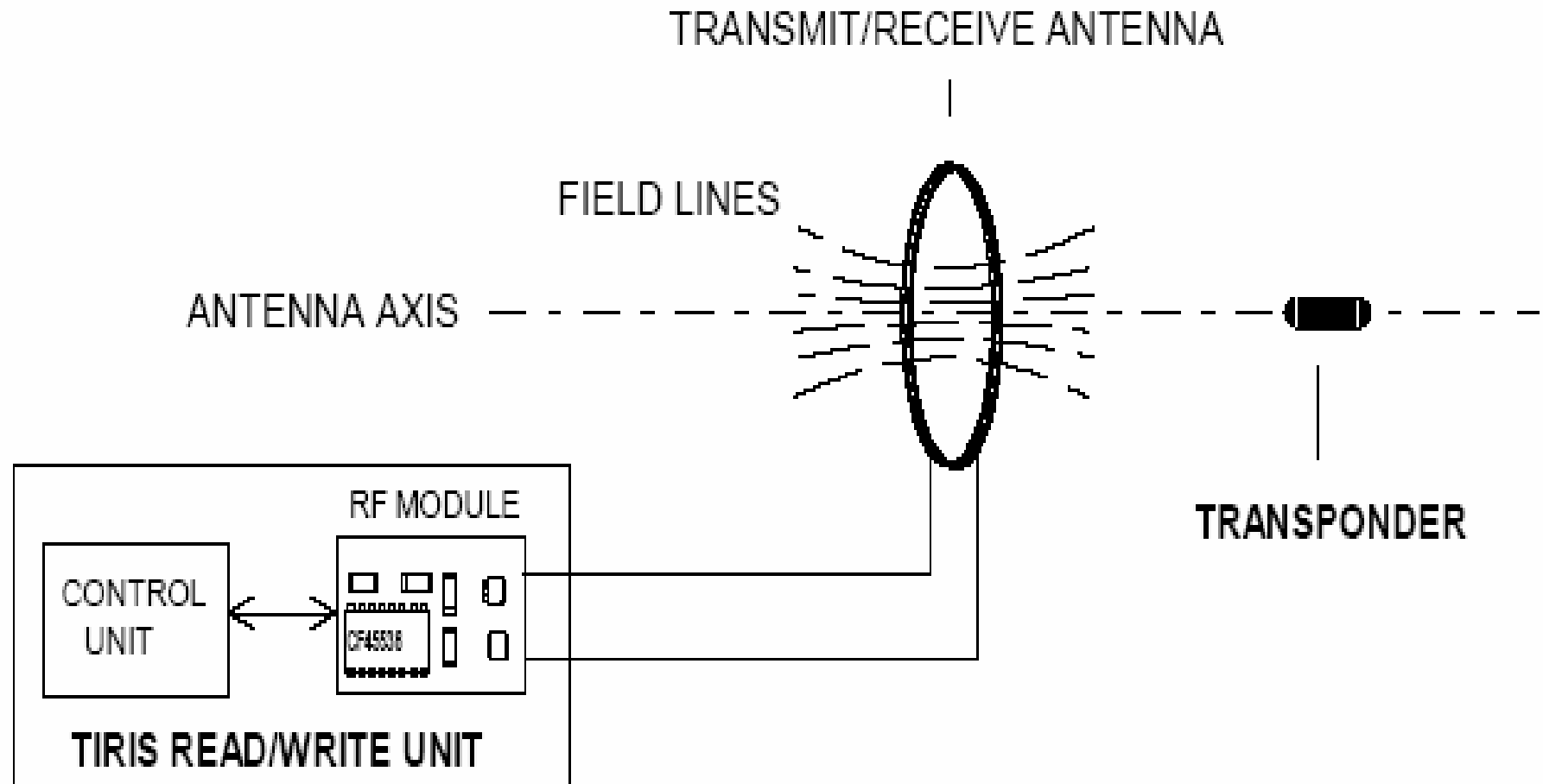
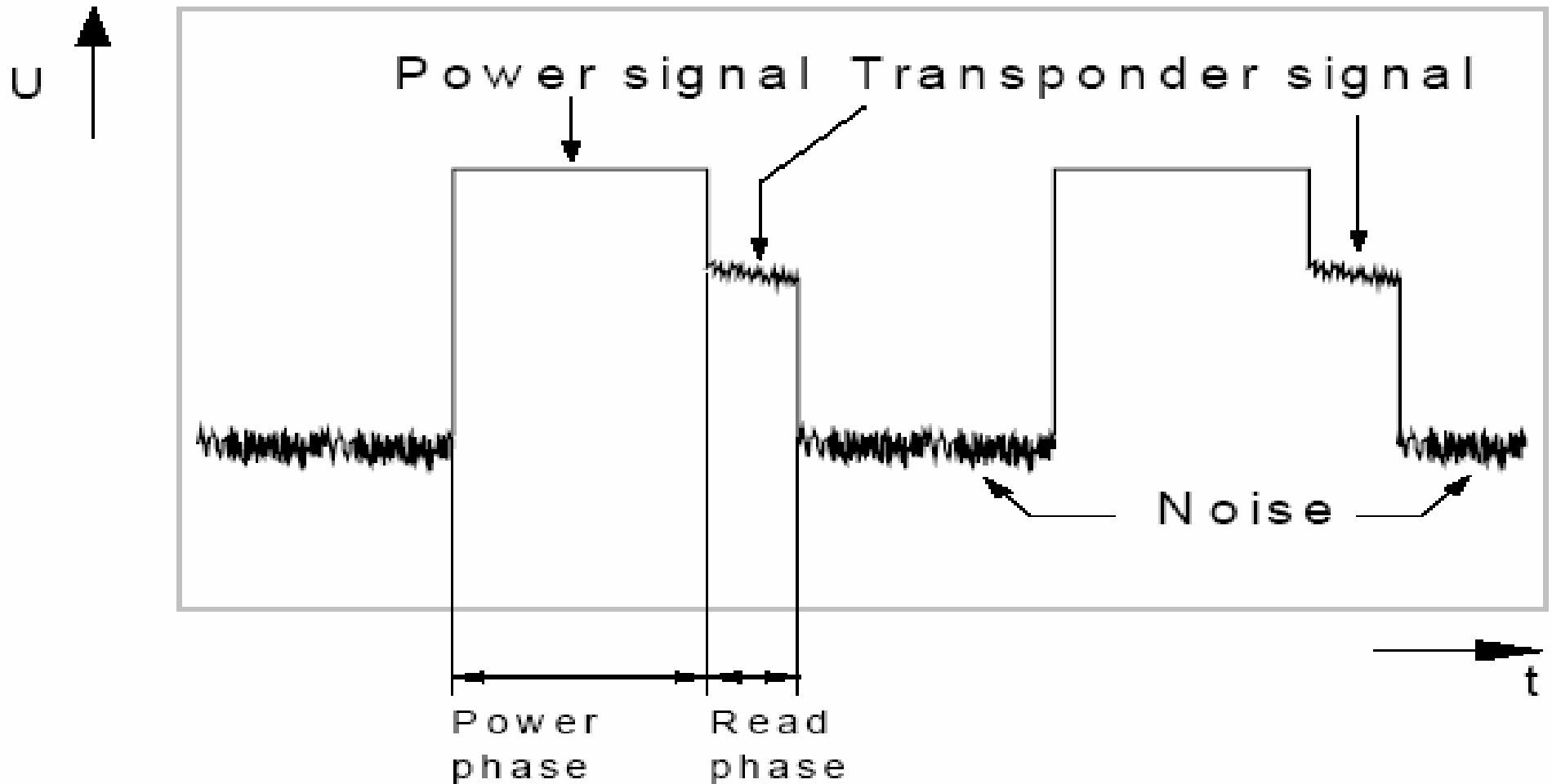


# RFId (Radio Frequency Identification)



# RFID (Radio Frequency Identification)



# RFID (Radio Frequency Identification)

---

TIRIS is Texas Instruments RFID products. This quote is from their “TIRIS for Automatic Recognition of Customers” brochure.

