Interfacing

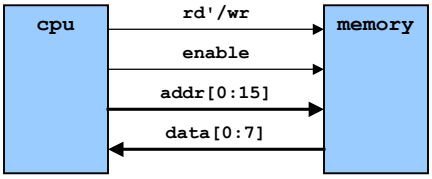
Introduction
Addressing
Interrupt
DMA
Arbitration
Advanced communication architectures

Introduction

- Embedded system functionality aspects
  - Processing
    - Transformation of data
    - Implemented using processors
  - Storage
    - Retention of data
    - Implemented using memory
  - Communication
    - Transfer of data between processors and memories
    - Implemented using buses
    - Called interfacing

A simple bus

- Wires:
  - Uni-directional or bi-directional
    - One line may represent multiple wires
- Bus
  - Sets of wires with a single function
    - Address bus, data bus, control bus
  - Single set of wires collecting all functions



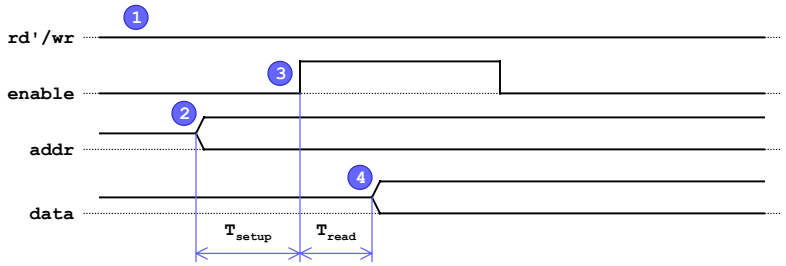
Timing Diagrams

- Common method for describing a protocol
  - Time proceeds to the right on the x axis
- Control signal:
  - Active high: wr
    - Asserted when 1
  - Active low: wr
    - Asserted when 0
- Data signal:
  - Valid:
    - Data on the bus can be read/written
  - Not valid
    - Data on the bus is meaningless

Timing Diagrams

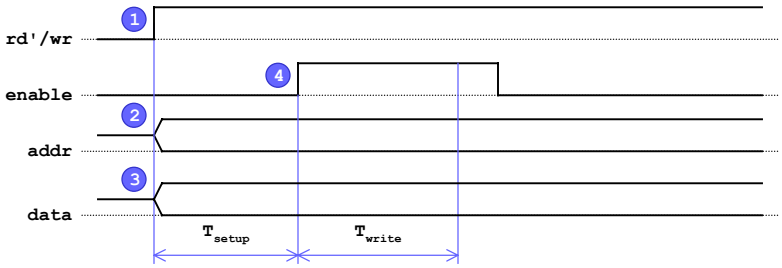
- Protocol:
  - Defines how communication takes place
  - May have subprotocols
    - Bus cycles
    - Each bus cycle may be several clock cycles

Timing Diagrams: Read example



1. **rd'** asserted
   starts read bus cycle
2. address placed on **addr**
   address must be stable at least T_{setup} before **enable** asserted
3. **enable** asserted
   Triggers memory to place data on **data** wires
   Not later than after the time delay T_{read}
4. data ready on **data** wires

Timing Diagrams: Write example

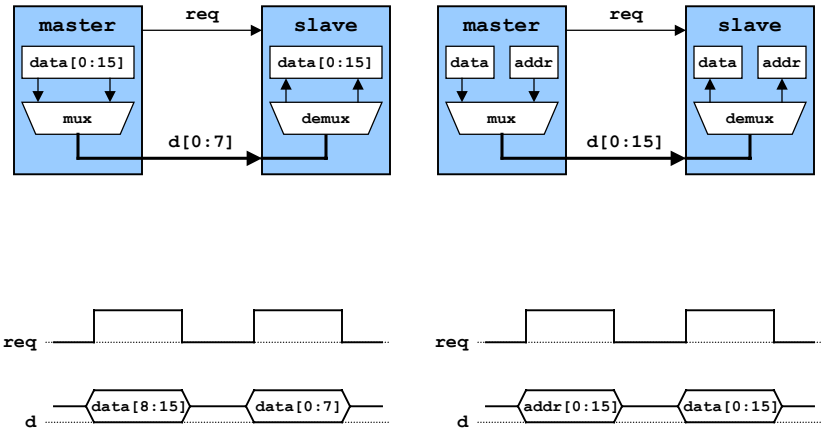


1. **wr** asserted
   starts write bus cycle
2. address placed on **addr**
   address must be stable at least T_{setup} before **enable** asserted
3. data placed on **data** wires
   data must be stable at least T_{setup} before **enable** asserted
4. **enable** asserted
   triggers memory save data found on **data** wires
   must remain asserted for at least T_{write}

