;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; ;

; QUEUE ;

; Queue Routines ;

; EE/CS 51 ;

; Archan Luhar ;

; TA: Joe Greef ;

; ;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; QueueInit

;

; Description: This function is used to create a queue of a given length

; and given element size at a given address.

;

; Operation: This function writes the meta data of the queue in the first

; byte and three words of the queue: the size of each element,

; the max number of elements, the index of the head (0), and

; the count of elements in the queue also initialized to 0.

; The start of the queue elements would be the eigth byte.

;

; Arguments: AX - the length, max number of elements in the queue.

; SI - the location at which to initialize the the queue.

; BL - size of each element (0: byes, 1: words)

;

; Return Value: The address of the byte after the end of the queue is in AX.

;

; Local Variables: SI (increment to write metadata)

;

; Shared Variables: None.

; Global Variables: None.

;

; Input: None.

; Output: None.

;

; Error Handling: None.

;

; Algorithms: None.

;

; Data Structures: Cyclic array

;

; Registers Used: AX (return value)

;

; Stack Depth: 0

;

; Author: Archan Luhar

; Last Modified: 10/28/2013

;

;

; Pseudo Code

; -----------

; queue.size = size ? 2 : 1 ; set queue's size - word if nonzero, byte if 0

; queue.length = length ; set queue's length

; queue.head = 0 ; set queue's head index

; queue.count = 0 ; set queue's count of number of elements

;

; queueSize = length \* queue.size

; metadataSize = 7 bytes

; afterQueuePtr = queuePtr + metadataSize + queueSize

; return afterQueuePtr

; QueueEmpty

;

; Description: This function is used to see if a given queue is empty.

;

; Operation: This function simply looks at the word five bytes into

; the metadata which stores the count of elements in queue.

; Then it returns true if it is zero, else it returns false.

;

; Arguments: SI - the address of the queue.

;

; Return Value: ZF - 1 if empty, else 0.

;

; Local Variables: None.

;

; Shared Variables: None.

; Global Variables: None.

;

; Input: None.

; Output: None.

;

; Error Handling: None.

;

; Algorithms: None.

;

; Data Structures: Cyclic array

;

; Registers Used: ZF

;

; Stack Depth: 0

;

; Author: Archan Luhar

; Last Modified: 10/28/2013

;

;

; Pseudo Code

; -----------

; return count == 0

; QueueFull

;

; Description: This function is used to see if a given queue is full.

;

; Operation: This function simply looks at the word five bytes into

; the metadata. This word stores the num of elements in queue.

; If it equals the word stored at 1 byte into the metadata,

; the length of the queue, then it returns true, else false.

;

; Arguments: SI - the address of the queue.

;

; Return Value: ZF - 1 if full, else 0.

;

; Local Variables: None.

;

; Shared Variables: None.

; Global Variables: None.

;

; Input: None.

; Output: None.

;

; Error Handling: None.

;

; Algorithms: None.

;

; Data Structures: Cyclic array

;

; Registers Used: ZF

;

; Stack Depth: 0

;

; Author: Archan Luhar

; Last Modified: 10/28/2013

;

;

; Pseudo Code

; -----------

; return queue.count == queue.length

; Dequeue

;

; Description: This function returns the value at the head of the queue.

; It is a blocking function that waits until there is a value

; if initially the queue is empty.

;

; Operation: This function loops, waiting, until the queue is not empty.

; Then, it stores the head in AL if element size is byte.

; Else, element size is word so it stores the head in AX.

; It then decrements the count.

; And also it sets the head to (head + 1) mod (length - 1).

; The location to read the value would be

;

; Arguments: SI - the address of the queue.

;

; Return Value: AX if element size is word, else AL - the head of queue.

;

; Local Variables: None.

;

; Shared Variables: None.

; Global Variables: None.

;

; Input: None.

; Output: None.

;

; Error Handling: None.

;

; Algorithms: None.

;

; Data Structures: Cyclic array

;

; Registers Used: AX if element size is word, else AL.

;

; Stack Depth: 0

;

; Author: Archan Luhar

; Last Modified: 10/28/2013

;

;

; Pseudo Code

; -----------

; while (queue.count == 0): ; queue is empty

; continue loop

;

; returnVal = queue.queue[queue.headIndex \* queue.size]

; queue.headIndex = (queue.headIndex + 1) mod (queue.length - 1)

; queue.count--

; return returnVal

; Enqueue

;

; Description: This function pushes to the end of a given queue a given

; value.

; It is a blocking function that waits until the queue is

; not full to enqueue the value.

;

; Operation: This function loops, waiting, until the queue is not full.

; Then it increments the count.

; The tail index is just (head index + count) mod (length - 1)

; If element size is byte, it stores argument from AL at tail.

; Elese element size is word so it stores argument from AX

; at tail.

; The location to store would be start of queue elements +

; tail index \* element size.

;

; Arguments: SI - the address of the queue.

; AX if element size is word, else AL - value to enqueue

;

; Return Value: None.

;

; Local Variables: None.

;

; Shared Variables: None.

; Global Variables: None.

;

; Input: None.

; Output: None.

;

; Error Handling: None.

;

; Algorithms: None.

;

; Data Structures: Cyclic array

;

; Registers Used: None.

;

; Stack Depth: 0

;

; Author: Archan Luhar

; Last Modified: 10/28/2013

;

;

; Pseudo Code

; -----------

; while (queue.count == queue.length): ; queue is empty

; continue loop

; queue.count++

; tailIndex = (queue.headIndex + queue.count) mod (queue.length - 1)

; queue.queue[tailIndex \* queue.size] = value