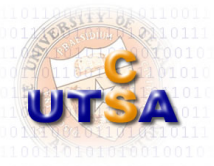
CS4833: Embedded Systems

I/O Interface

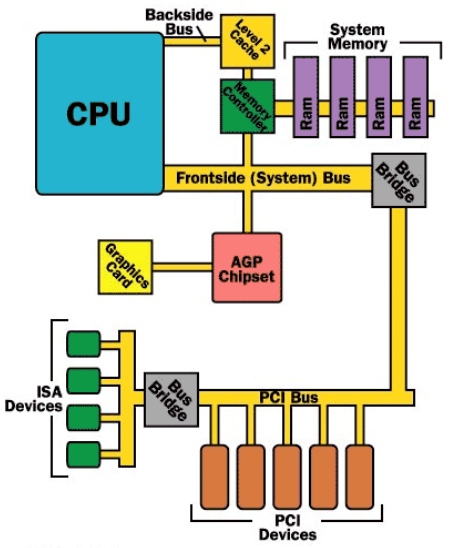


Outline

- Overview of I/O interface
  - Types of devices & communications; Bus vs. ports
- Bus in computing systems
  - Hierarchical buses: system bus vs. I/O bus
- I/O Addressing
  - Standard I/O mapping
  - Memory-mapped I/O
- I/O control: Interrupt and DMA
  - Arbitration: who can use the communication media?
- Error checking and correction

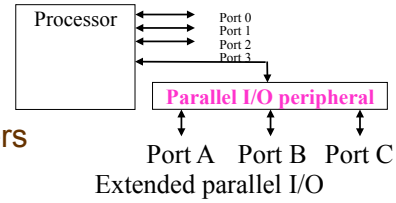
uProcessors Interact with Outside World

- Tasks for embedded systems
  - Sensing & Controlling
  - Interacting w. subsystems
  - Computing: accelerators
  - Communication
- I/O devices
  - File/Storage;
  - Network devices;
  - Scan/print/display devices;
  - Sensors and actuators
- Interface: ports vs. bus

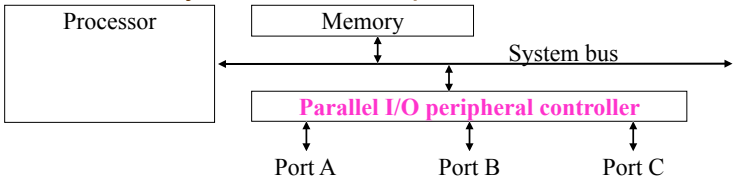


Microprocessor Interfaces

- Port-based I/O (parallel I/O)
  - Processor has one or more dedicated N-bit ports: more pins
  - Read/write ports just like registers
- Bus-based I/O
  - Address, data and control lines → a single bus
  - Communication protocols built into processor
  - Instructions carry out read/write protocols



Extended parallel I/O



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

How to Talk to I/O Peripherals/Addressing

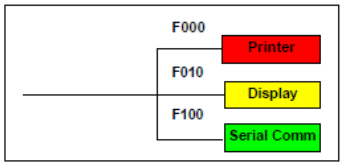
- Standard I/O (I/O-mapped I/O)
  - Specialized instructions: IN/OUT for ports (e.g., Intel x86)
  - Additional pin (M/IO) on bus indicates memory or peripheral access
- Memory-mapped I/O
  - Same instructions: Load/Store
  - Peripheral registers occupy addresses in the same address space as memory
- Most CPUs use memory-mapped I/O
  - I/O instructions do not preclude memory-mapped I/O.

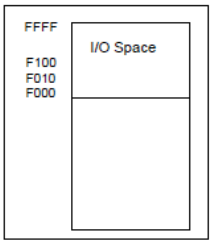
Standard I/O via Bus

- Processor talks to both **memory** and **peripherals** using same bus – two ways to talk to peripherals
- Standard I/O (I/O-mapped I/O)
  - Additional pin (*M/IO*) on bus indicates memory/IO 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
  - Special instructions (e.g., IN, OUT)
  - 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