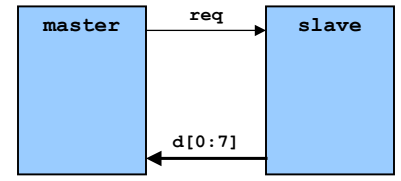
Basic protocol concepts

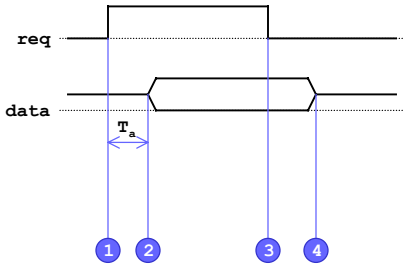
- Actor
  - Master: initiates the communication
  - Slave: responds to master
- Direction
  - Sender: Data source
  - Receiver: Data destination
- Time multiplexing
  - Share a single set of wires for multiple pieces of data
    - Subsequent bytes of a word
    - Data/Address
  - Saves wires at expense of time

Time multiplexing



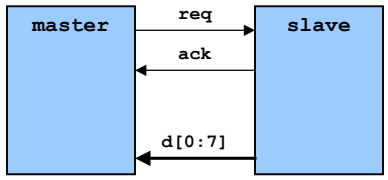
Strobe protocol

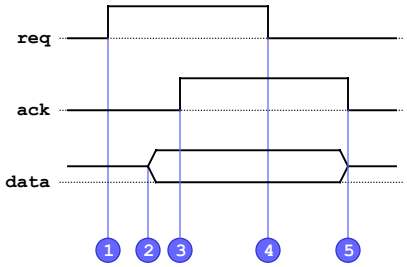




1. Master asserts `req` to receive data
2. Slave puts data on bus Within T_a
2 3. Master receives data
3. Master deasserts `req`
4. Slave ready for next request

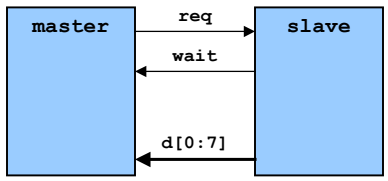
Handshake protocol

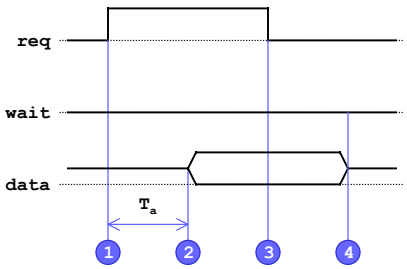




1. Master asserts **req** to receive data
2. Slave puts data on bus
3. Slave asserts **ack**
3 4. Master receives data
4. Master deasserts **req**
5. Slave deasserts **ack**
   Slave ready for next request

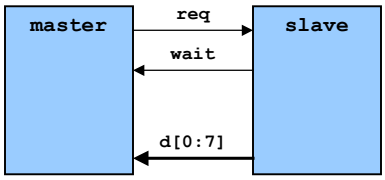
Strobe/handshake protocol: Fast

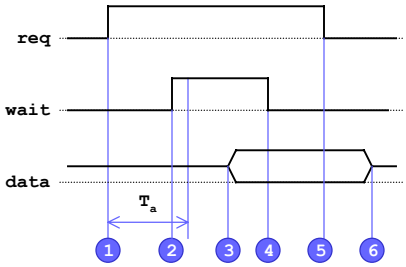




1. Master asserts **req** to receive data
2. Slave puts data on bus
   Within T_a
   The **wait** line is unused
2 3. Master receives data
3. Master deasserts **req**
4. Slave ready for next request

Strobe/handshake protocol: Slow

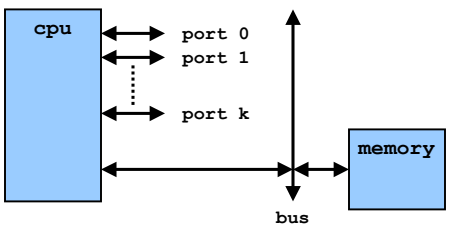




1. Master asserts **req** to receive data
2. Slave can't put data on bus within T_a
   Slave asserts **wait**
3. Slave puts data on bus
4. Slave deasserts **wait**
4 5. Master receives data
5. Master deasserts **req**
6. Slave ready for next request

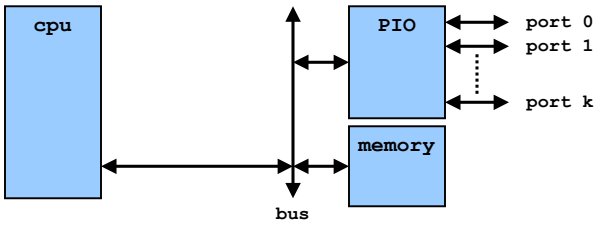
I/O Addressing

- CPU communicates using some of its pins
- Port-based I/O or parallel I/O
  - CPU has one or more n-bit ports
  - Software reads and writes a port just like a register
- Bus-based I/O
  - CPU has address, data and control ports