*With TIRIS in place, you can get detailed profiles of customers for a certain geographical area or even as finely defined as a single store. If the same tag is used at other retail outlets, you can collect data of the same customer’s buying patterns of other products. For example, you can determine what airlines your customers fly, what department stores they visit, and so on. This allows you to custom-tailor your programs to market segments—to anticipate customer demands.*

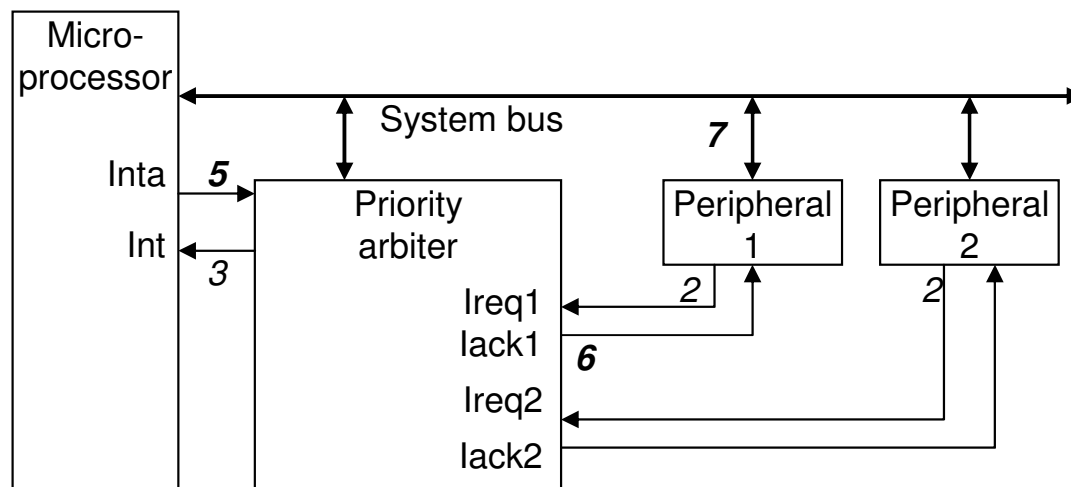
# Quick Review of Interrupts

---

- Three types of ISR addressing:
  - Fixed interrupt
  - Vectored interrupt
  - Compromise: interrupt address table
- Some interrupt options:
  - Maskable or Non-maskable
  - Different methods for how to treat registers while inside the ISR

# Arbitration: Priority arbiter

- Consider the situation where multiple peripherals request service from single resource (e.g., microprocessor, DMA controller) simultaneously - which gets serviced first?
- Priority arbiter
  - Single-purpose processor
  - Peripherals make requests to arbiter, arbiter makes requests to resource
  - Arbiter connected to system bus for configuration only



