Polled IO versus Interrupt Driven IO

- Polled IO – processor continually checks IO device to see if it is ready for data transfer
  - Inefficient, processor wastes time checking for ready condition
- Interrupt Driven IO – IO device interrupts processor when it is ready for data transfer
  - Processor can be doing other tasks while waiting for last data transfer to complete – very efficient.
  - All IO in modern computers is interrupt driven.

Interrupt Driven IO

- INTR input on x86 generates a hardware interrupt
- INTA# output used to acknowledge the interrupt – valid for 2 bus cycles
- On 2nd cycle, the x86 inputs the interrupt number from the D0-D7 of the data bus
  - Interrupt number * 4 is memory location of interrupt service routine vector
- External device must provide this number

8259A PIC

- 8259A Programmable interrupt controller external device used to provide 8 interrupt sources
- Multiple 8259A chips can be cascaded to provide up to 64 interrupt sources
- Initial PCs had two 8259A chips to provide IRQ0-IRQ15

Interrupt Priorities

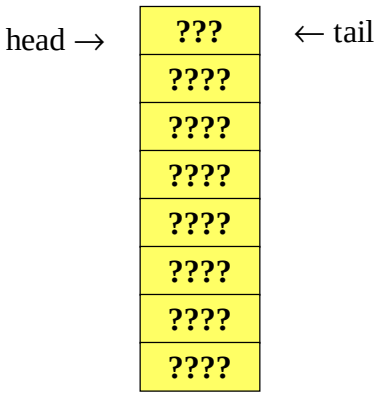
- A priority scheme determines what happens in the case of simultaneous interrupts
- A *fixed priority* scheme assigns priorities in a fixed order (ie. IRQ0 has highest priority, IRQ7 has lowest priority).
  - Can result in lowest priority devices not being serviced enough
- A *rotating priority* scheme rotates the highest priority among all sources by shifting the priorities
  - Highest priority is IRQ0, then after an interrupt, highest priority is changed to IRQ1, etc...
  - Results in more fair servicing of interrupts.

Interrupt Driven Input

- A *circular buffer* is most often used to handle interrupt driven INPUT.
- A circular buffer requires the following pointers
  - base address of memory buffer
  - head index (head pointer)
  - tail index (tail pointer)
  - size of buffer
- A circular buffer is simply another name for a *FIFO (First-In-First-Out)* buffer.
  - The name *circular buffer* helps to visualize the wraparound conditon

Circular buffer, 8 locations long

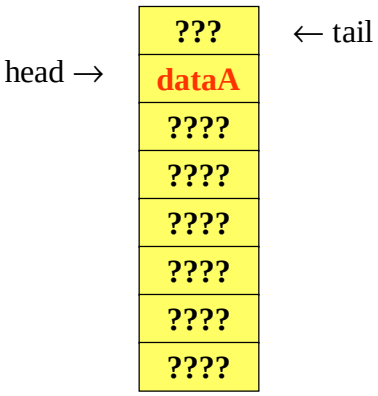
When buffer is empty, head = tail index



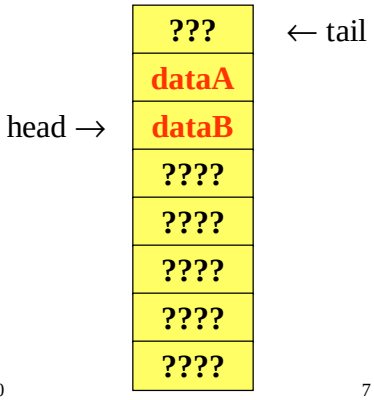
Circular buffer, write operation

Interrupt service routine places items in memory buffer by incrementing head index, then storing value

write a value



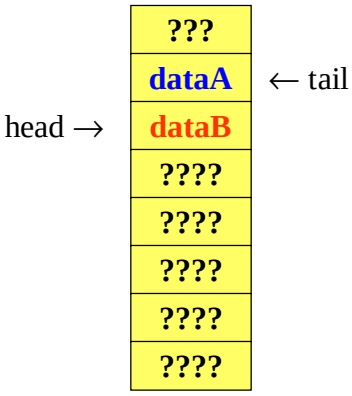
write a 2nd value



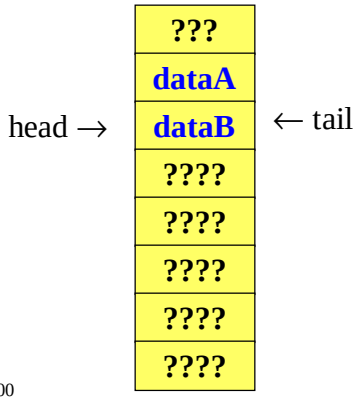
Circular buffer, read operation

Input function occassionally checks to see if head not equal to tail, if true, then read value by incrementing tail, then reading memory.