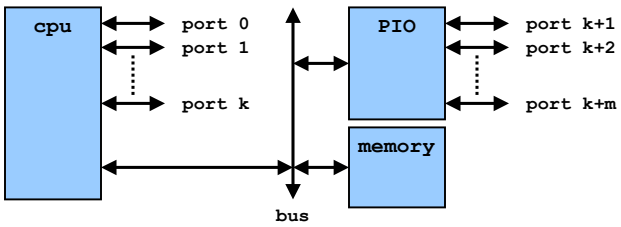
I/O Addressing

- Parallel I/O peripheral (PIO)
  - Processor only supports bus-based I/O
    - … But parallel I/O needed
  - Ports are managed by a dedicated peripheral



I/O Addressing

- Extended parallel I/O
  - Processor supports port-based I/O
    - … But more ports needed
  - One or more PIO extending total number of ports



Memory mapped and Standard I/O

- Processor uses the same bus to talk to
  - Memory
  - Peripherals
- There are two main ways to communicate
  - Memory-mapped I/O
  - Standard I/O

Memory mapped I/O

- Peripheral registers occupy addresses in same address space as memory
  - E.g. Bus has 16-bit address
    - Lower 32K map to memory
    - Upper 32k map to peripherals
- Requires no special instructions
  - Assembly instructions involving memory work with peripherals as well
    - E.g. **`MOV`**, **`LOAD`**, **`STORE`**, **`ADD`**, …
  - Standard I/O requires special instructions to move data between peripheral registers and memory
    - E.g. **`IN`**, **`OUT`**

Standard I/O

- No loss of memory addresses to peripherals
  - E.g. Bus has 16-bit address
    - All 64K map to memory when **M/IO** set to 0
    - all 64K map to peripherals when **M/IO** set to 1
- Simpler address decoding logic in peripherals
  - When the number of peripherals much smaller than address space, high-order address bits are ignored
    - Smaller and/or faster comparators
  - Additional pin on bus indicates whether a memory or peripheral access
    - **M/IO**

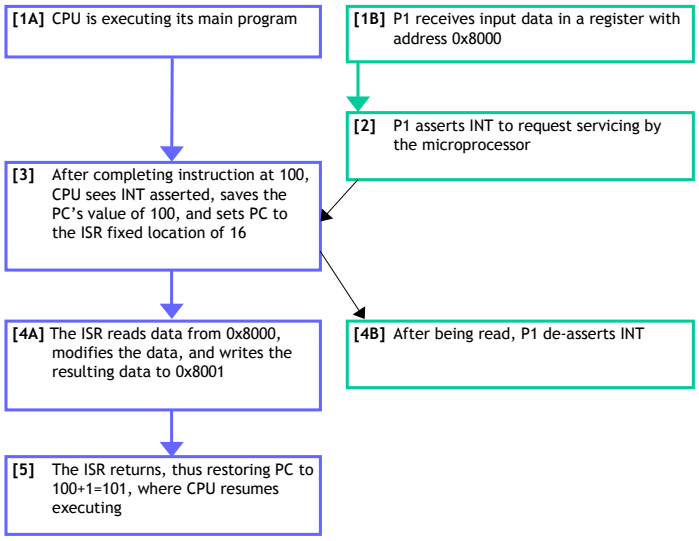
Interrupts

- Suppose a peripheral intermittently receives data, which must be serviced by the processor
  - Processor can poll the peripheral regularly
    - To see if data has arrived
    - Wastes a lot of time
  - Peripheral can interrupt the processor when ready
- Requires one or more extra pin or pins
  - Interrupt pins: INT0, INT1, …
- If INT is 1
  - Processor suspends current program
  - Jumps to an Interrupt Service Routine, or ISR
  - Accomplishes the requested I/O operation