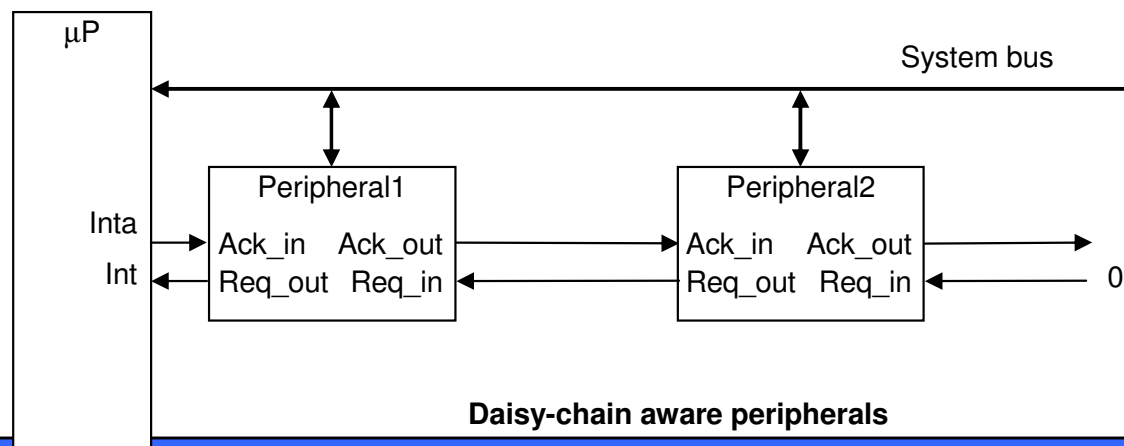
# Arbitration: Priority arbiter

---

- Types of priority
  - 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

# Arbitration: Daisy-chain arbitration

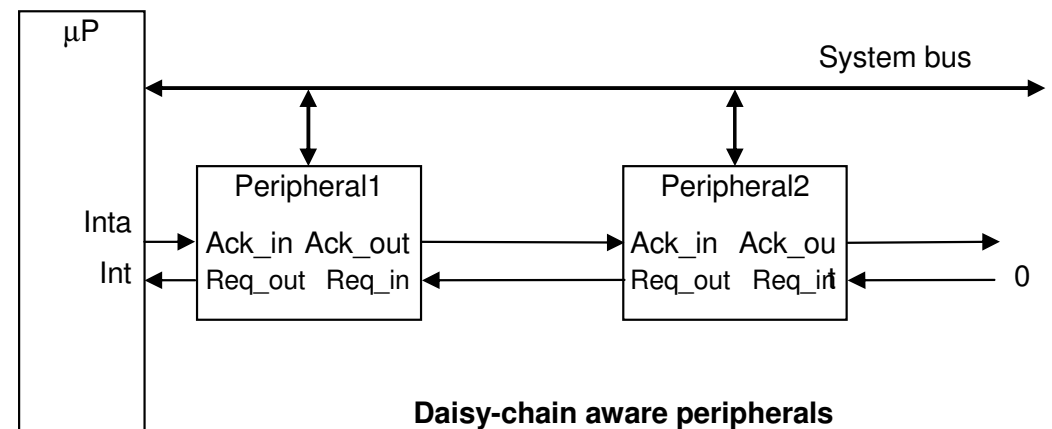
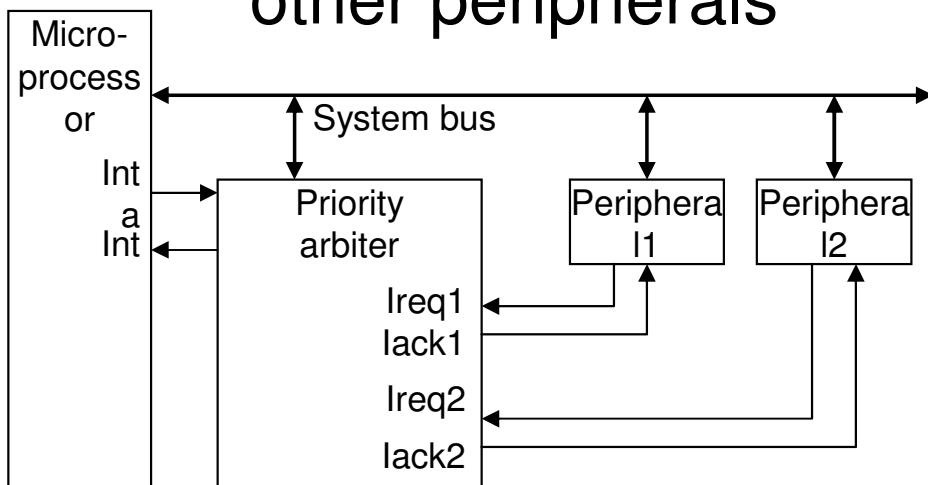
- Arbitration done by peripherals
  - Built into peripheral or external logic added
    - *req* input and *ack* output added to each peripheral
- Peripherals connected to each other in daisy-chain manner
  - One peripheral connected to resource, all others connected “upstream”
  - Peripheral’s *req*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



# Arbitration: Daisy-chain arbitration

- Pros/cons

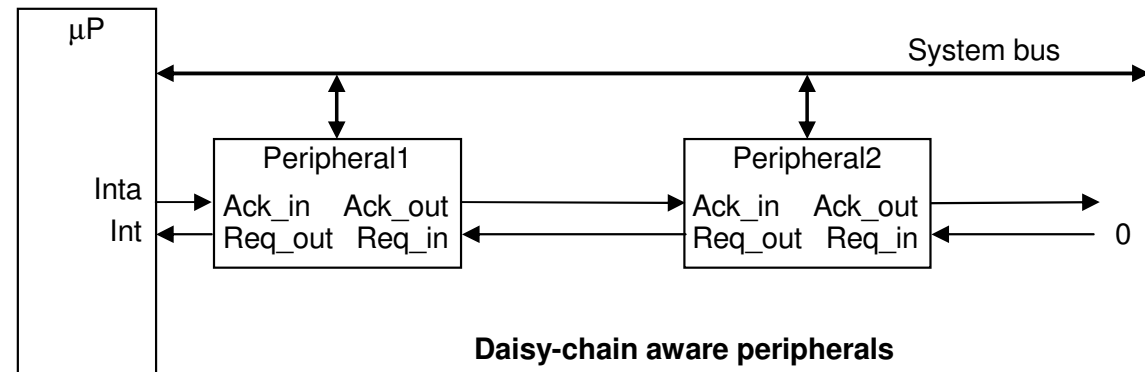
- Easy to add/remove peripheral - no system redesign needed
- Does not support rotating priority
- One broken peripheral can cause loss of access to other peripherals



# ICE! Vectored interrupt

Put the following events during a vectored interrupt in chronological order if peripheral 2 is asserting an interrupt, starting with 1 being the very first thing to happen:

- \_\_\_\_\_ Microprocessor asserts IntA to acknowledge interrupt
- \_\_\_\_\_ Peripheral 2 asserts Req\_out to request servicing
- \_\_\_\_\_ Peripheral 2 puts the address of the ISR on the bus
- \_\_\_\_\_ Peripheral 1 de-asserts Req\_out
- \_\_\_\_\_ Peripheral 1 asserts Req\_out
- \_\_\_\_\_ Peripheral 1 asserts Ack\_out
- \_\_\_\_\_ Microprocessor resumes execution where it was before interruption
- \_\_\_\_\_ Microprocessor jumps to ISR routine and executes code there
- \_\_\_\_\_ Peripheral 2 de-asserts Req\_out