Memory Mapped I/O

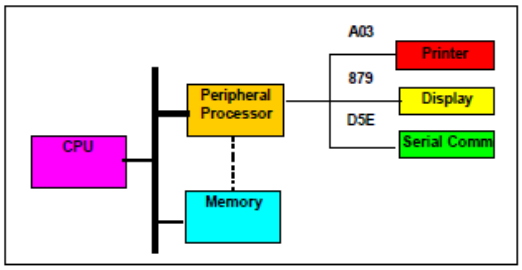
- Map I/O space into an memory address: share the same address space
- I/O reads and writes are done to memory addresses
- Implementation
  - Assign address to each input or output device
- Advantage
  - Easy to implement
  - Low cost
- Disadvantage
  - **Extra burden on CPU**
  - Potentially slow





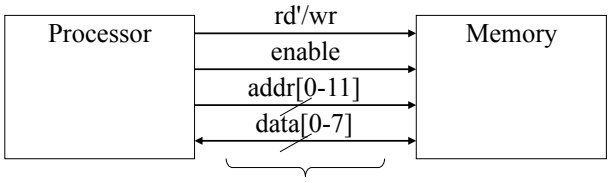
Peripheral Processors

- Dedicates processor to handle all I/O tasks
  - May or may not be connected to memory
- Alternative protocols
  - CPU send / receive from peripheral processor
  - CPU tells peripheral processor where to find / put data in memory
- Advantage
  - I/O speed independent
  - Unburdens CPU
- Disadvantage
  - Higher cost
  - Complex



Bus: Set of Wires with a Single Function

- Address vs. data vs. control buses
  - Wires: Uni-directional or bi-directional
  - Each wire → pin on processor (limited)



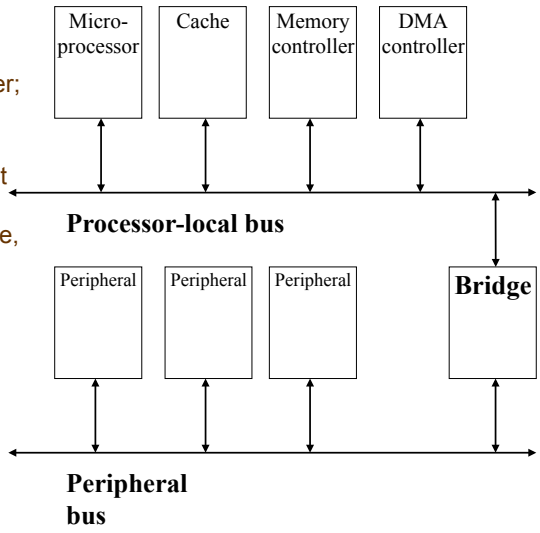
bus structure

Three Basic Functions for Buses

- Control: typical signals may include
  - Read / Write
  - Address / Data present or stable
  - Transmission Direction
  - Ready or Active
- Address
  - Identify devices
  - Serial: address shares path with data
  - Parallel: separate lines
- Data: size of data, serial vs. parallel

Multilevel Bus Architectures

- Don't want one bus for all communication
  - Different speeds: CPU bus faster; peripheral bus slower;
- Processor-local (system) 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: converts communication between different speed buses
  - Single-purpose processor

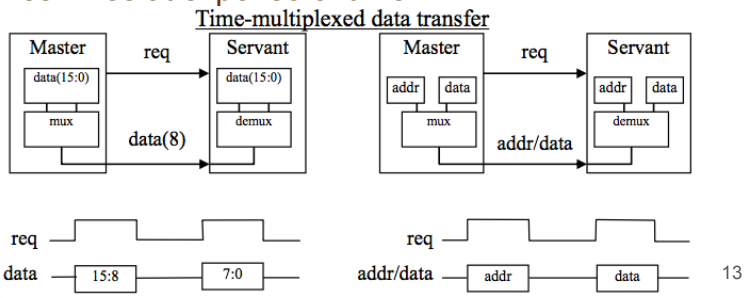


Timing Diagrams on Bus

- 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 sub-protocols
  - Called bus cycle, e.g., read and write
  - Each may be several clock cycles

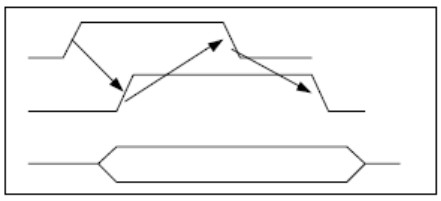
Basic Protocol Concepts

- Actors: master initiates and servant (slave) responds
- Direction: sender vs. receiver
- Addresses: special data on bus
  - Specifies location in memory, a peripheral or its register
- Time multiplexing: shared for multiple addr. or data
  - Saves wires at expense of time

