Interfacing Techniques

ICT 2203 Computer Architecture

Partially based on
Embedded Systems Design: A Unified Hardware/Software Introduction, Vahid/Givargis (2000)

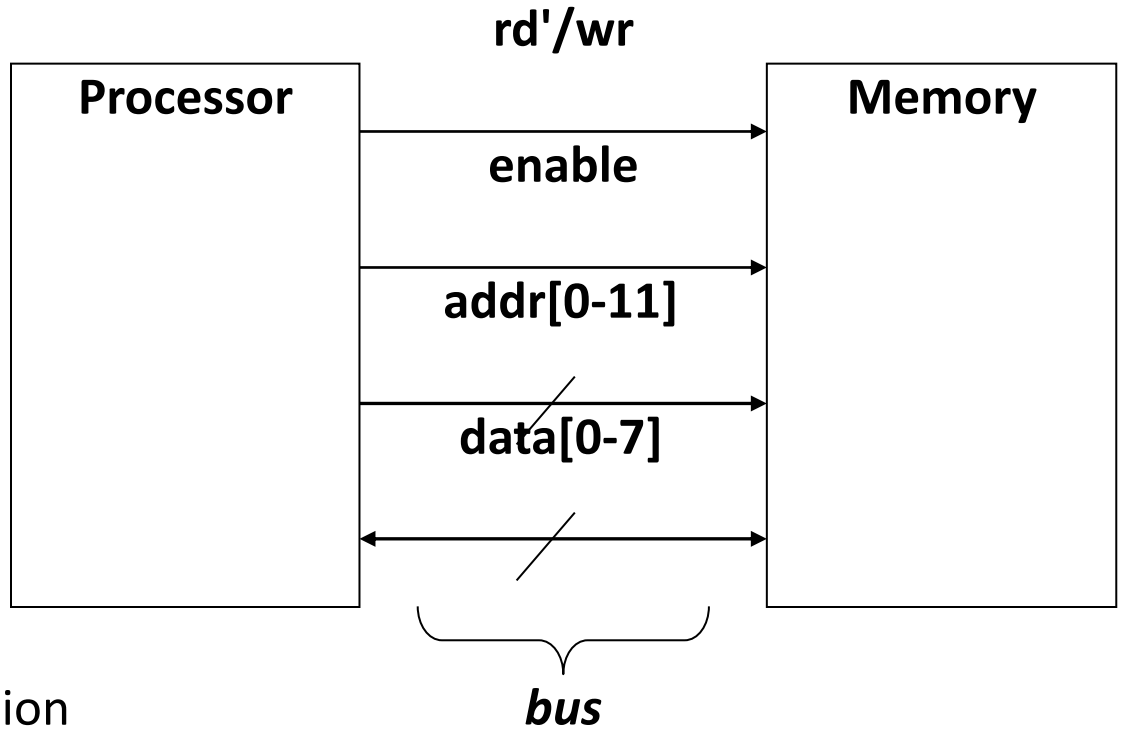
Introduction

- Transfer of data among processors and memories is known as **interfacing**.

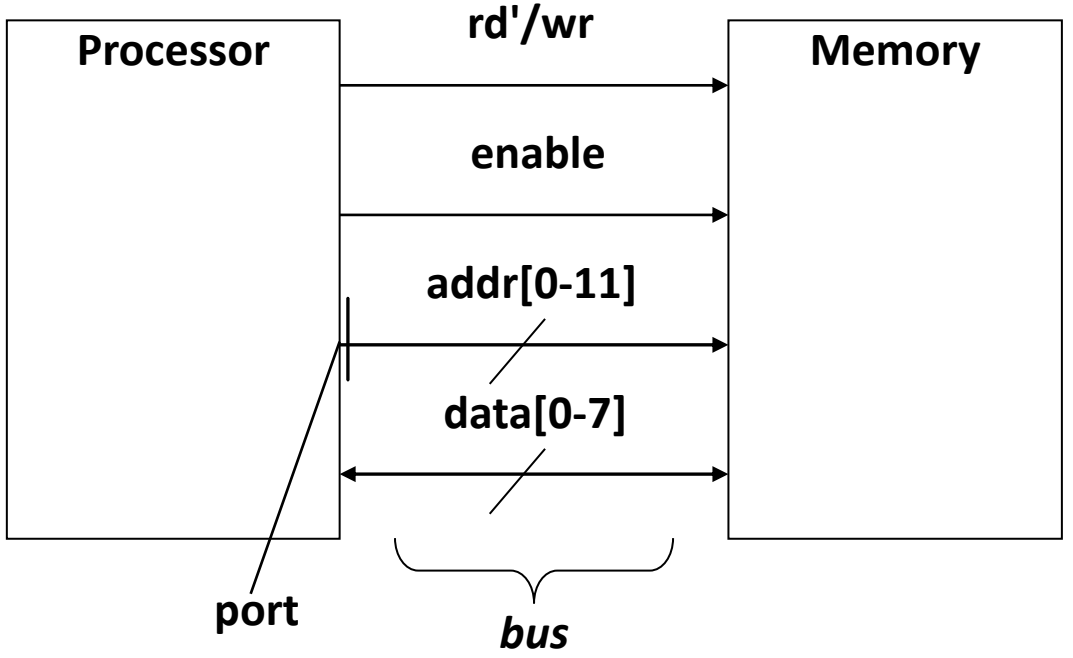
Processor	⟶	Process data
Memory	⟶	Storage
Buses	⟶	Communication

Bus

- Wires:
  - Uni-directional or bi-directional
  - One line may represent multiple wires
- Bus
  - Set of wires with a single function
    - Address bus, data bus
  - Or, entire collection of wires
    - Address, data and control
    - Associated protocol: rules for communication