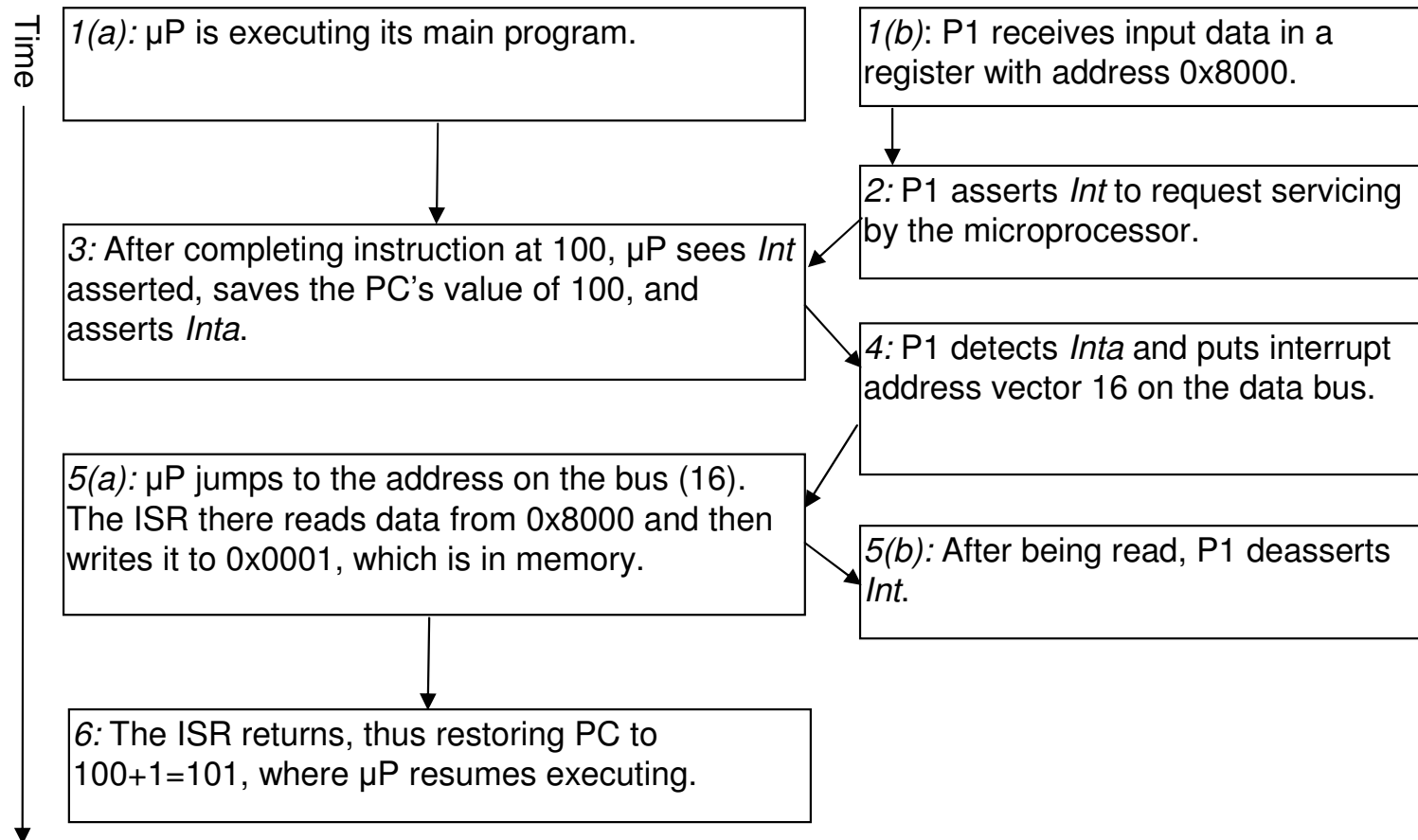


# Direct Memory Access (DMA)

---

- 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
- Buffering
  - Temporarily storing data in memory before processing
  - Data accumulated in peripherals commonly buffered

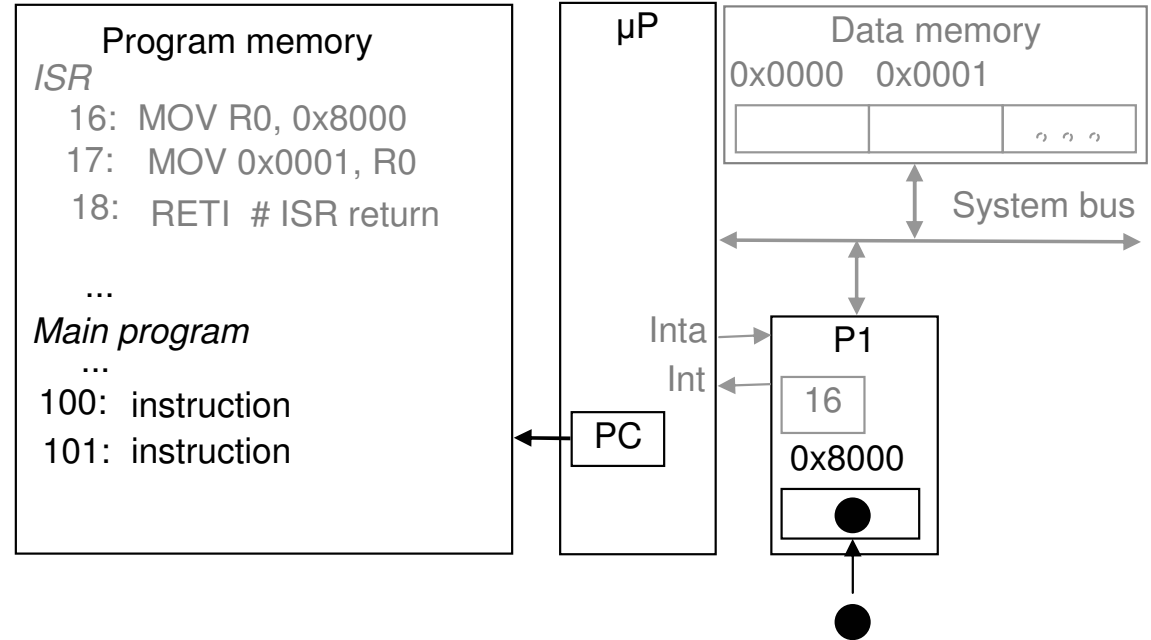
# Peripheral to memory transfer *without* DMA, using vectored interrupt



# Peripheral to memory transfer *without* DMA, using vectored interrupt

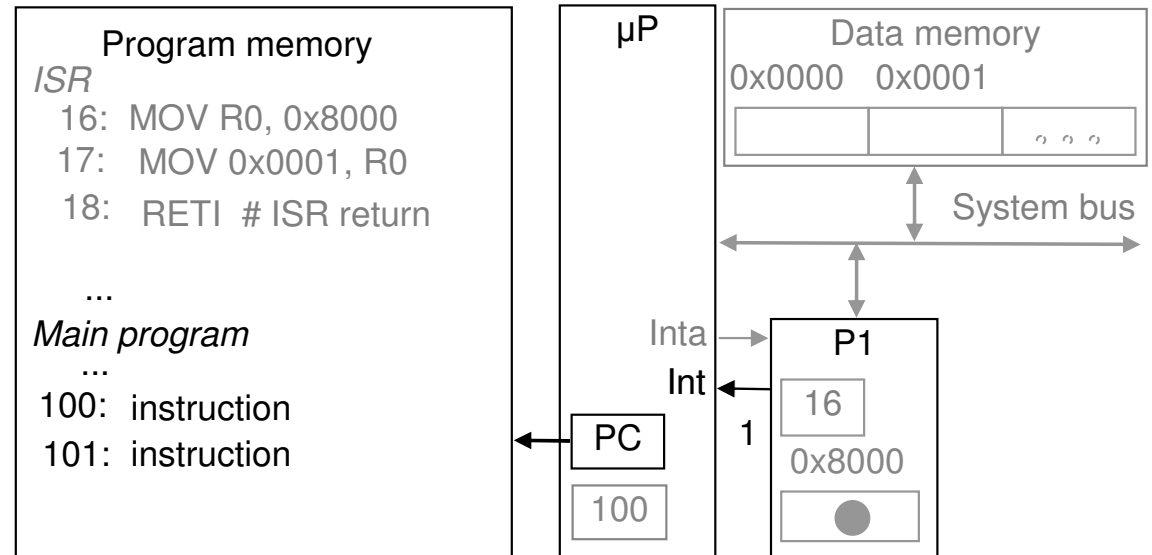
1(a):  $\mu$ P is executing its main program

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



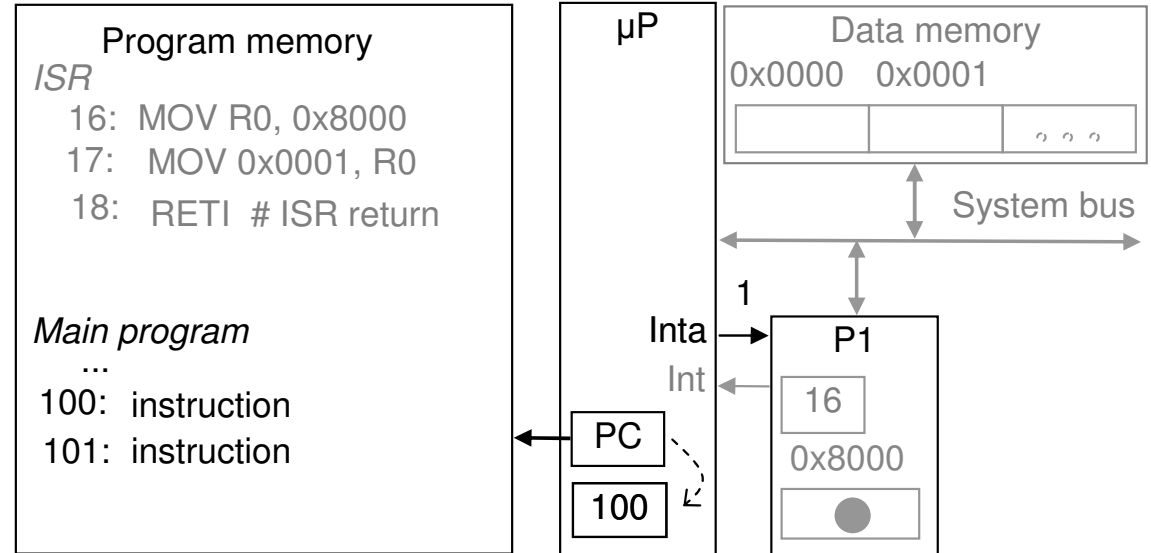
# Peripheral to memory transfer *without* DMA, using vectored interrupt

