:" as shown below.
     ```
     Type a queue command (D,E,H,P,S):e
     Character to enqueue:A
     Success:
     ```
     Otherwise, print "Failure:" if the queue was full, as shown below.
     ```
     Type a queue command (D,E,H,P,S):e
     Character to enqueue:A
     Failure:
     ```
     In either case continue with step 9.
   - `H` or `h`: Print a help string that lists the valid commands. An example is shown below.
     ```
     Type a queue command (D,E,H,P,S):h
     D (dequeue), E (enqueue), H (help), P (print), S (status)
     ```
     Next, output a carriage return character (0x0D) and a line feed character (0x0A) to move the cursor to the next line of the screen, and repeat from step 3.
   - `P` or `p`: First print ">" as a beginning delimiter, and then print all the actively queued characters in the queue. The characters should be printed in order from the first queued to the last queued—the same order as they would be removed from the queue, but *without removing them from the queue*. If the queue is empty, no characters should print. Then print "<" as an ending delimiter, followed by a carriage return character (0x0D) and a line feed character (0x0A) to move the cursor to the next line of the screen, and repeat from step 3. The example output below shows the result for only one character "A" in the queue.
     ```
     Type a queue command (D,E,H,P,S):p
     >A<

     ```
   - `S` or `s`: Print "Status:" and then continue with step 9.

9. Report the status of the queue:
   a. Print "`In=0x`";
   b. Use `PutNumHex` from prelab work to print the hexadecimal representation of the value of the queue record's *InPointer*;
   c. Print "`   Out=0x`";
   d. Use `PutNumHex` from prelab work to print the hexadecimal representation of the value of the queue record's *OutPointer*;
   e. Print "`Num=`";
   f. Use `PutNumUB` from prelab work to print the value of the queue record's *NumberEnqueued*; and
   g. Print a carriage return character (0x0D) and a line feed character (0x0A) to move the cursor to the next line of the screen.

   Note that you should adjust the spacing in the output for readability and consistency among the commands, (i.e., `D`, `E`, and `S`), to produce the output shown below.

```
Type a queue command (D,E,H,P,S):S
 Status:  In=0x1FFFE100  Out=0x1FFFE100  Num=0
Enter a queue command (D,E,H,P,S):E
Character to enqueue:A
Success:  In=0x1FFFE101  Out=0x1FFFE100  Num=1
Enter a queue command (D,E,H,P,S):D
A:        In=0x1FFFE101  Out=0x1FFFE101  Num=0
Enter a queue command (D,E,H,P,S):D
Failure:  In=0x1FFFE101  Out=0x1FFFE101  Num=0
```

10. Repeat from step 3.

In addition, the program must meet these requirements.

- It must use subroutines from prelab work for queue operations and for hexadecimal and decimal number output. The following is a summary of the parameter specifications for each of these subroutines.
  - Dequeue
    - Input parameter:
      - (none)
    - Output parameters:
      - R0: character dequeued (unsigned byte ASCII code)
      - R1: queue record structure (unsigned word address)
      - C: dequeue operation status: 0 success; 1 failure (PSR bit flag)
  - Enqueue
    - Input parameter:
      - R0: character to enqueue (unsigned byte ASCII code)
    - Output parameters:
      - R1: queue record structure (unsigned word address)
      - C: enqueue operation status: 0 success; 1 failure (PSR bit flag)
  - InitQueue
    - Input parameters:
      - R0: queue buffer (unsigned word address)
      - R1: queue record structure (unsigned word address)
      - R2: queue capacity in bytes (unsigned byte value)
    - Output parameter:
      - (none)
  - PutNumHex
    - Input parameter:
      - R0: number to print in hexadecimal (unsigned word value)
    - Output parameter:
      - (none)
  - PutNumUB
    - Input parameter:
      - R0: number to print in decimal (unsigned byte value)
    - Output parameter:
      - (none)
- It must use subroutines from Lab Exercise Five for character I/O.
- It must use subroutines from Lab Exercise Six for string I/O.
- It must have properly commented subroutines, including a comment header block that includes these components.
  - Description of functionality
  - List of subroutines called
  - Lists of registers
    - Input parameters
    - Output parameters
    - Modified registers (after return, not preserved by subroutine)

Lab Procedure

1. Create a new directory and Keil MDK-ARM project for this exercise.
2. Write a properly commented and properly formatted KL05 assembly language program according to the preceding specification.
3. Build the program with Keil MDK-ARM to create a listing file and a linker map file.
4. Test your program. Make sure to test these cases, which must be demonstrated to the instructor.
   - Uppercase and lowercase commands
   - Commands with an empty queue
   - Commands with a partially full queue
   - Commands with a full queue
   - Commands with circular buffer operation
5. Examine the KL05 memory map produced for your program, and list the exact memory ranges, (i.e., the starting address and the address of the last byte), for these components:
   a. Executable code, (i.e., total `MyCode AREA`),
   b. Constants in ROM, (e.g., prompt string), and
   c. RAM used for the following:
      i. Queue record structure, and
      ii. Queue buffer.
6. Demonstrate steps 4 and 5 for your lab instructor, who will sign your grading sheet. Following demonstration, capture the terminal screen output to include and discuss in your documentation.
7. Submit your source (.s) file, listing (.lst) file, map (.map) file, and report to the myCourses assignments for this exercise.

Baseline Metrics

The metrics in the table below can be used to compare your solution to a baseline solution. They are provided solely for your information and personal curiosity about the efficiency of your solution. There are no grading criteria associated with how your code compares with them.

Language	**Routine**	**Instructions/LOC**	**Code Size (bytes)**
Assembly	main program[*]	114	304
C	main	63	376
Assembly	Dequeue	21	42
C	Dequeue	12	44
Assembly	Enqueue	22	44
C	Enqueue	12	44
Assembly	InitQueue	11	22
C	InitQueue	8	18
Assembly	PutNumHex	17	36
C	PutNumHex	4	84
Assembly	PutNumUB	5	12
C	PutNumUB	2	12
Assembly	**Total RO Size**	—	2144
C	**Total RO Size**	—	2220

[*]Not including instructions from provided program template.

Report

Write a report consisting of the following sections. Your writing should follow the rules of professional technical writing and should meet the specifications in “Laboratory and Report Guidelines” on myCourses.

- Abstract
- Procedure
- Results
  - ☞ The memory ranges specified in step 5 should be part of your report’s results section. (Remember to introduce and explain them in text rather than merely pasting them into the report.)
  - ☞ The screen capture from step 6 should be part of your report’s results section. (Remember to introduce and explain it in text rather than merely pasting it into the report.)
- Conclusion

Submission

myCourses Assignments
Separate assignment for each item below • Report (including cover sheet) • Source code (.s) • Listing (.lst) • Map (.map)

As specified in “Laboratory and Report Guidelines,” late submissions are penalized per day late, where “day late” is calculated by rounding submitted time to the nearest greater or equal integer number of days late, (i.e., ceiling function).

Note: To receive grading credit for this lab exercise you must earn points for *both* the demonstration *and* the report.