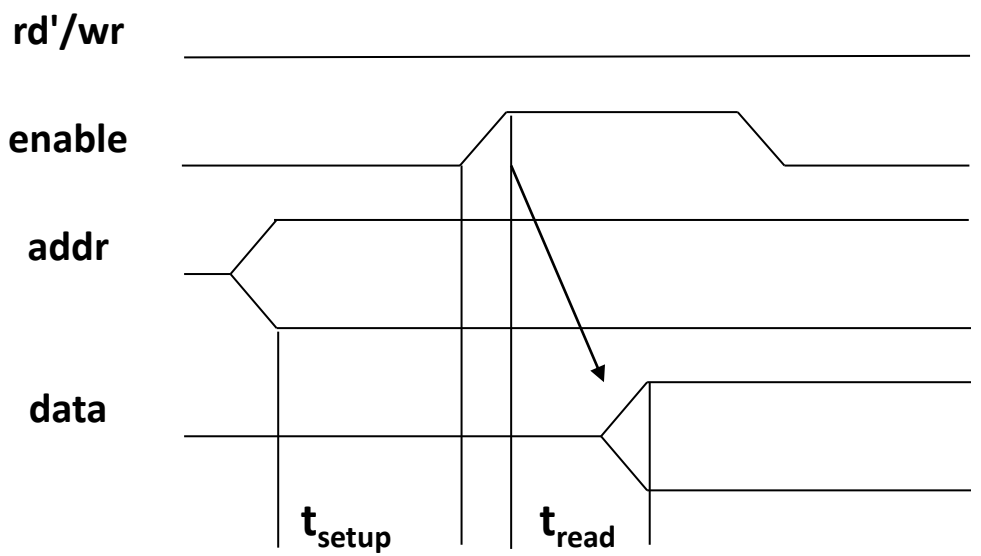
Ports



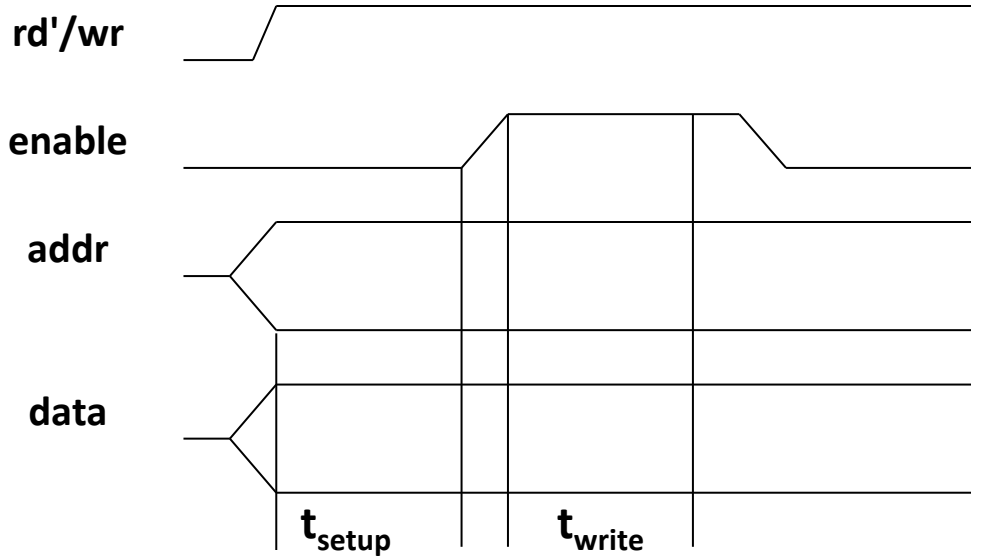
- Conducting device on periphery
- Connects bus to processor or memory
- Often referred to as a pin
  - Actual pins on periphery of IC package that plug into socket on printed-circuit board
  - Sometimes metallic balls instead of pins
  - Today, metal “pads” connecting processors and memories within single IC
- Single wire or set of wires with single function
  - e.g., 12-wire address port

Timing diagrams

- Most common method for describing a communication protocol
- Time proceeds to the right on x-axis
- **Control signal**: low or high
  - May be active low (e.g., go', /go, or go_L)
  - Use terms assert (active) and deassert
  - Asserting go' means go=0
- **Data signal**: not valid or valid
- Protocol may have subprotocols
  - Called bus cycle, e.g., read and write
  - Each may be several clock cycles
- Read example
  - rd'/wr set low,address placed on addr for at least tsetup time before enable asserted, enable triggers memory to place data on data wires by time tread



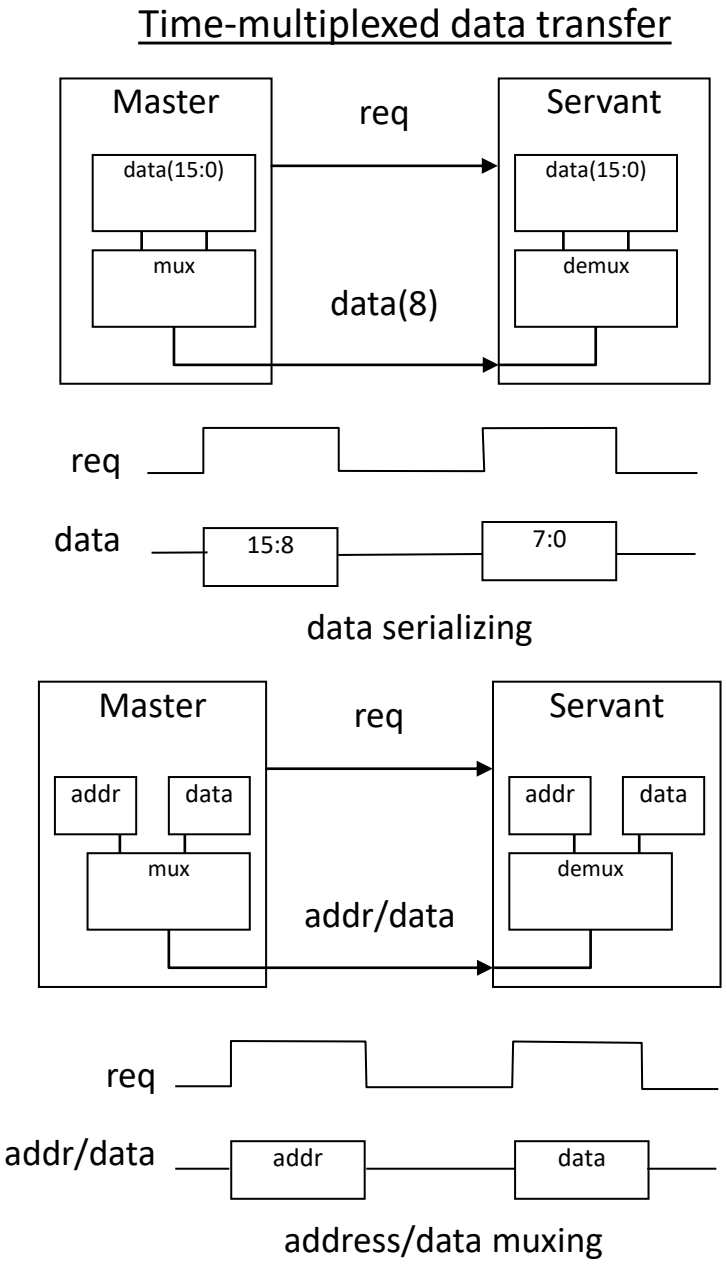
read protocol



write protocol

Basic protocol concepts

- Actor:
  - master initiates, servant (slave) respond
- Direction:
  - sender, receiver
- Addresses:
  - Special kind of data
  - Specifies a location in memory, a peripheral, or a register within a peripheral
- Time multiplexing:
  - Share a single set of wires for multiple pieces of data
  - Saves wires at expense of time