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



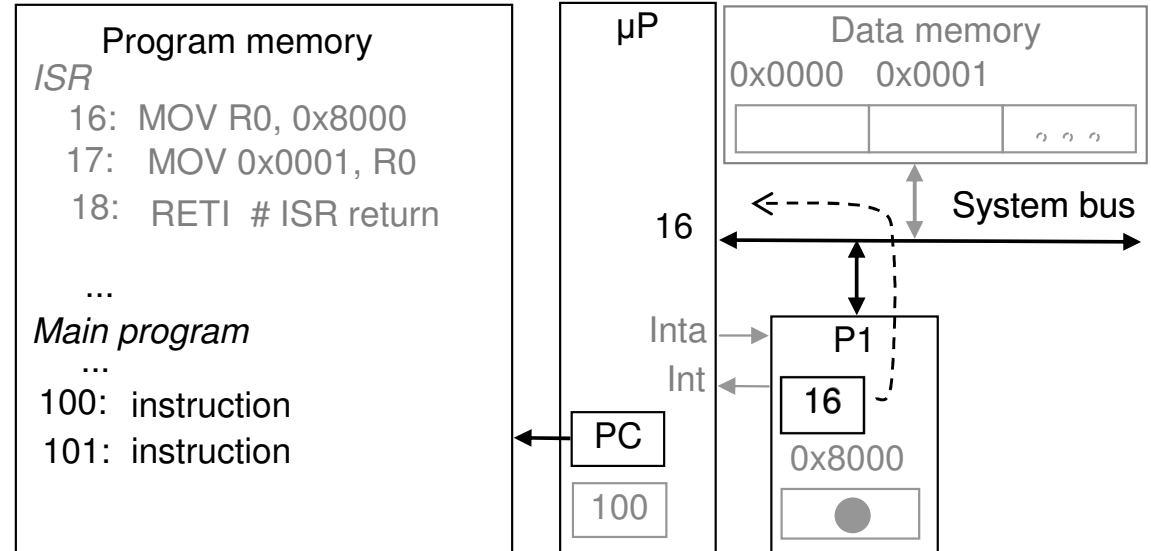
# Peripheral to memory transfer *without* DMA, using vectored interrupt

3: After completing instruction at 100,  $\mu\text{P}$  sees *Int* asserted, saves the PC's value of 100, and asserts *Inta*.



# Peripheral to memory transfer *without* DMA, using vectored interrupt (cont')

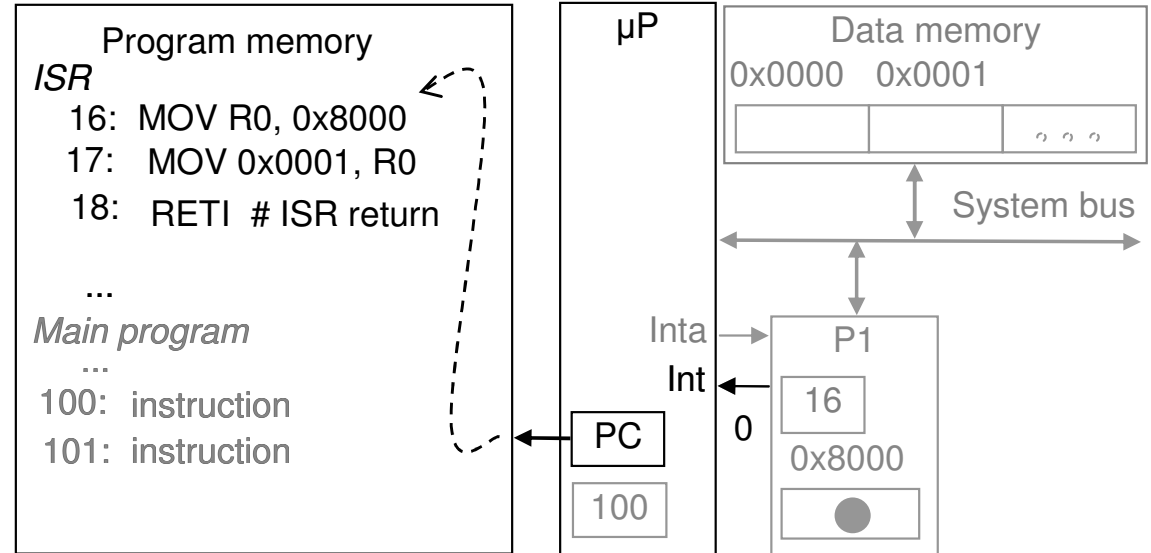
4: P1 detects *Inta* and puts interrupt address vector 16 on the data bus.



# Peripheral to memory transfer *without* DMA, using vectored interrupt (cont')

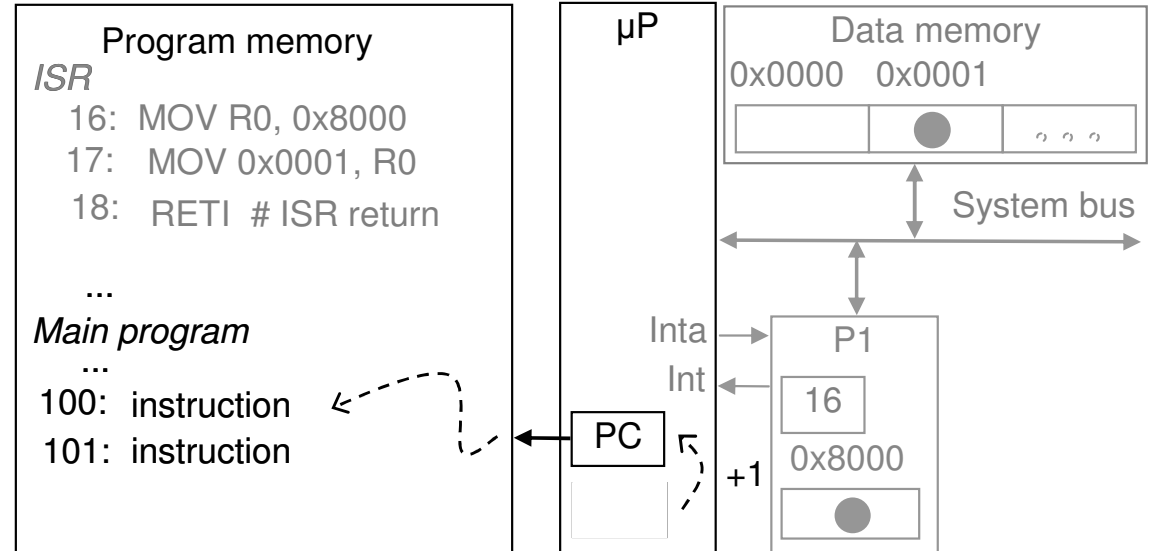
5(a):  $\mu$ P jumps to the address on the bus (16). The ISR there reads data from 0x8000 and then writes it to 0x0001, which is in memory.