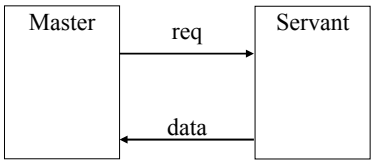


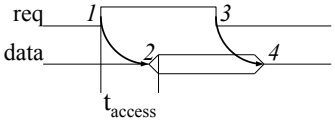
Typical Protocols

- Full handshake
  - Ready for data ?
  - Here's data
  - I've got it
  - OK
- Need 2 control wires and 4 steps



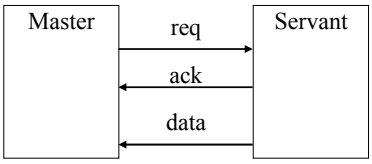
Typical Protocols (cont.)

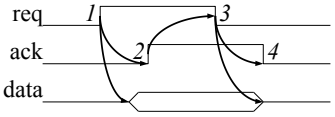




1. Master asserts *req* to receive data
2. Servant puts data on bus **within time t_{access}**
3. Master receives data and deasserts *req*
4. Servant ready for next request

Strobe protocol

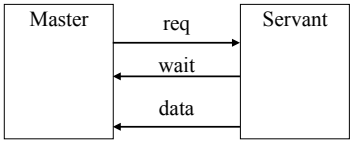


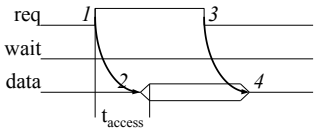


1. Master asserts *req* to receive data
2. Servant puts data on bus **and asserts *ack***
3. Master receives data and deasserts *req*
4. Servant ready for next request

Handshake protocol

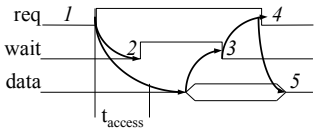
Compromise of Strobe and Handshake





1. Master asserts *req* to receive data
2. Servant puts data on bus **within time t_{access}** (wait line is unused)
3. Master receives data and deasserts *req*
4. Servant ready for next request

Fast-response case



1. Master asserts *req* to receive data
2. Servant can't put data within t_{access}, **asserts *wait*** ack
3. Servant puts data on bus and **deasserts *wait***
4. Master receives data and deasserts *req*
5. Servant ready for next request

Slow-response case

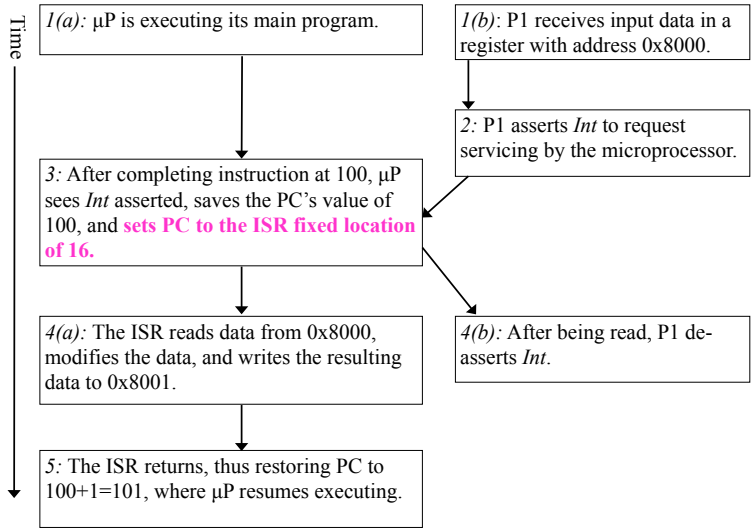
When Data Arrives?