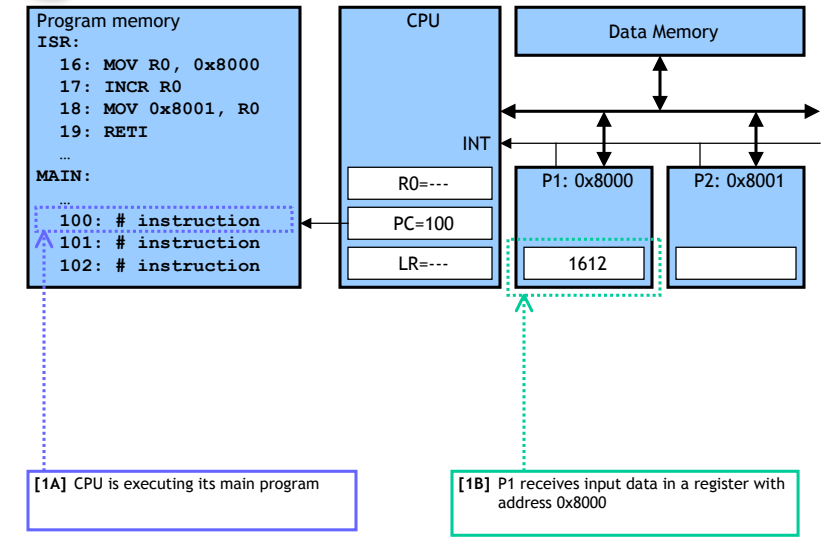
Interrupts

- Different interrupt signals (INT0, INT1, …) have different service routines (ISR0, ISR1, …)
- What is the address of the correct ISR?
  - Fixed interrupt
    - Address built into microprocessor, cannot be changed
    - Either ISR or jump to actual ISR stored at address
  - Vectored interrupt
    - Peripheral must provide the address
    - Common when microprocessor has multiple peripherals connected by a system bus
  - Compromise: interrupt address table