5(b): After being read, P1 de-asserts *Int*.

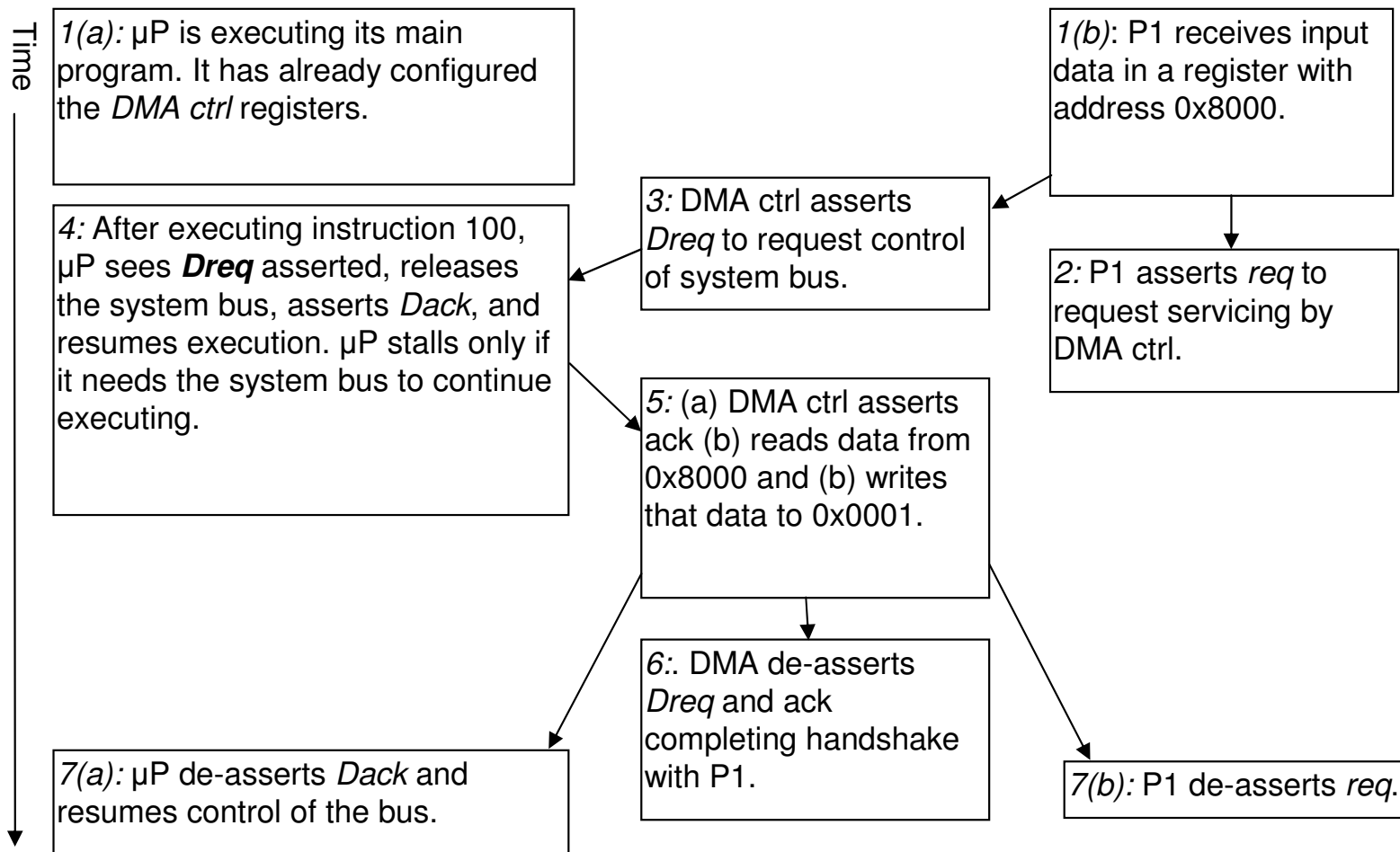


# Peripheral to memory transfer *without* DMA, using vectored interrupt (cont')

6: The ISR returns, thus restoring PC to  $100+1=101$ , where  $\mu\text{P}$  resumes executing.



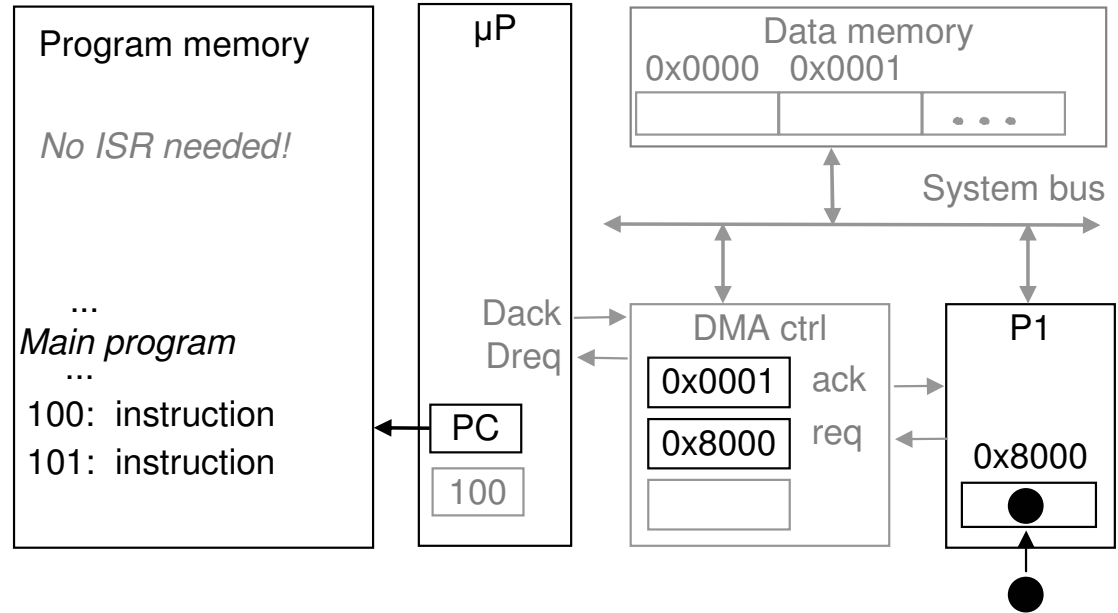
# Peripheral to memory transfer with DMA