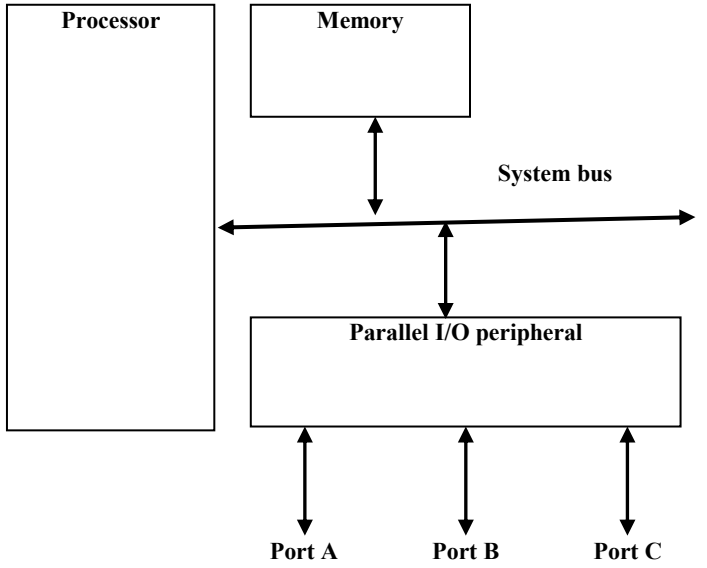


Microprocessor interfacing: I/O addressing

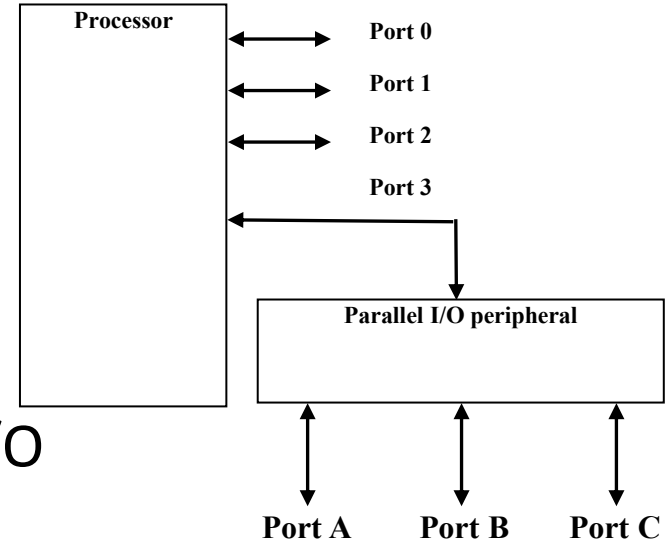
- A microprocessor communicates with other devices using some of its pins
  - **Port-based I/O** (parallel I/O)
    - Processor has one or more N-bit ports
    - Processor's software reads and writes a port just like a register
    - e.g., P0 = 0xFF; v = P1.2; -- P0 and P1 are 8-bit ports
  - **Bus-based I/O**
    - Processor has address, data and control ports that form a single bus
    - Communication protocol is built into the processor
    - A single instruction carries out the read or write protocol on the bus

Compromises/extensions

- Parallel I/O peripheral
  - When processor only supports bus-based I/O, but parallel I/O needed
  - Each port on peripheral connected to a register within peripheral that is read/written by the processor
- Extended parallel I/O
  - When processor supports port-based I/O but more ports needed
  - One or more processor ports interface with parallel I/O peripheral extending total number of ports available for I/O
  - e.g., extending 4 ports to 6 ports in figure



Adding parallel I/O to a bus-based I/O processor



Extended parallel I/O

Types of bus-based I/O

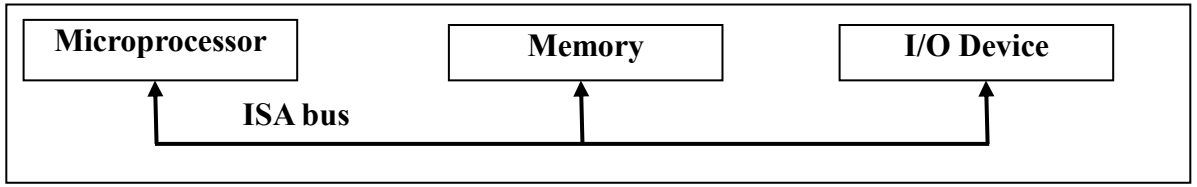
- Processor talks to both memory and peripherals using same bus – two ways to talk to peripherals
  - Memory-mapped I/O
    - Peripheral registers occupy addresses in same address space as memory
    - e.g., Bus has 16-bit address
      - lower 32K addresses may correspond to memory
      - upper 32k addresses may correspond to peripherals
  - Standard I/O (I/O-mapped I/O)
    - Additional pin (M/IO) on bus indicates whether a memory or peripheral access
    - e.g., Bus has 16-bit address
      - all 64K addresses correspond to memory when M/IO set to 0
      - all 64K addresses correspond to peripherals when M/IO set to 1

