Circular FIFO Queue Operation

Operating on a circular FIFO queue requires two structures in variable memory (RAM). First, the queue contents are stored in a queue buffer. Second, queue management data are stored in a queue record structure: *InPointer*, *OutPointer*, *BufferStart*, *BufferPast*, *BufferSize*, and *NumberEnqueued*. The first four fields in this record are addresses, which need one word (4 bytes) each. To support queue sizes up to 255, the last two fields can be unsigned numbers that require 1 byte each. Thus, the queue record structure will occupy 18 bytes in memory. The first set of EQUates below defines the address displacement of each field within the queue management record structure. The rest of the directives could be used to allocate structures for an 80-character queue.

```
;Management record structure field displacements
IN_PTR    EQU    0
OUT_PTR   EQU    4
BUF_STRT  EQU    8
BUF_PAST  EQU    12
BUF_SIZE  EQU    16
NUM_ENQD  EQU    17

; Queue structure sizes
Q_BUF_SZ  EQU    80        ;Room for 80 characters
Q_REC_SZ  EQU    18        ;Management record size

; Queue structures
QBuffer   SPACE  Q_BUF_SZ  ;Queue contents
QRecord   SPACE  Q_REC_SZ  ;Queue management record
```

To use the queue in a program, the QRecord structure must be initialized. Enqueue and Dequeue subroutines can then be used to put data into the queue and get data out of the queue (respectively) by passing a pointer to QRecord. Enqueue and Dequeue operate on the queue with only three parameters.

- R0: character enqueued or dequeued
- R1: pointer to QRecord
- C (flag of PSR): success′/failure of enqueue or dequeue

QStruct Initialization

For initializing an empty queue, *InPointer* and *OutPointer* both point to the beginning of QBuffer. *BufferStart* is the address of the beginning of QBuffer, and *BufferPast* is the first address past the end of the QBuffer. *BufferSize* is the size (capacity) of the QBuffer. For an empty queue, *NumberEnqueued* is 0. The following instructions initialize QRecord for an empty queue.

```
LDR   R0,=QBuffer
LDR   R1,=QRecord
STR   R0,[R1,#IN_PTR]
STR   R0,[R1,#OUT_PTR]
STR   R0,[R1,#BUF_STRT]
MOVS  R2,#Q_BUF_SZ
ADDS  R0,R0,R2
STR   R0,[R1,#BUF_PAST]
STRB  R2,[R1,#BUF_SIZE]
MOVS  R0,#0
STRB  R0,[R1,#NUM_ENQD]
```

Managing the Queue

Enqueue and Dequeue manage the queue through R1, which is a pointer to QRecord (the queue management record). The fields of QRecord contain both the pointers (*InPointer*, *OutPointer*, *BufferStart*, and *BufferPast*) and the values (*BufferSize* and *NumberEnqueued*) needed for Enqueue and Dequeue. These subroutines do not have direct access to QBuffer (the queue buffer). Before calling, (i.e., BL or BLX), Enqueue or Dequeue, R1 must contain the address of QRecord (e.g, `LDR R1,=QRecord`).

As an example of how Enqueue and Dequeue work with the queue through QRecord, suppose you need the address where the next character should be enqueued: *InPointer*. *InPointer* is a word value stored in QRecord at the address IN_PTR bytes away from the beginning address of QRecord. The following instruction would load R2 with *InPointer*.

```
LDR   R2,[R1,#IN_PTR]
```

As another example of how Enqueue and Dequeue work with the queue through QRecord, suppose you want to know how many items are currently in the queue buffer: *NumberEnqueued*. *NumberEnqueued* is a byte value stored in QRecord at the address NUM_ENQD bytes away from the beginning address of QRecord. The following instruction would load R3 with *NumberEnqueued*.

```
LDRB  R3,[R1,#NUM_ENQD]
```