Keiran Berry

CENG320L

Lab 6 Report

24 October 2022

1. Lab Overview

The purpose of this lab was to write the enque and deque functions for a queue ADT. In doing this, I learned about the mangling of function names of class member functions as well as the “hidden parameter”, the pointer to the object being modified in the class function. In this case, x0 would point towards the beginning of the queue, and from there could be offset to look at items in the queue, the front index, end index, and number of items, which show up after the queue array.

I began by loading in the pointer as well as the item for either function into non-volatile registers, and then calling the isfull function (for enque) or the isempty function (for deque). After checking what the functions returned, I either returned 0 or continued on with the function. After doing either the enque or deque steps, I made sure to load in the new number of items as well as the new front/end index. Under specific conditions, the indexes may need to loop, which I added a branch statement for in my code. Enque returns 1 upon success, and deque returns the item upon success.

2. Bugs and Hurdles

This lab was fairly straightforward, and consisted of mostly converting C code to assembly language. I did mess up my stp and ldp statements just because I tried to do them

completely from memory, but that was an easy fix after looking at it on a new day. I also originally neglected to see that I should update the front index and end index, so I started getting some random numbers when the queue should have just simply been going from 1 to 99. A final hurdle which I encountered was using some x registers when I should have been using w registers. All in all, the only issues that I encountered with this lab were minor and easy fixes.

3. Results

This lab introduced me to working with class functions in assembly language, and gave me even more practice with gdb, debugging my code, as well as working to convert C code into assembly. I changed the function main in order to have the queue working with the i value instead of the random numbers, so that I would have an easier time seeing what was going on and know for sure if I had gotten the correct output. Once I had my enque working, the deque function became very self-explanatory, as it was largely the same thing with a few minor changes.

```
s101080740@george:~/CENG320/lab6/queue$ make
g++ -g -I. -c queue.cc
g++ -g -o queuetest main.o queue.o queue_asm.o
s101080740@george:~/CENG320/lab6/queue$ ./queuetest
Current queue:
1
2
3
4
5
6
7
8
9
10
Current queue:
11
12
13
14
15
16
17
18
19
20
Current queue:
21
22
23
24
25
26
27
28
29
30
```

C: > Users > 101080740 > Source > Repos > ceng320 > lab6 > queue_asm.S

```
/*****************************************************************
* enque: _ZN5queue5enqueEi
* deque: _ZN5queue5dequeERi
* isfull: _ZN5queue6isfullEv
* isempty: _ZN5queue7isemptyEv
*****************************************************************/


    .equ    MAX_QUEUE_SIZE, 52

    .equ    frontindex,     MAX_QUEUE_SIZE * 4
    .equ    endindex,       frontindex + 4
    .equ    nitems,         endindex + 4

    .global _ZN5queue5enqueEi

_ZN5queue5enqueEi:

    stp     x29, x30, [sp, #-16]!
    stp     x19, x20, [sp, #-16]!

    mov     x20, x0             //move the queue into a nonvolatile
    mov     x19, x1             //move item to a nonvolatile

    bl      _ZN5queue6isfullEv  //x0 is already the correct argument

    cmp     x0, xzr             //compare x0 to the zero register
```

C: > Users > 101080740 > Source > Repos > ceng320 > lab6 > queue_asm.S

```
    bl      _ZN5queue6isfullEv  //x0 is already the correct argument

    cmp     x0, xzr             //compare x0 to the zero register
    bne     queue_full

    ldr     w1, [x20, endindex] //get endindex
    mov     x2, x1              //copy it for use later

    lsl     x1, x1, 2           //endindex*4

    str     w19, [x20, x1]      //array[endindex] = item

    add     x2, x2, #1          //endindex++

    cmp     x2, MAX_QUEUE_SIZE  //if statement
    bge     loop_end_index

    str     w2, [x20, endindex]

    ldr     w4, [x20, nitems]
    add     x4, x4, #1          //nitems++
    str     w4, [x20, nitems]
    mov     x0, #1
    ldp     x19, x20, [sp], #16
    ldp     x29, x30, [sp], #16
    ret

```

C: > Users > 101080740 > Source > Repos > ceng320 > lab6 > queue_asm.S

```
49      ldp     x29, x30, [sp], #16
50      ret
51
52  queue_full:
53
54      mov     x0, #0              //return 0
55      ldp     x19, x20, [sp], #16
56      ldp     x29, x30, [sp], #16
57      ret
58
59  loop_end_index:
60
61      mov     x3, #0
62      str     x3, [x20, endindex] //endindex = 0
63
64      ldr     x4, [x20, nitems]
65      add     x4, x4, #1          //nitems++
66      str     x4, [x20, nitems]
67      mov     x0, #1
68      ldp     x19, x20, [sp], #16
69      ldp     x29, x30, [sp], #16
70      ret
71
72
73          .global _ZN5queue5dequeERi
74
75  _ZN5queue5dequeERi:
```

C: > Users > 101080740 > Source > Repos > ceng320 > lab6 > queue_asm.S

```
73      .global _ZN5queue5dequeERi
74
75  _ZN5queue5dequeERi:
76
77      stp     x29, x30, [sp, #-16]!
78      stp     x19, x20, [sp, #-16]!
79
80      mov     x19, x1         //load item into a less volatile register
81      mov     x20, x0         //load queue class
82
83      bl _ZN5queue7isemptyEv
84
85      cmp     x0, #1          //if(isempty())
86      beq     queue_empty
87
88      ldr     w1, [x20, frontindex]
89      mov     x2, x1
90
91      lsl     x1, x1, 2       //frontindex*4
92
93      ldr     w29, [x20, x1]
94      str     w29, [x19]      //item = array[frontindex]
95
96      add     x2, x2, #1          //frontindex++
97
98      cmp     x2, MAX_QUEUE_SIZE
99      bge     front_index_greater //if frontindex++ >= MAX_QUEUE_SIZE
```

C: > Users > 101080740 > Source > Repos > ceng320 > lab6 > ASM queue_asm.S

```
100
101     str     w2, [x20, frontindex]
102
103     ldr     w4, [x20, nitems]
104     sub     x4, x4, #1              //nitems--
105     str     w4, [x20, nitems]
106
107     mov     x0, x29                 //return item
108
109     ldp     x19, x20, [sp], #16
110     ldp     x29, x30, [sp], #16
111     ret
112
113 queue_empty:
114
115     mov     x0, #0                  //return 0
116     ldp     x19, x20, [sp], #16
117     ldp     x29, x30, [sp], #16
118     ret
119
120 front_index_greater:
121     mov     x3, #0
122     str     w3, [x20, frontindex]   //frontindex = 0
123
124     ldr     w4, [x20, nitems]
125     sub     x4, x4, #1              //nitems--
126     str     w4, [x20, nitems]
127
128     mov     x0, x29                 //return item
129
130     ldp     x19, x20, [sp], #16
131     ldp     x29, x30, [sp], #16
132     ret
```