Memory-mapped I/O vs. Standard I/O

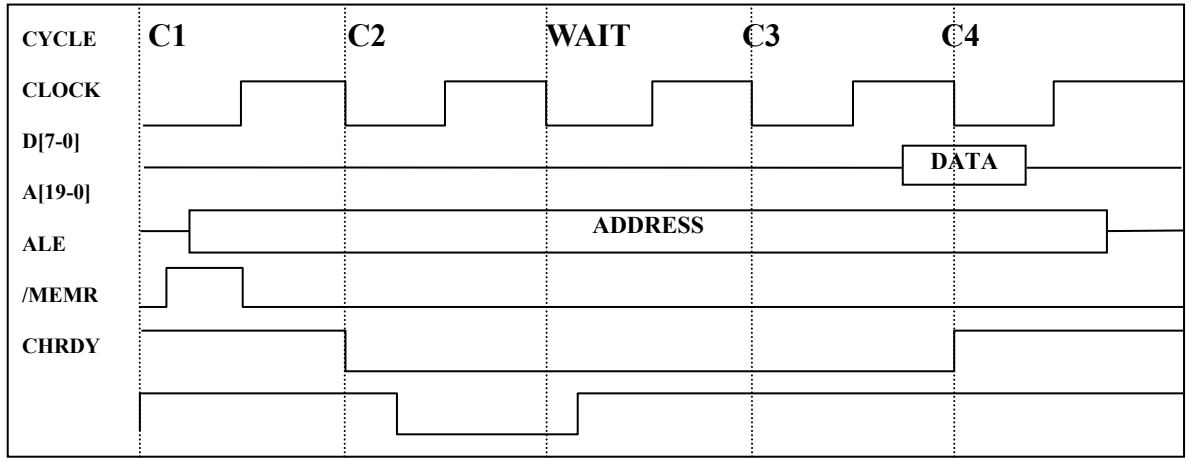
- Memory-mapped I/O
  - Requires no special instructions
    - Assembly instructions involving memory like MOV and ADD work with peripherals as well
    - Standard I/O requires special instructions (e.g., IN, OUT) to move data between peripheral registers and memory
- Standard I/O
  - No loss of memory addresses to peripherals
  - Simpler address decoding logic in peripherals possible
    - When number of peripherals much smaller than address space then high-order address bits can be ignored
    - Smaller and/or faster comparators

ISA bus protocol

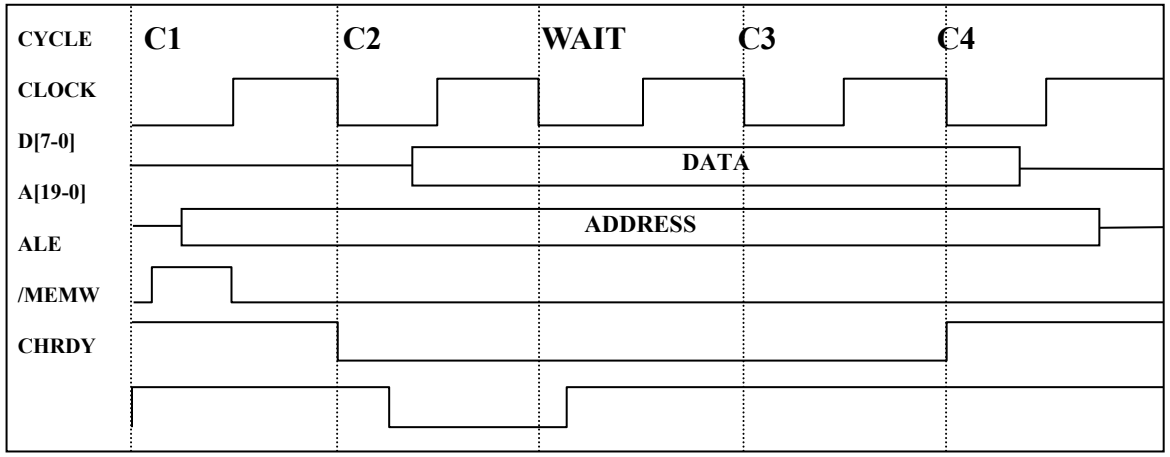
- ISA: Industry Standard Architecture
  - Common in 80x86's
- Features
  - 20-bit address
  - Compromise strobe/handshake control
    - 4 cycles default
    - Unless CHRDY deasserted – resulting in additional wait cycles (up to 6)



memory-read bus cycle

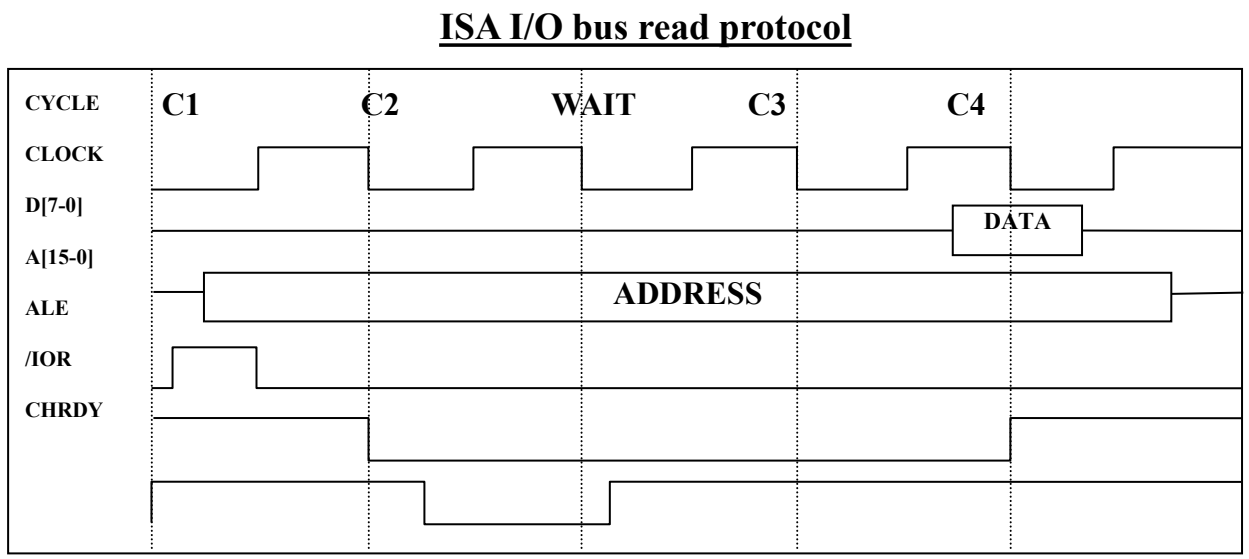


memory-write bus cycle



ISA bus

- ISA supports standard I/O
  - /IOR distinct from /MEMR for peripheral read
    - /IOW used for writes
  - 16-bit address space for I/O vs. 20-bit address space for memory
  - Otherwise very similar to memory protocol