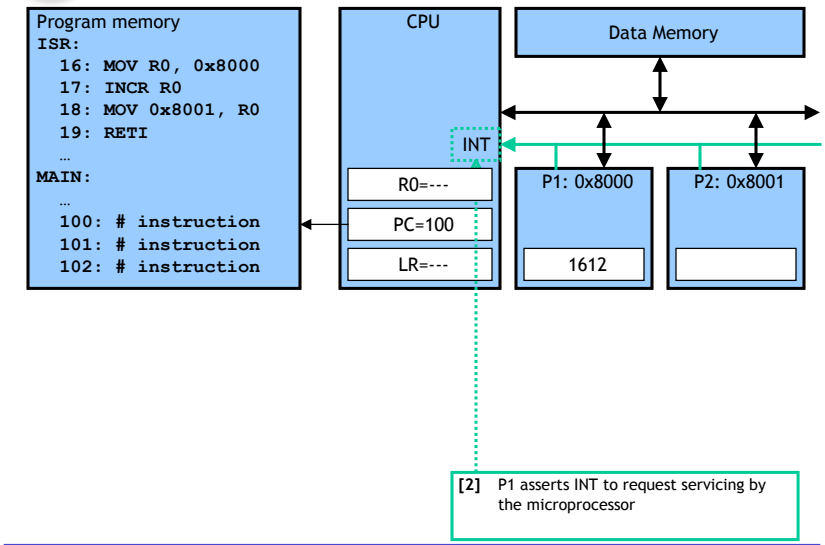
Fixed ISR location



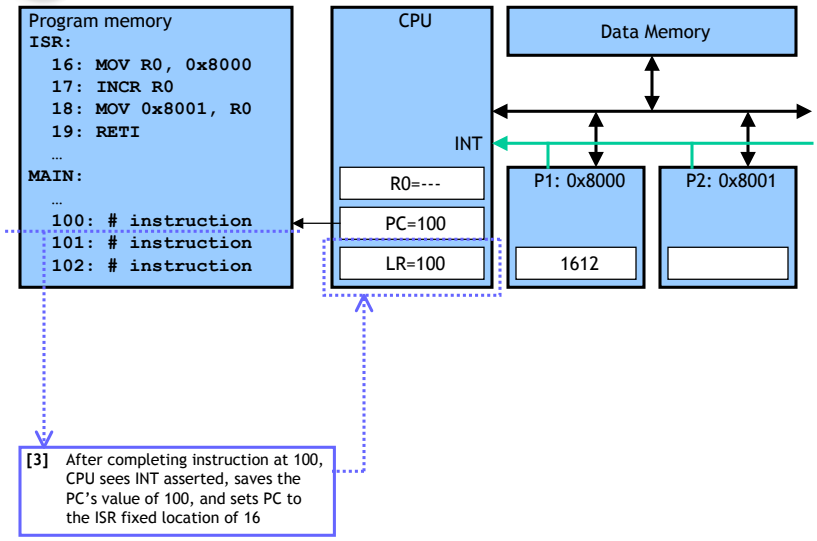
Fixed ISR location



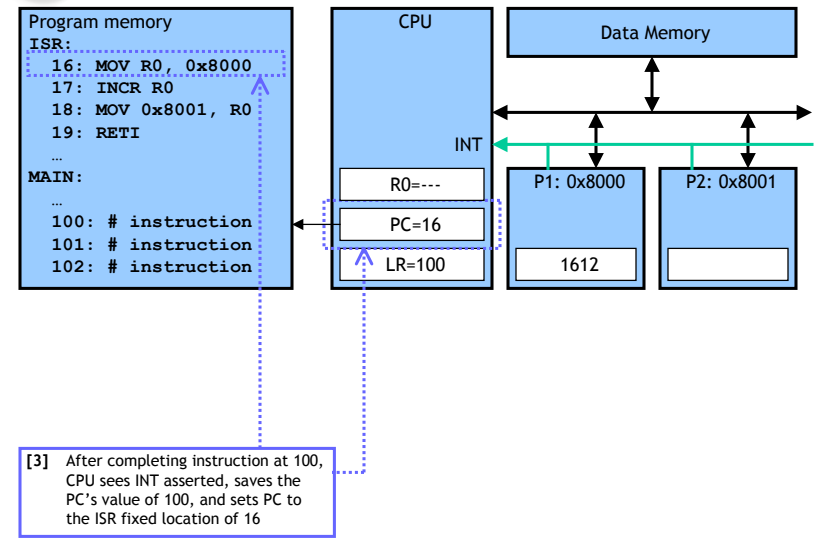
Fixed ISR location



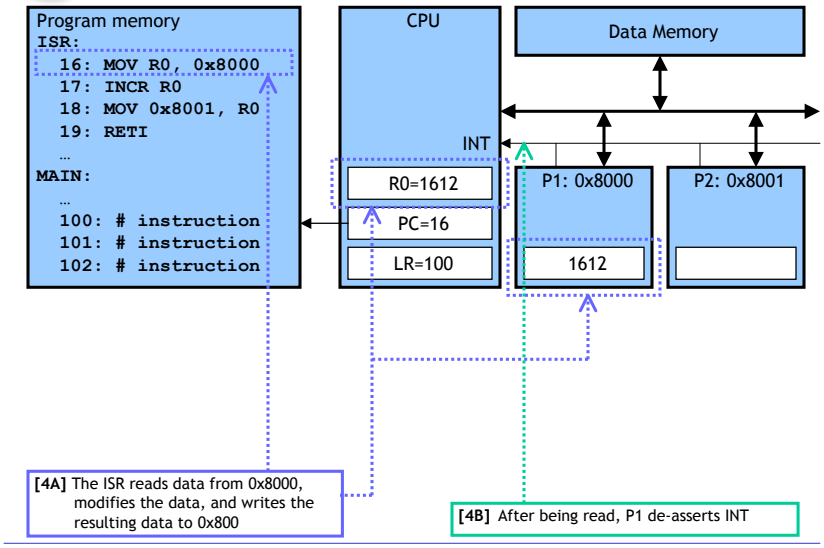