- Peripheral I/O devices intermittently receives data
- Two approaches: CPU polling vs. I/O interrupts
- CPU polling
  - Busy checking and wait: very inefficient
  - Hard to do simultaneous I/O
- Interrupt driven
  - Peripheral interrupt processor when the data is ready
  - Extra interrupt pin Int: if 1, CPU jumps to ISR
  - "polling" of the interrupt pin: built-in hardware, no extra time
- I/O system may have from 1 to many interrupt lines
- Each line may be connected to 1 or several devices

Address for Interrupt Service Routines

- 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

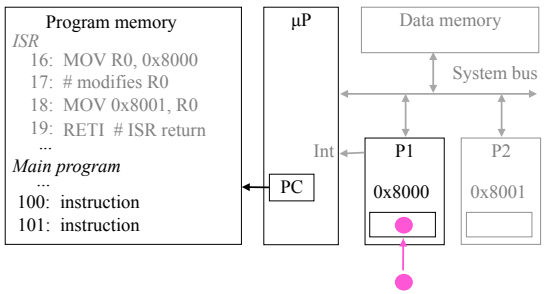
Overview of Fixed ISR Location



Fixed ISR Location: Detailed Steps

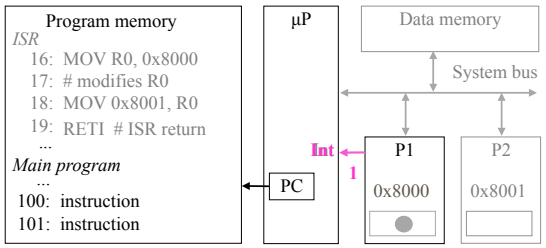
1(a): µP is executing its main program

1(b): P1 receives input data in a register with address 0x8000.



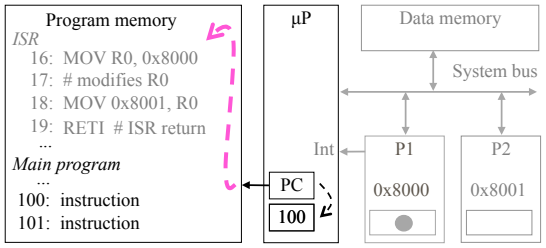
Fixed ISR Location: Detailed Steps (cont.)

2: P1 asserts *Int* to request servicing by the microprocessor



Fixed ISR Location: Detailed Steps (cont.)

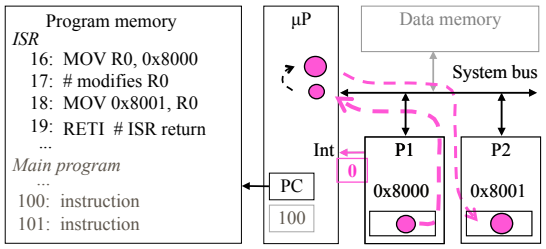
3: After completing instruction at 100, µP sees *Int* asserted, saves the PC's value of 100, and sets PC to the ISR fixed location of 16.



Fixed ISR Location: Detailed Steps (cont.)

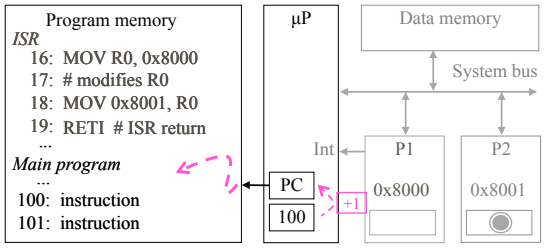
4(a): The ISR reads data from 0x8000, modifies the data, and writes the resulting data to 0x8001.

4(b): After being read, P1 deasserts *Int*.



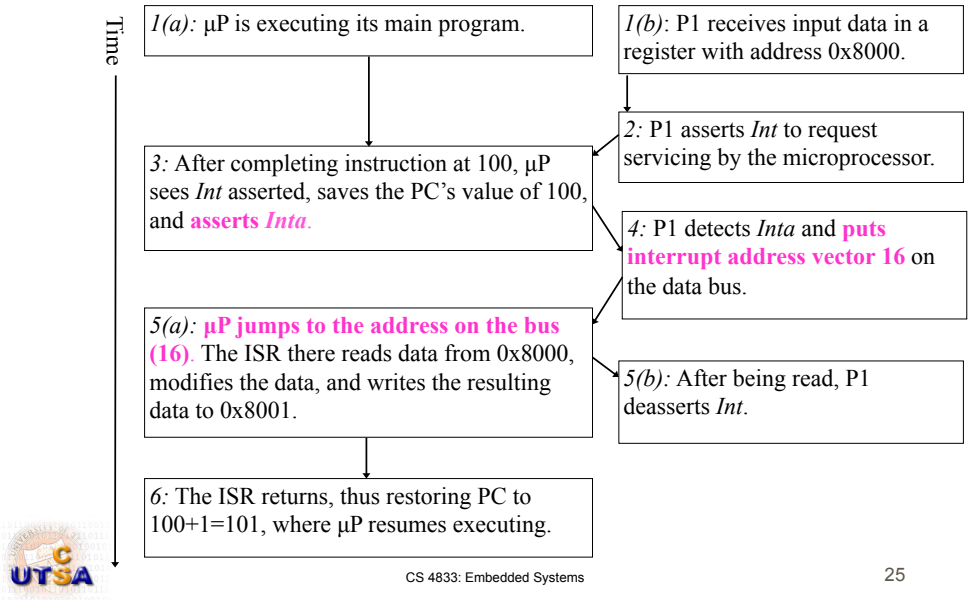
Fixed ISR Location: Detailed Steps (cont.)