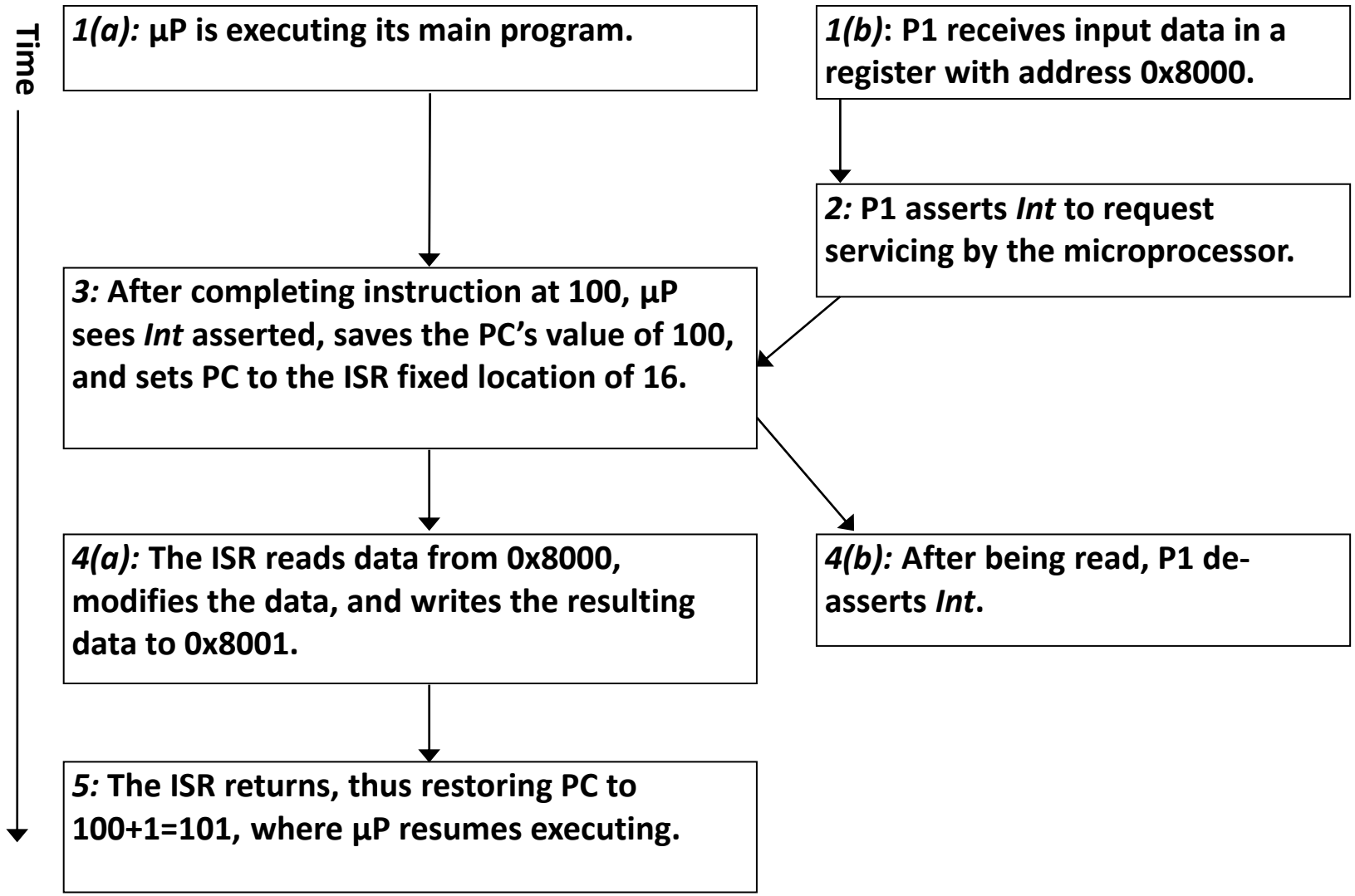
Microprocessor interfacing: interrupts

- Suppose a peripheral intermittently receives data, which must be serviced by the processor
  - The processor can *poll* the peripheral regularly to see if data has arrived
    - This is wasteful!
  - The peripheral can *interrupt* the processor when it has data
- Requires an extra pin or pins: Int
  - If Int is 1, processor suspends current program, jumps to an *Interrupt Service Routine*, or *ISR*
  - Known as interrupt-driven I/O
  - Essentially, “polling” of the interrupt pin is built-into the hardware, so no extra time!