Fixed ISR location



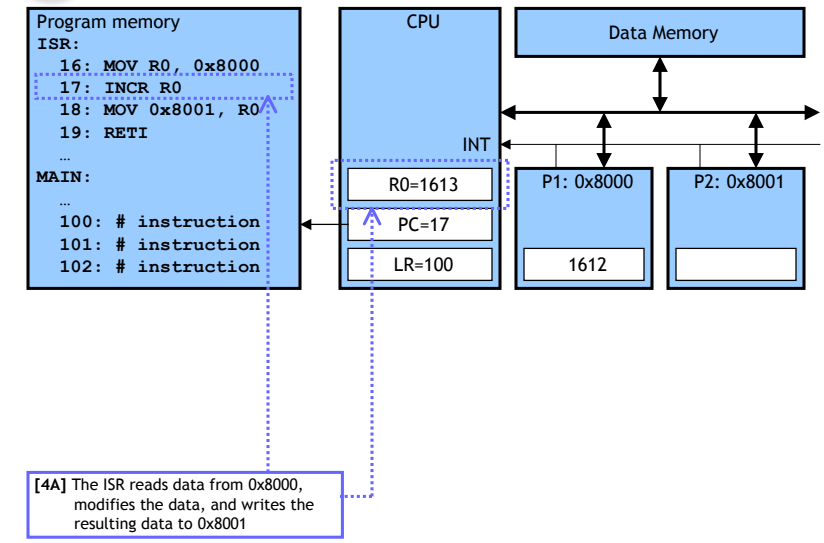
Fixed ISR location



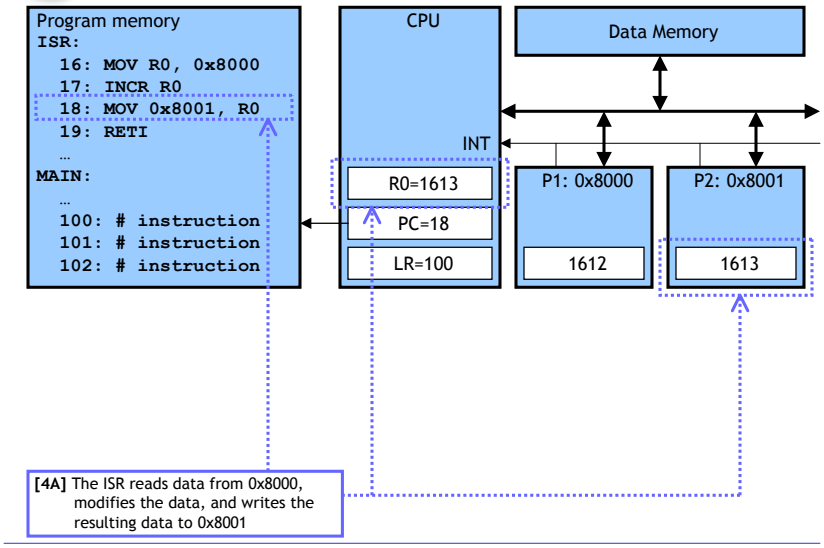
Fixed ISR location



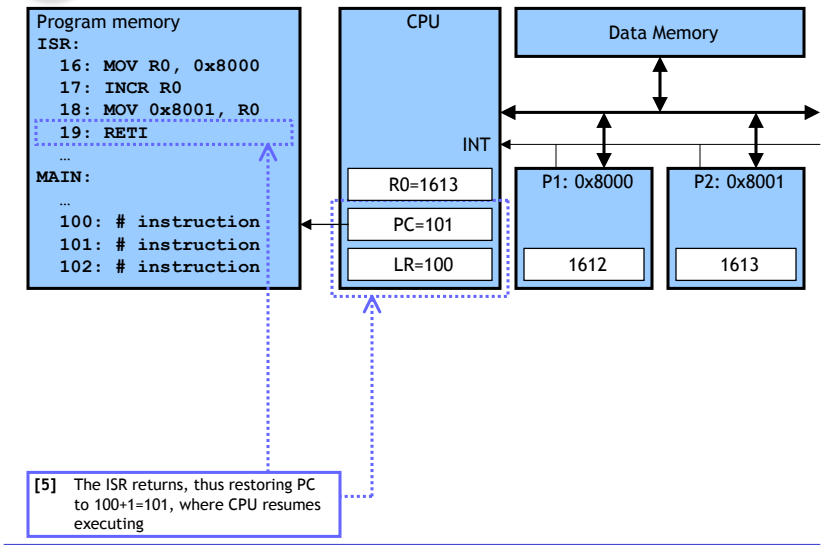
Fixed ISR location



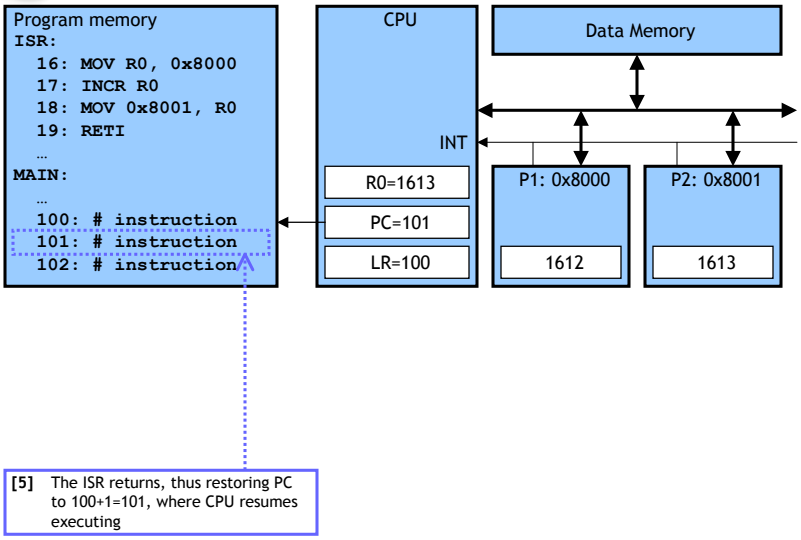