read dataA value

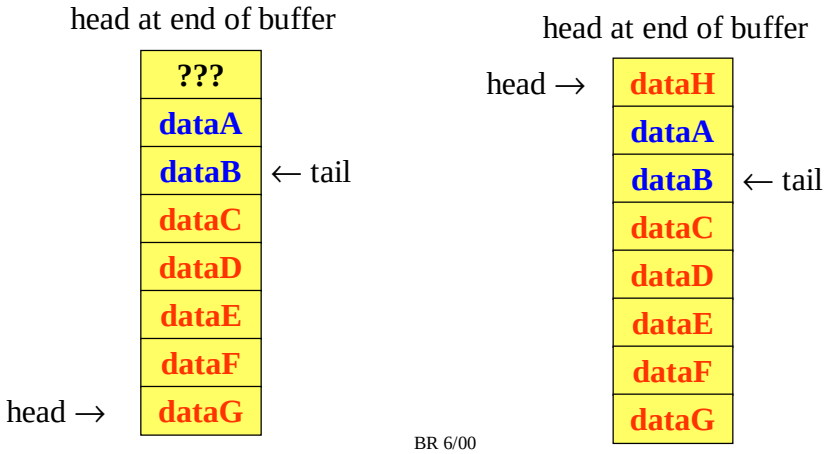


read dataB value



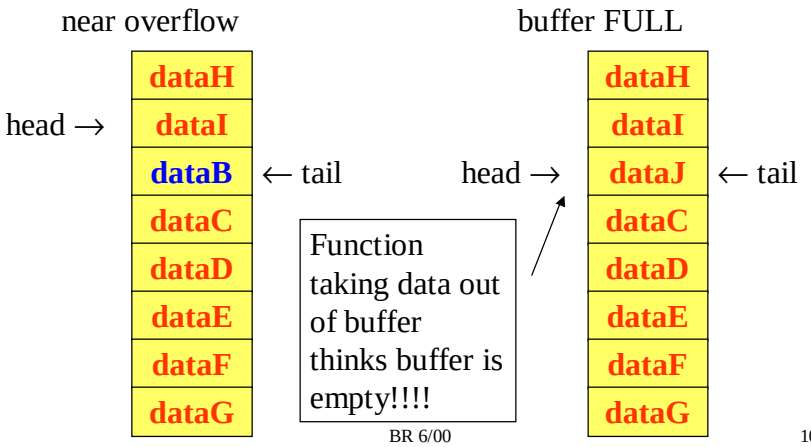
Circular buffer, wraparound

when head pointer gets to end of buffer, set back to top of buffer (wraparound)



Circular buffer, buffer FULL

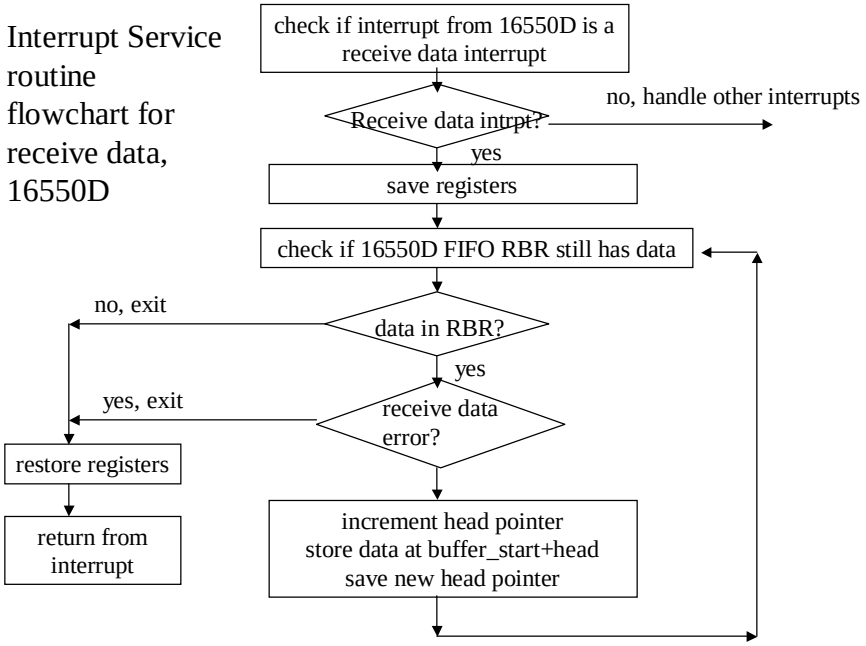
buffer FULL occurs if interrupt service routines increments head pointer to place new data, and head = tail!!!!



How to pick size of circular buffer?

- Must be big enough so that buffer full condition never occurs
- Routine that is taking data out of buffer must check it often enough to ensure that buffer full condition does not occur.
- Buffer must be big enough so that bursts of data into buffer does not cause buffer full condition.

Interrupt Service routine flowchart for receive data, 16550D



Subroutine flowchart for reading data out of the circular buffer.