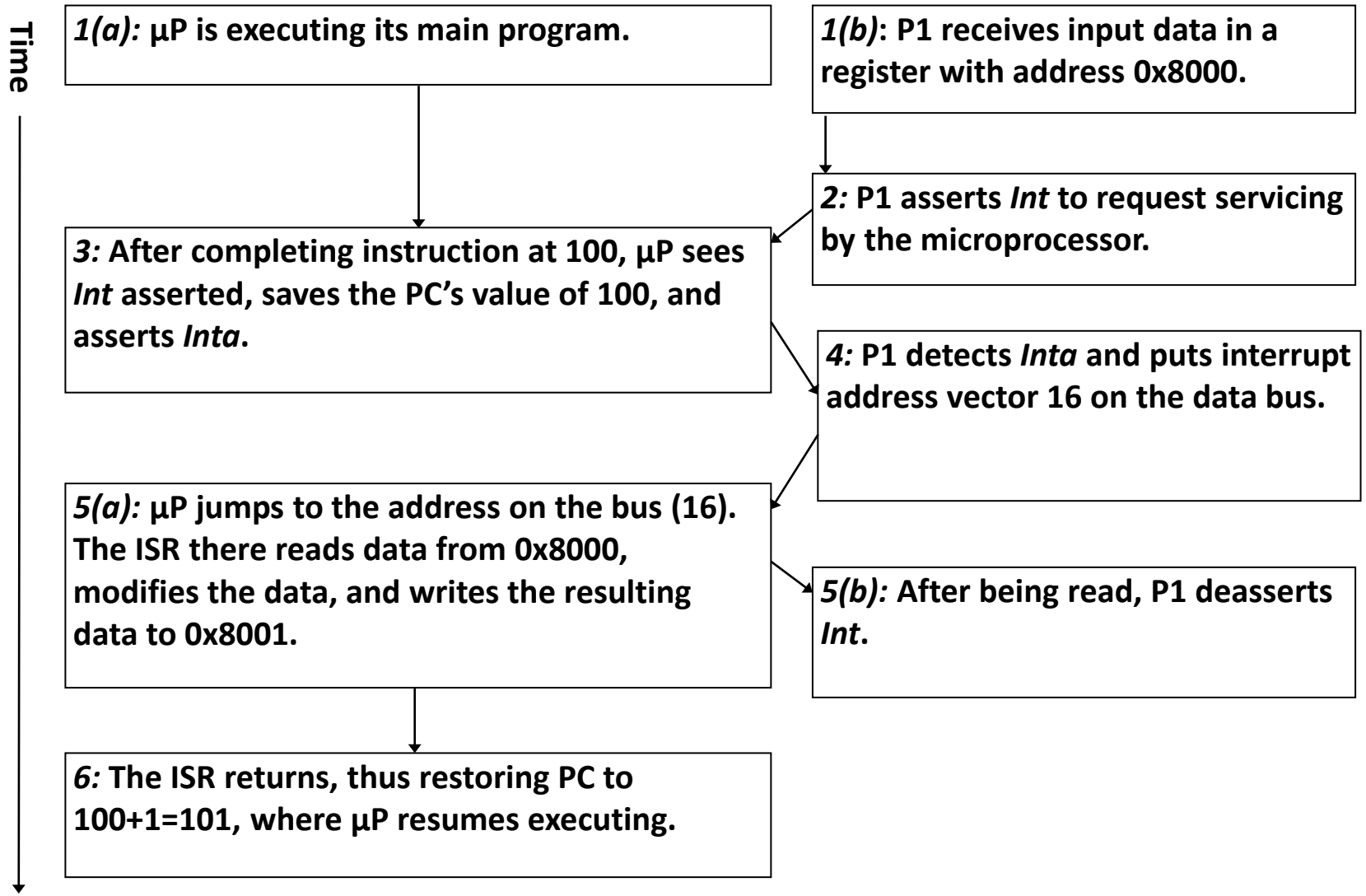
Microprocessor interfacing: interrupts

- What is the address (interrupt address vector) of the ISR?
  - Fixed interrupt
    - Address built into microprocessor, cannot be changed
    - Either ISR stored at address or a jump to actual ISR stored if not enough bytes available
  - Vectored interrupt
    - Peripheral must provide the address
    - Common when microprocessor has multiple peripherals connected by a system bus
  - Compromise: interrupt address table

Interrupt-driven I/O using fixed ISR location



Interrupt-driven I/O using vectored interrupt



Interrupt address table

- Compromise between fixed and vectored interrupts
  - One interrupt pin
  - Table in memory holding ISR addresses (maybe 256 words)
  - Peripheral doesn’t provide ISR address, but rather index into table
    - Fewer bits are sent by the peripheral
    - Can move ISR location without changing peripheral