Fixed ISR location



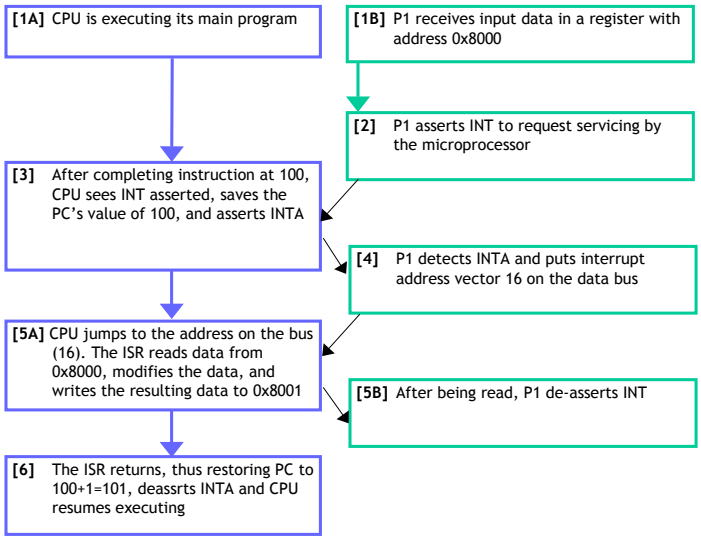
Fixed ISR location



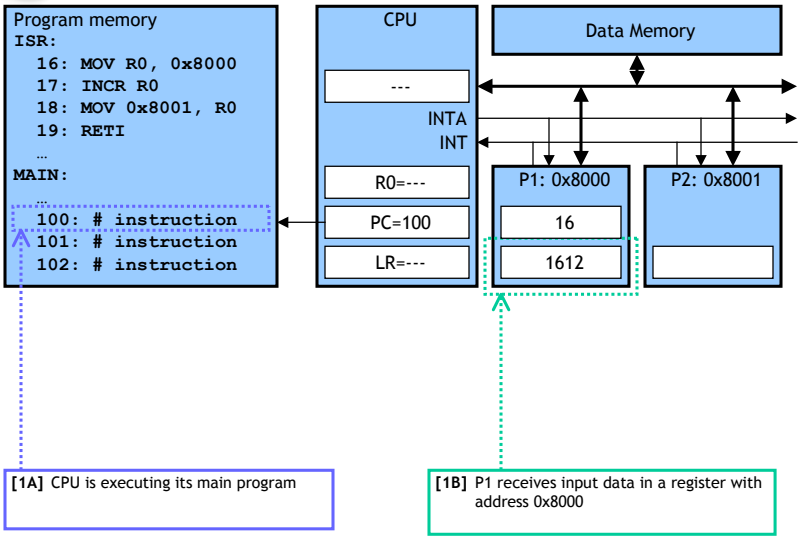
Fixed ISR location



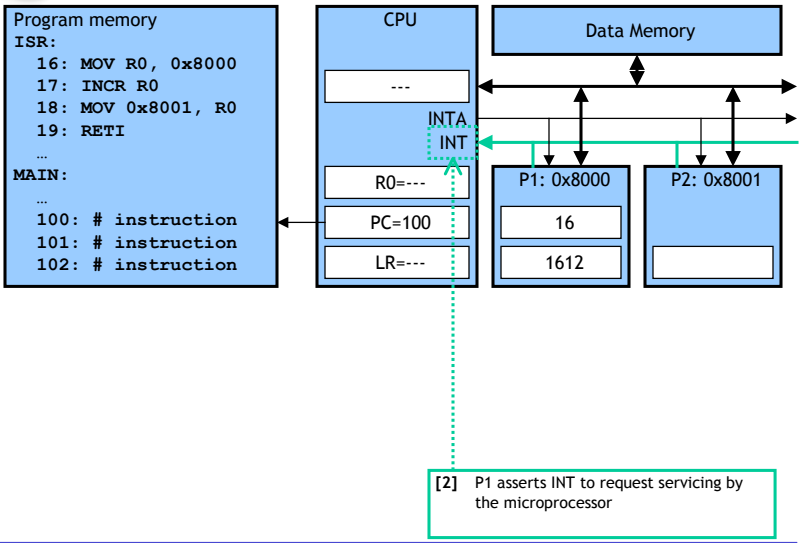
Vectored interrupt