5: The ISR returns, thus restoring PC to 100+1=101, where μP resumes executing.



Separate *Int* pins are needed for different peripherals.

Overview of Vectored Interrupt

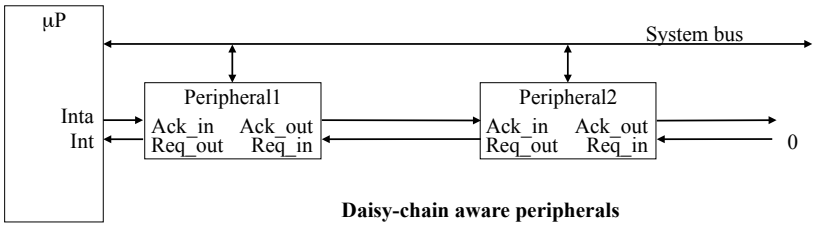


Interrupt Address Table

- Compromise between fixed and vectored interrupts
  - One interrupt pin
  - Table in memory holding ISR addresses (maybe 256 words): for example: 6811 → interrupt vector table
  - 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

Priority Arbitration: Daisy-Chain

- Arbitration done by peripherals
  - Built into peripheral or external logic added
- 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

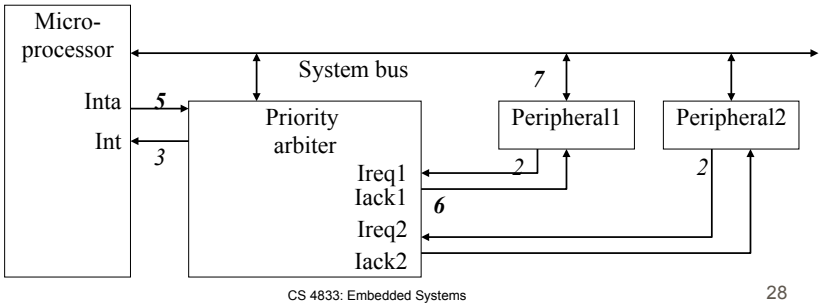


Daisy-chain aware peripherals

What if one upstream (e.g., peripheral1) breaks down?!

Priority Arbitration: Arbiter

- Priority arbiter
  - Single-purpose processor
  - Peripherals request to arbiter, arbiter makes requests to resource
  - Arbiter connected to system bus for configuration only



Arbitration using a Priority Arbiter

1. Microprocessor is executing its program.
2. Peripheral1 asserts *Ireq1*. Peripheral2 asserts *Ireq2*.
3. Priority arbiter sees at least one *Ireq*, so asserts *Int*.
4. Microprocessor stops executing and stores its state.
5. Microprocessor asserts *Inta*.
6. Priority arbiter **asserts *Iack1* to ack. Peripheral1**.
7. Peripheral1 puts its interrupt address on 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: Priority Arbiter

- Fixed priority
  - each peripheral has unique rank
  - highest rank chosen first with simultaneous requests
  - preferred when clear difference in rank between peripherals
- Rotating priority (round-robin)
  - priority changed based on history of servicing
  - better distribution of servicing especially among peripherals with similar priority demands