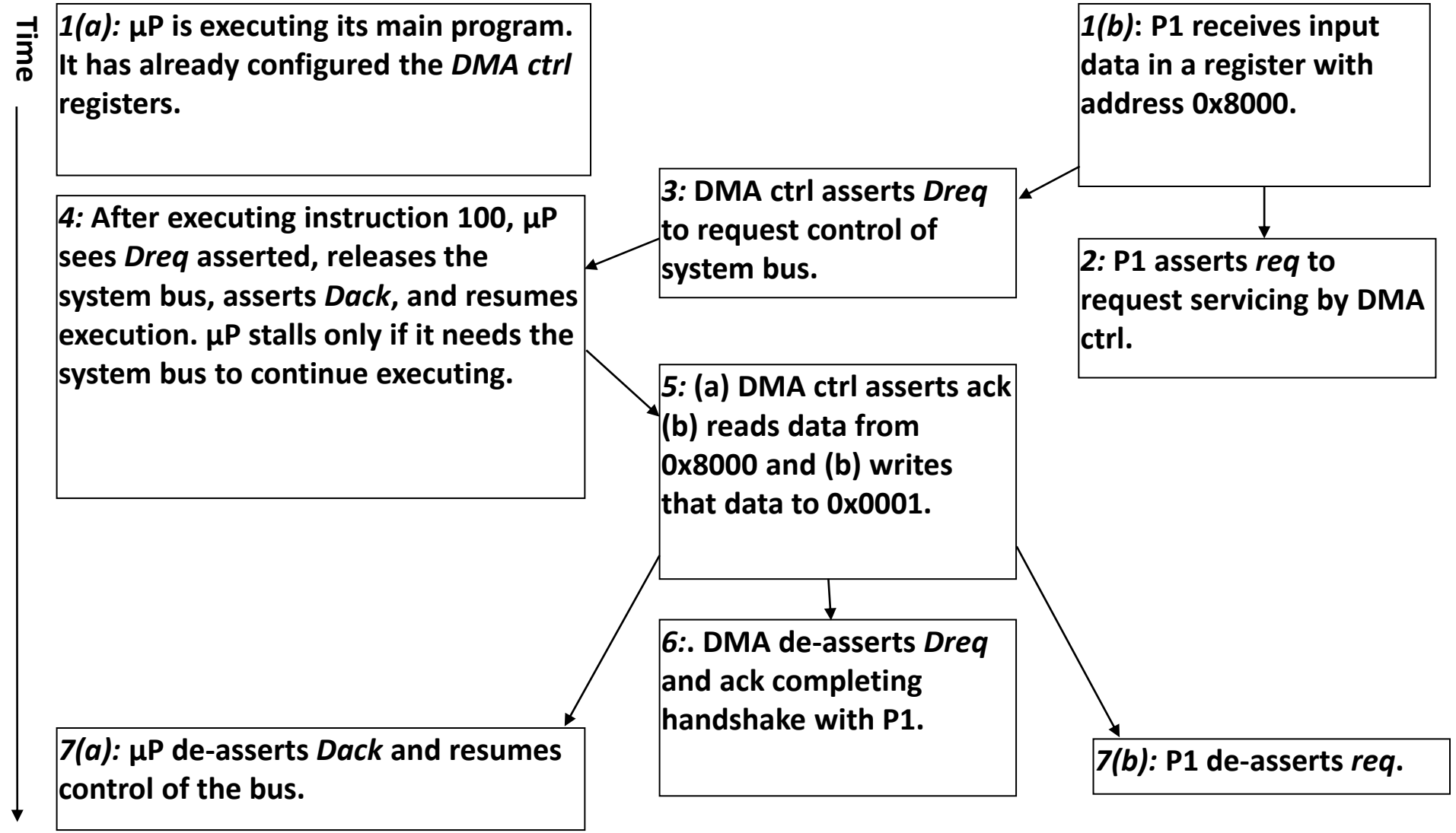
Additional interrupt issues

- Maskable vs. non-maskable interrupts
  - Maskable: programmer can set bit that causes processor to ignore interrupt
    - Important when in the middle of time-critical code
  - Non-maskable: a separate interrupt pin that can't be masked
    - Typically reserved for drastic situations, like power failure requiring immediate backup of data to non-volatile memory
- Jump to ISR
  - Some microprocessors treat jump same as call of any subroutine
    - Complete state saved (PC, registers) – may take hundreds of cycles
  - Others only save partial state, like PC only
    - Thus, ISR must not modify registers, or else must save them first
    - Assembly-language programmer must be aware of which registers stored

Direct memory access

- Buffering
  - Temporarily storing data in memory before processing
  - Data accumulated in peripherals commonly buffered
- Microprocessor could handle this with ISR
  - Storing and restoring microprocessor state inefficient
  - Regular program must wait
- DMA controller more efficient
  - Separate single-purpose processor
  - Microprocessor relinquishes control of system bus to DMA controller
  - Microprocessor can meanwhile execute its regular program
    - No inefficient storing and restoring state due to ISR call
    - Regular program need not wait unless it requires the system bus

Peripheral to memory transfer with DMA

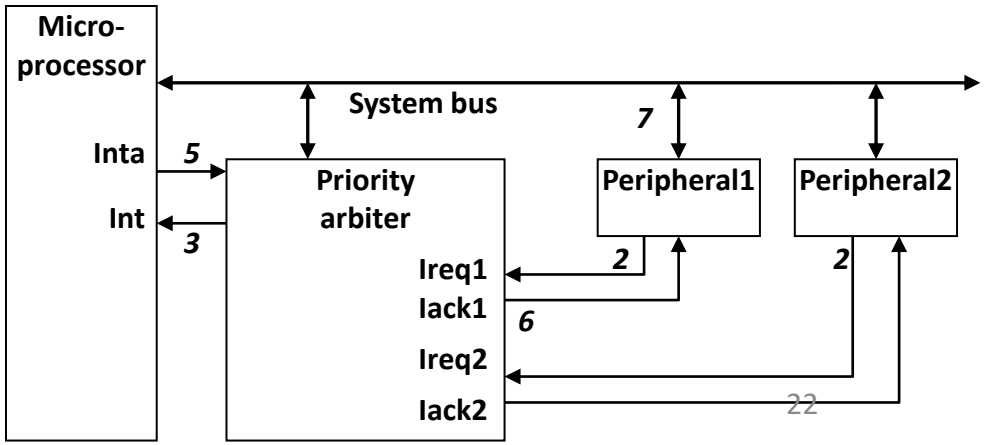


Arbitration

- A conflict may arise if the number of DMA controllers or other controllers or processors try to access the common bus at the same time, but access can be given to only one of those.
- Consider the situation where multiple peripherals request service from single resource simultaneously
  - e.g., microprocessor, DMA controller – which gets serviced first?
- To resolve these conflicts, Bus Arbitration procedure is implemented to coordinate the activities of all devices requesting memory transfers.

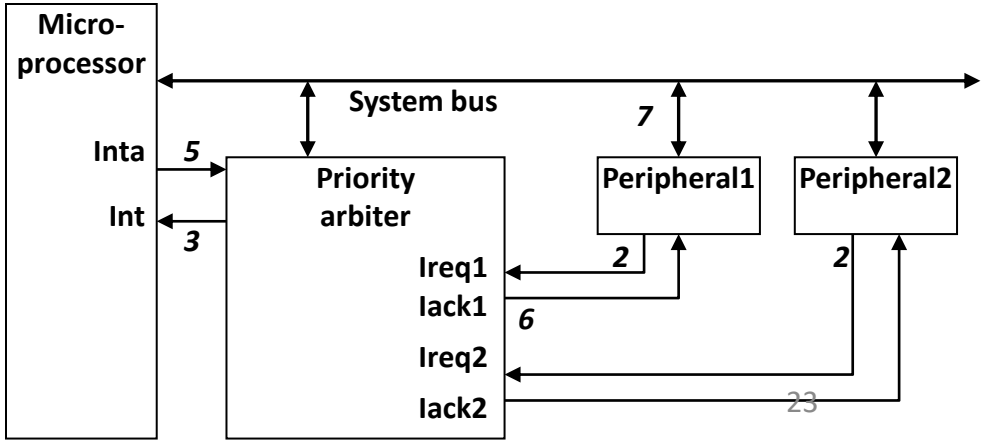
Arbitration: Priority arbiter

- Priority arbiter
  - Single-purpose processor
  - Peripherals make requests to arbiter, arbiter makes requests to resource
  - Arbiter connected to system bus for configuration only
- Types of priority
  - Fixed priority
    - each peripheral has unique rank
    - highest rank chosen first with simultaneous requests
    - preferred when clear rank differ between peripherals
  - Rotating priority (round-robin)
    - priority changed based on history of servicing
    - better distribution of servicing especially among peripherals with similar priority demands



