

CMPE-250 Assembly and Embedded Programming

Laboratory Exercise 7

Circular FIFO Queue Operations

By submitting this report, I attest that its contents are wholly my individual writing about this exercise and that they reflect the submitted code. I further acknowledge that permitted collaboration for this exercise consists only of discussions of concepts with course staff and fellow students. Other than code provided by the instructor for this exercise, all code was developed by me.

Andrei Tumbar

Submitted: 10-13-20

Lab Section: 5

Instructor: Gordon Werner

TA: Tianran Cui

Anthony Bacchetta

Lecture Section: 1

Lecture Instructor: Melton

Abstract

In this laboratory exercise, the functionality of queue structures was investigated. These queues were defined with a record structure and a queue buffer. The buffer was circular meaning when the end of the buffer area was filled, the next item would be enqueued into the start of the buffer. The buffer size of the queue was of arbitrary size defined at four bytes. Subroutines were created to perform operations on the queue structure as well as to print the status of the structure. The exercise was successful as the results indicated that the queue structure worked as intended.

Procedure

A command prompt was written that accepted commands: D, E, P, and S represented dequeue, enqueue, print, and status respectively. The status command printed current value of InPointer, OutPointer and NumEnqueued. InPointer is the next location in the queue buffer that an item is enqueued. OutPointer points to the next character that will be dequeued. NumEnqueued keeps track of how many items have been stored.

Because InPointer and OutPointer are pointers, convention is to print them in hexadecimal. To facilitate this, a PutNumHex subroutine was written to print hexadecimal numbers.

NumEnqueued is defined in the queue record structure as an 8-bit unsigned integer. To print this number, a PutNumUB subroutine was written to wrap functionality of PutNumU to only print a single byte.

The queue operations, Enqueue and Dequeue were implemented to work for an arbitrarily sized queue buffer. A set APSR C-flag was used in both subroutines to signify failure. Failure in Enqueue meant that the buffer was full whereas failure in Dequeue meant the queue was empty.

The prompt and status operations required parameterized string to be printed. Because of this, a printf-like subroutine was written to expand a format string and a number of associated arguments. The subroutines uses printf-like format string and takes contents of the stack to pass arguments. For example, a format string with stack contents:

```
1 format: Hello %s, %d, 0x%x, %b, %c\r\n\0
2 stack order of PUSH #'c', #25, 0x1ffe100, 1600, "World\0"
3
4 Output: Hello World, 1600, 0x1ffe100, 25, c
```

One thing to note is that the contents of the stack are read backwards meaning that they must be PUSH'ed in reverse order. The format string shown above illustrates every implemented format specifier. All format specifiers shown are equivalent to printf from glibc. Because the stack was used to pass arguments, this subroutine could not make changes to the stack and therefore had to store register values in a defined block of RAM.

printf was useful for debug and completing this exercise. This subroutine can easily be expanded to print different formats.


Results

The total size of the executable code as seen in the map file was 1080 bytes. The large size has mostly to do with the extra printf subroutine.

Constants in the Read-Only Memory (ROM) make up 248. This area consists of all format string and prompt strings used in this lab.

Finally Random-Access Memory (RAM) consists of 18 bytes for the queue structure, 2 bytes of padding, 4 bytes for the circular buffer, and finally 20 bytes used internally by the printf function. This makes up 44 bytes total used in RAM.

After the program was written and compiled, the executable was loaded into a KL-05 board to be tested.

 COM3 - PuTTY

```

Type a queue command (D,E,H,P,S):h
D (dequeue), E (enqueue), H (help), P (print), S (status)
Type a queue command (D,E,H,P,S):s
Status: In=0x1FFFFD14 Out=0x1FFFFD14 Num=0
Type a queue command (D,E,H,P,S):e
Character to enqueue:g
Success:      In=0x1FFFFD15 Out=0x1FFFFD14 Num=1
Type a queue command (D,E,H,P,S):p
>g<
Type a queue command (D,E,H,P,S):e
Character to enqueue:b
Success:      In=0x1FFFFD16 Out=0x1FFFFD14 Num=2
Type a queue command (D,E,H,P,S):p
>gb<
Type a queue command (D,E,H,P,S):d
g:      In=0x1FFFFD16 Out=0x1FFFFD15 Num=1
Type a queue command (D,E,H,P,S):p
>b<
Type a queue command (D,E,H,P,S):d
b:      In=0x1FFFFD16 Out=0x1FFFFD16 Num=0
Type a queue command (D,E,H,P,S):d
Failure:     In=0x1FFFFD16 Out=0x1FFFFD16 Num=0
Type a queue command (D,E,H,P,S):

```

Figure 1: Terminal screen capture

Figure 1 shows the output of the terminal after each prompt command was used in various steps. The queue contents are printed after command that modifies the queue. Figure 1 shows an enqueue of g and b followed by three dequeues. The third dequeue will print an error because the queue was empty.

Conclusion

This laboratory exercise was useful in exploring circular queue structures. It was also useful because it practiced the use of data structures and use of `MACRO` in assembly. The exercise was successful in teaching queue structures as a working one was implemented. This will be useful later when doing buffered I/O to improve performance. In addition to the requirements of this lab, the implementation of the `printf` subroutine will prove useful in later labs as the subroutine has many practical uses.