# Peripheral to memory transfer with DMA (cont')

1(a):  $\mu\text{P}$  is executing its main program. It has already configured the DMA ctrl registers

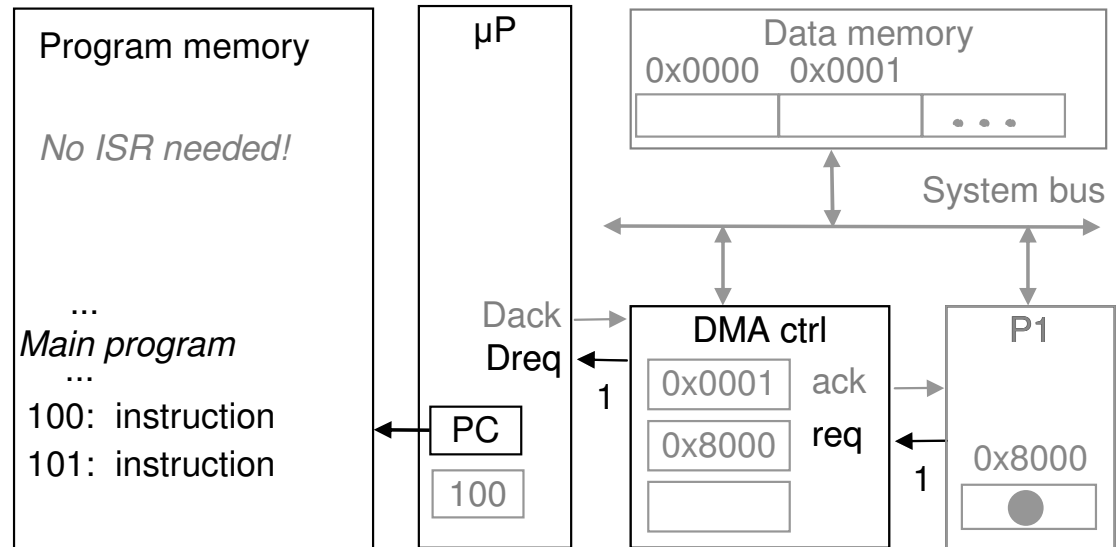
1(b): P1 receives input data in a register with address 0x8000.



# Peripheral to memory transfer with DMA (cont')

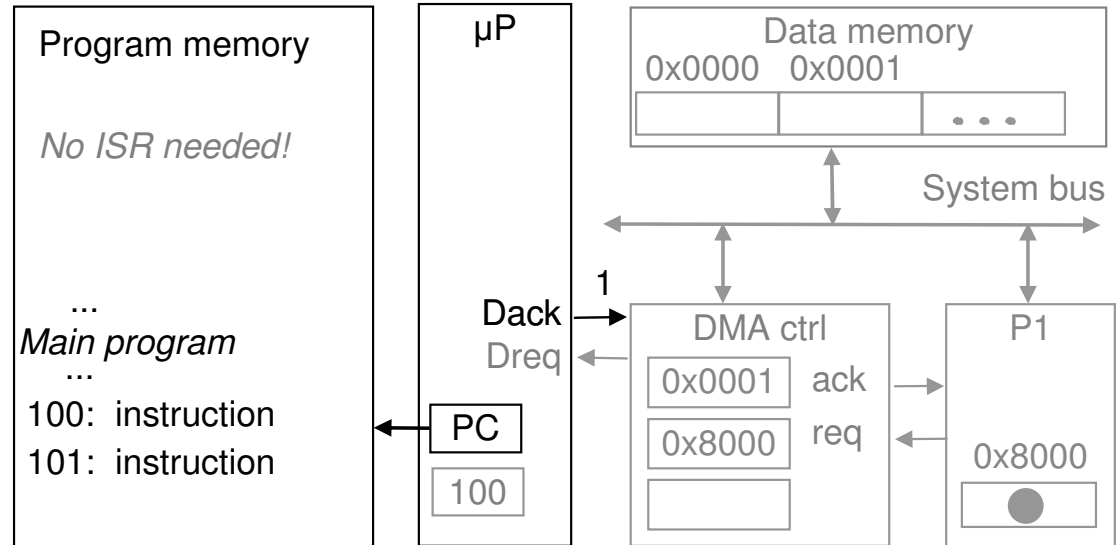
2: P1 asserts *req* to request servicing by DMA ctrl.

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



# Peripheral to memory transfer with DMA (cont')

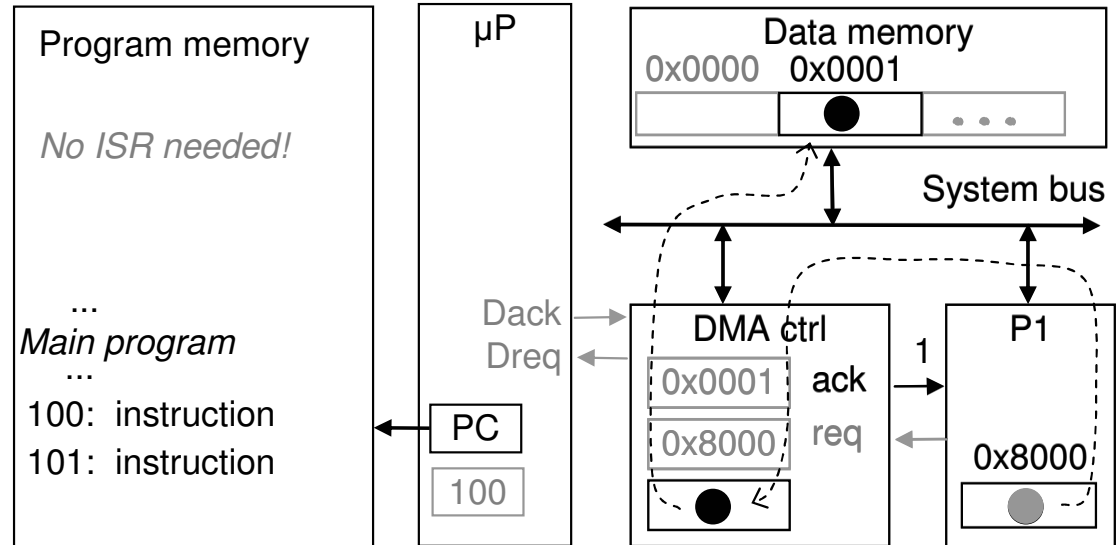
4: After executing instruction 100,  $\mu\text{P}$  sees *Dreq* asserted, releases the system bus, asserts *Dack*, and resumes execution,  $\mu\text{P}$  stalls only if it needs the system bus to continue executing.