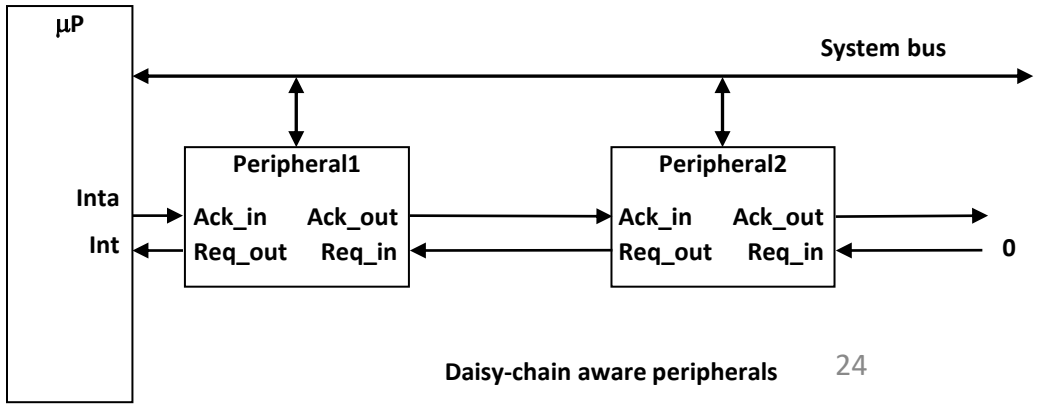
Arbitration: Priority arbiter

1. Microprocessor is executing its program.
2. Peripheral1 needs servicing so asserts Ireq1. Peripheral2 also needs servicing so asserts Ireq2.
3. Priority arbiter sees at least one Ireq input asserted, so asserts Int.
4. Microprocessor stops executing its program and stores its state.
5. Microprocessor asserts Inta.
6. Priority arbiter asserts Iack1 to acknowledge Peripheral1.
7. Peripheral1 puts its interrupt address vector on the system bus
8. Microprocessor jumps to the address of ISR read from data bus, ISR executes and returns (and completes handshake with arbiter).
9. Microprocessor resumes executing its program.



Arbitration: Daisy-chain arbitration

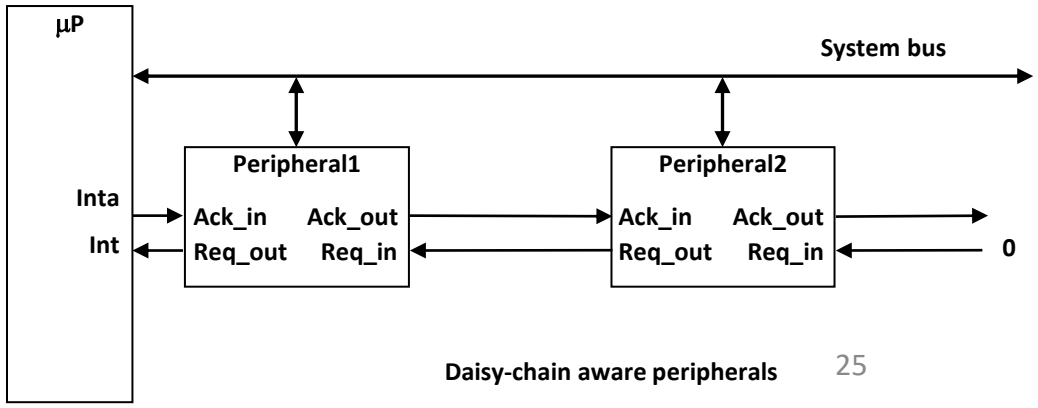
- Arbitration done by peripherals
  - Built into peripheral or external logic added
    - req input and ack output added to each peripheral
- Peripherals connected to each other in daisy-chain manner
  - One peripheral connected to resource, all others connected “upstream”
  - Peripheral’s req flows “downstream” to resource, resource’s ack flows “upstream” to requesting peripheral
  - Closest peripheral has highest priority



Daisy-chain aware peripherals

Arbitration: Daisy-chain arbitration

- Pros
  - Simplicity and Scalability
  - Easy to add/remove peripheral - no system redesign needed
  - The user can add more devices anywhere along the chain, up to a certain maximum value.
- Cons
  - One broken peripheral can cause loss of access to other peripherals
  - Propagation delay is arises in this method
  - Does not support rotating priority



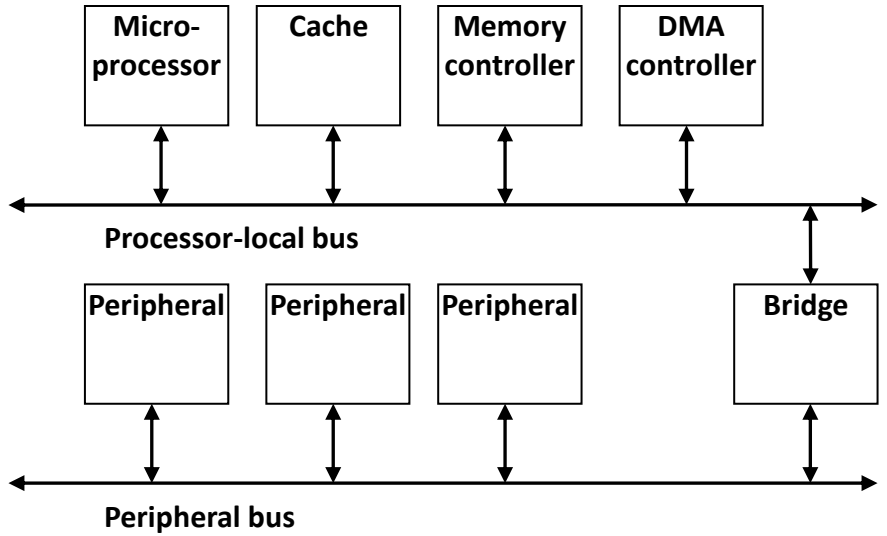
Daisy-chain aware peripherals

Network-oriented arbitration

- When multiple microprocessors share a bus (sometimes called a network)
  - Arbitration typically built into bus protocol
  - Separate processors may try to write simultaneously causing collisions
    - Data must be resent
    - Don’t want to start sending again at same time
      - statistical methods can be used to reduce chances
- Typically used for connecting multiple distant chips
  - Trend – use to connect multiple on-chip processors

Multilevel bus architectures

- Don’t want one bus for all communication
  - Peripherals would need high-speed, processor-specific bus interface
    - excess gates, power consumption, and cost; less portable
  - Too many peripherals slows down bus
- Processor-local bus
  - High speed, wide, most frequent communication
  - Connects microprocessor, cache, memory controllers, etc.
- Peripheral bus
  - Lower speed, narrower, less frequent communication
  - Typically industry standard bus (ISA, PCI) for portability
- Bridge
  - Single-purpose processor converts communication between busses



Advanced communication principles

- Parallel communication
  - Physical layer capable of transporting multiple bits of data
- Serial communication
  - Physical layer transports one bit of data at a time
- Wireless communication
  - No physical connection needed for transport at physical layer

Advanced communication principles

- Layering
  - Break complexity of communication protocol into pieces easier to design and understand
- Lower levels provide services to higher level
  - Lower level might work with bits while higher level might work with packets of data
- Physical layer
  - Lowest level in hierarchy
  - Medium to carry data from one actor (device or node) to another

Next:

Communication