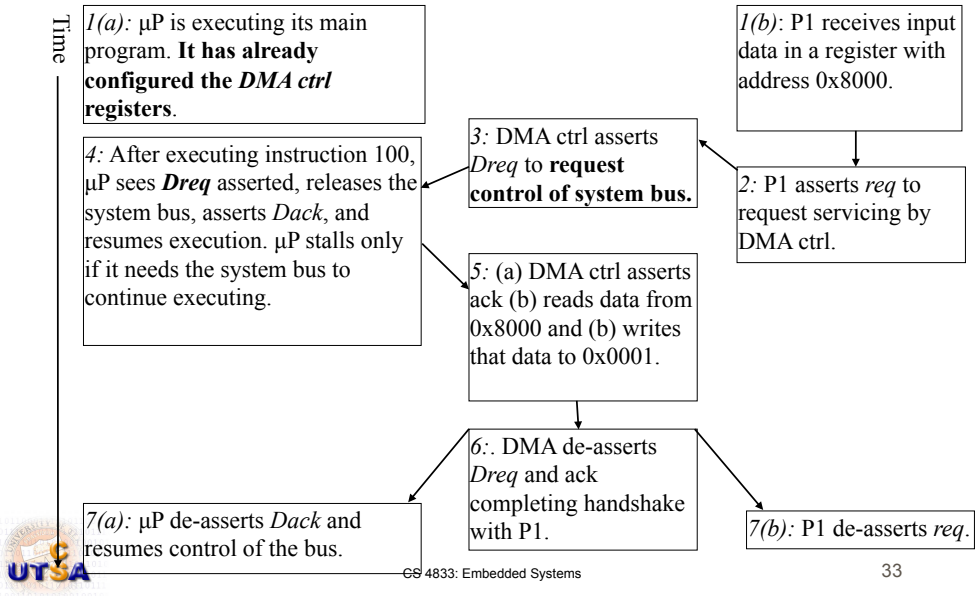
Additional Interrupt Issues

- 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 (DMA)

- 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

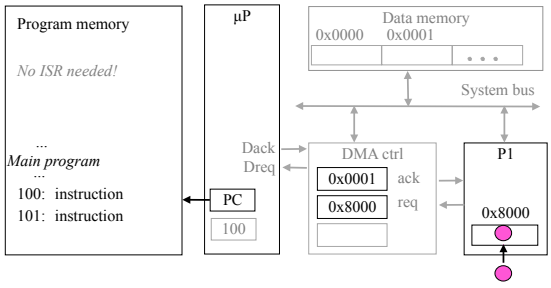
Peripheral to Memory Transfer with DMA



Peri. to Memory Transfer with DMA (cont.)

1(a): µP is executing its main program. It has already configured the DMA ctrl registers

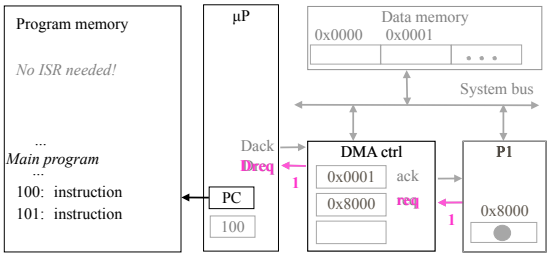
1(b): P1 receives input data in a register with address 0x8000.



Peri. to Memory Transfer with DMA (cont.)

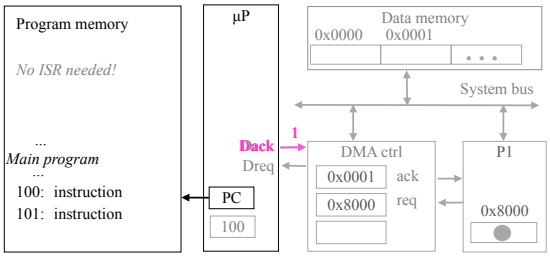
2: P1 asserts *req* to request servicing
by DMA ctrl.

3: DMA ctrl asserts *Dreq* to request control of system bus



Peri. to Memory Transfer with DMA (cont.)

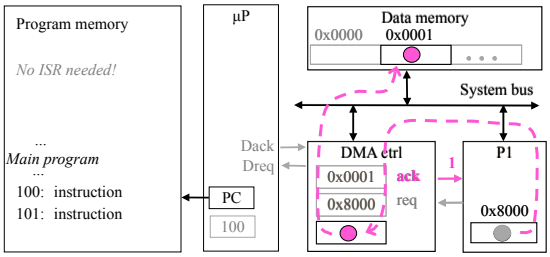
4: After executing instruction 100, μP sees *Dreq* asserted, releases the system bus, asserts *Dack*, and resumes execution, ***μP stalls only if it needs the system bus*** to continue executing.