# Peripheral to memory transfer with DMA (cont')

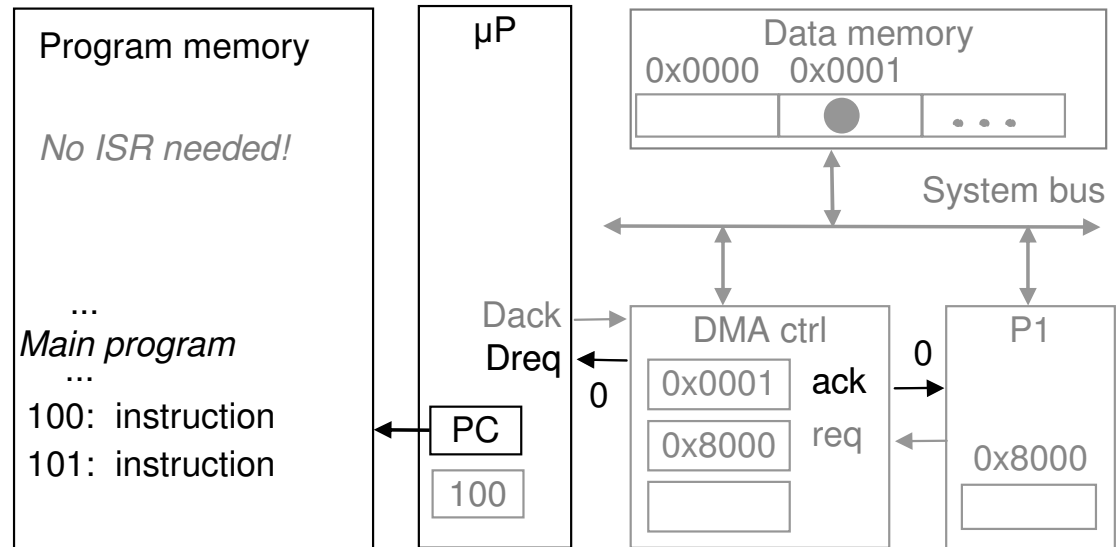
5: DMA ctrl (a) asserts ack, (b) reads data from 0x8000, and (c) writes that data to 0x0001.

(Meanwhile, processor still executing if not stalled!)

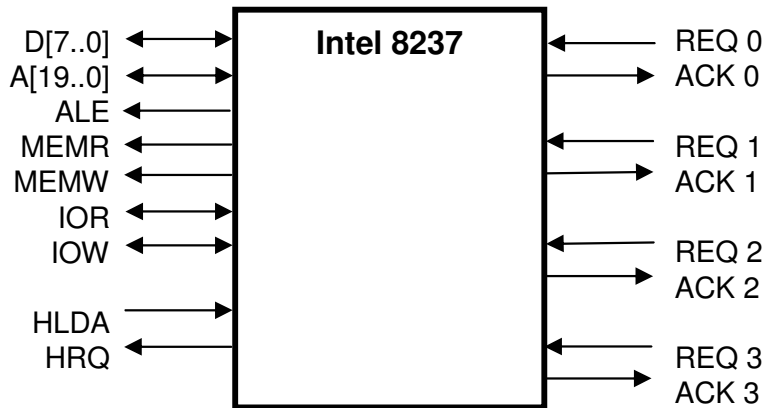


# Peripheral to memory transfer with DMA (cont')

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

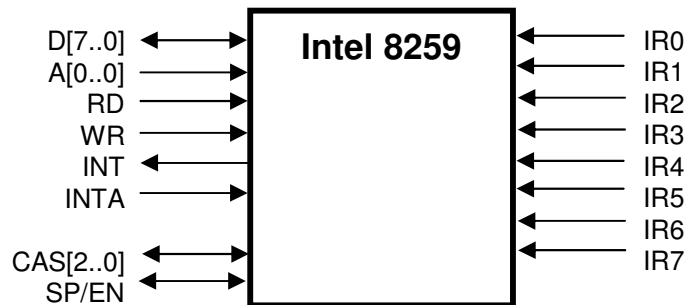


# Intel 8237 DMA controller



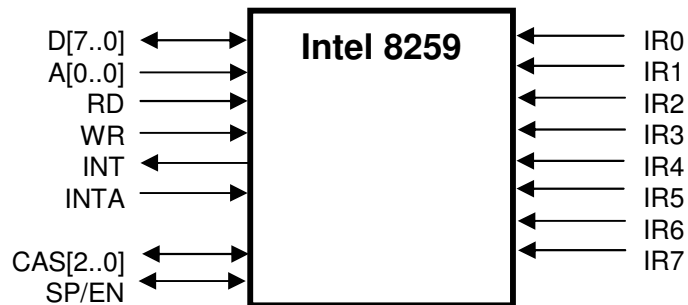
Signal	Description
D[7..0]	These wires are connected to the system bus (ISA) and are used by the microprocessor to write to the internal registers of the 8237.
A[19..0]	These wires are connected to the system bus (ISA) and are used by the DMA to issue the memory location where the transferred data is to be written to. The 8237 is
ALE*	This is the address latch enable signal. The 8237 use this signal when driving the system bus (ISA).
MEMR*	This is the memory write signal issued by the 8237 when driving the system bus (ISA).
MEMW*	This is the memory read signal issued by the 8237 when driving the system bus (ISA).
IOR*	This is the I/O device read signal issued by the 8237 when driving the system bus (ISA) in order to read a byte from an I/O device
IOW*	This is the I/O device write signal issued by the 8237 when driving the system bus (ISA) in order to write a byte to an I/O device.
HLDA	This signal (hold acknowledge) is asserted by the microprocessor to signal that it has relinquished the system bus (ISA).
HRQ	This signal (hold request) is asserted by the 8237 to signal to the microprocessor a request to relinquish the system bus (ISA).
REQ 0,1,2,3	An attached device to one of these channels asserts this signal to request a DMA transfer.
ACK 0,1,2,3	The 8237 asserts this signal to grant a DMA transfer to an attached device to one of these channels.
*See the ISA bus description in this chapter for complete details.	

# Intel 8259 programmable priority controller



Signal	Description
D[7..0]	These wires are connected to the system bus and are used by the microprocessor to write or read the internal registers of the 8259.
A[0..0]	This pin acts in conjunction with WR/RD signals. It is used by the 8259 to decipher various command words the microprocessor writes and status the microprocessor wishes to read.
WR	When this write signal is asserted, the 8259 accepts the command on the data line, i.e., the microprocessor writes to the 8259 by placing a command on the data lines and asserting this signal.
RD	When this read signal is asserted, the 8259 provides on the data lines its status, i.e., the microprocessor reads the status of the 8259 by asserting this signal and reading the data lines.
INT	This signal is asserted whenever a valid interrupt request is received by the 8259, i.e., it is used to interrupt the microprocessor.
INTA	This signal, is used to enable 8259 interrupt-vector data onto the data bus by a sequence of interrupt acknowledge pulses issued by the microprocessor.
IR 0,1,2,3,4,5,6,7	An interrupt request is executed by a peripheral device when one of these signals is asserted.
CAS[2..0]	These are cascade signals to enable multiple 8259 chips to be chained together.
SP/EN	This function is used in conjunction with the CAS signals for cascading purposes.

# Why No Interrupt Acknowledges to Downstream Peripherals?



It gets programmed with the look-up table. So, once the interrupt is acknowledged, the 8259 writes the ISR address to the bus, and by running the ISR, the peripheral will know it has been serviced.

# Things to do before next Tuesday

---

- Eat Turkey (or Tofurkey for vegetarians)