Peri. to Memory Transfer with DMA (cont.)

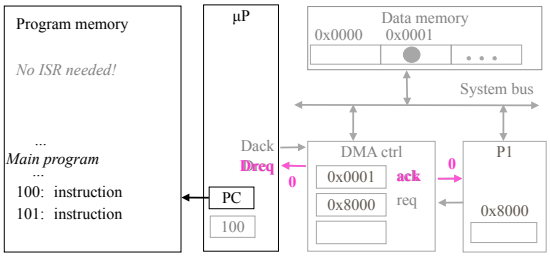
5: DMA ctrl (a) asserts ack, (b) reads data from 0x8000, and (c) writes that data to 0x0001.

(Meanwhile, processor still executing if not stalled!)

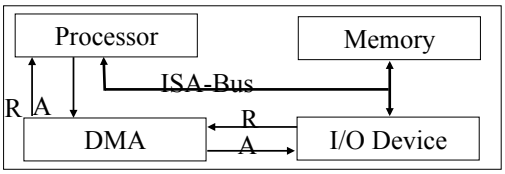


Peri. to Memory Transfer with DMA (cont.)

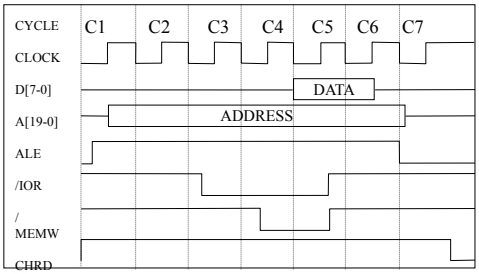
6: DMA de-asserts *Dreq* and *ack* completing the handshake with P1.



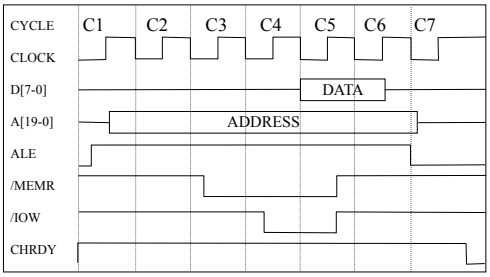
ISA Bus DMA Cycles



DMA Memory-Write Bus Cycle



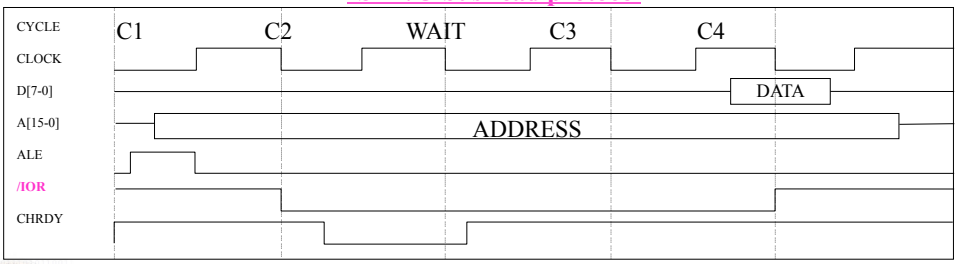
DMA Memory-Read Bus Cycle



ISA Bus for Standard I/O Support

- 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

ISA I/O bus read protocol



Error Detection and Correction

- Error detection: receiver's ability to detect errors during transmission
  - Bit error: single bit is inverted
  - Burst of bit error: consecutive bits received incorrectly
- Error correction: ability of receiver and transmitter to cooperate to correct problem
  - Typically done by acknowledgement/retransmission protocol
- **Parity**: **extra bit** sent with word used for error detection
  - Odd parity: data word plus parity bit contains odd number of 1's
  - Even parity: data word plus parity bit contains even number of 1's
  - Always detects single bit errors, but not all burst bit errors
- **Checksum**: extra word sent with data packet of multiple words
  - e.g., extra word contains XOR sum of all data words in packet

Summary

- Overview of I/O interface
  - Types of devices & communications; Bus vs. ports
- Bus in computing systems
  - Hierarchical buses: system bus vs. I/O bus
- I/O Addressing
  - Standard I/O mapping
  - Memory-mapped I/O
- I/O control: Interrupt and DMA
  - Arbitration: who can use the communication media?
- Error checking and correction