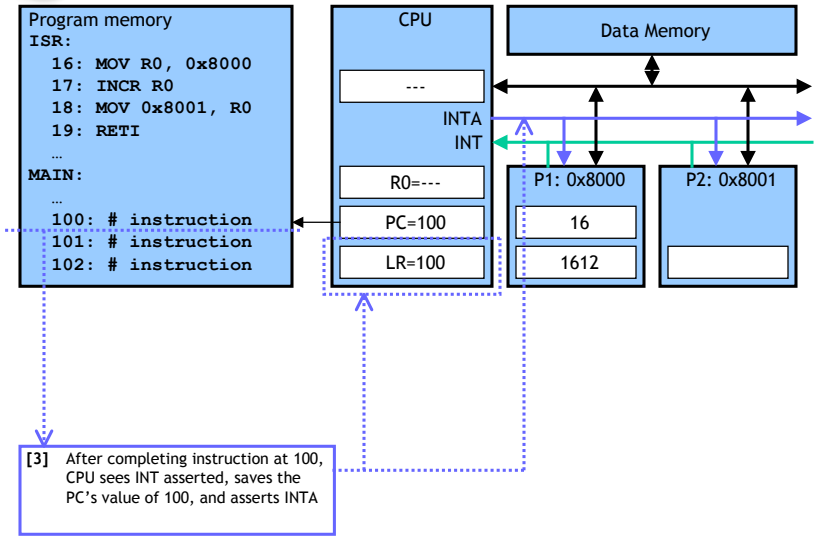
Vectored interrupt



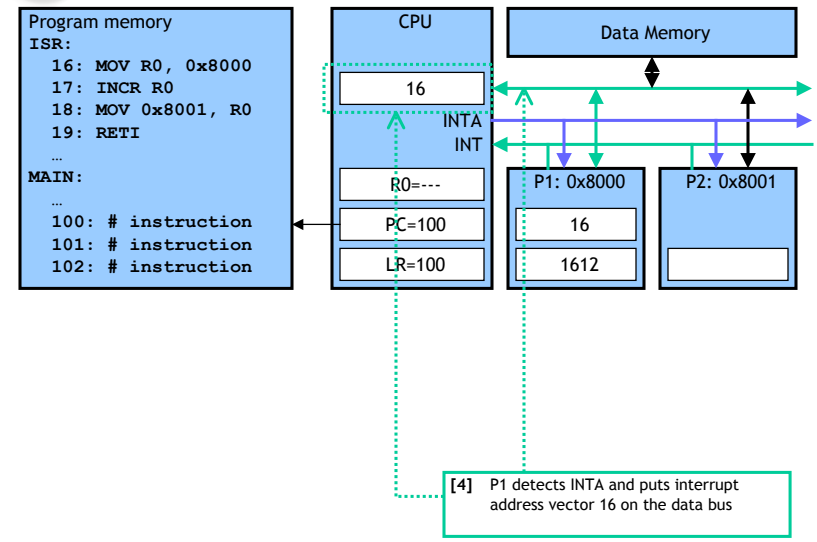
Vectored interrupt



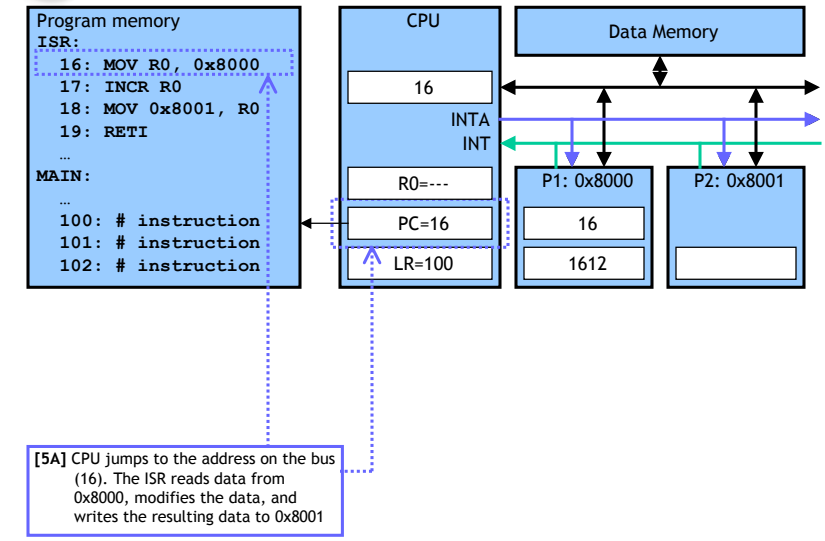
Vectored interrupt



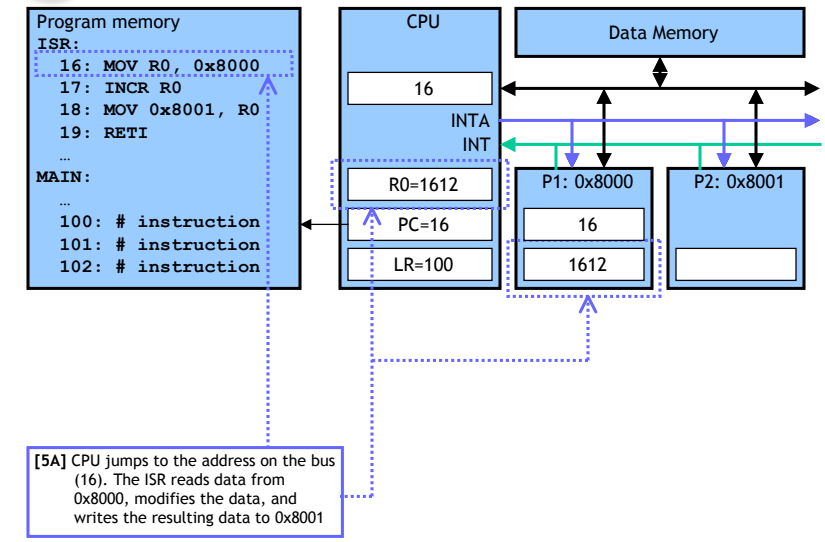
Vectored interrupt



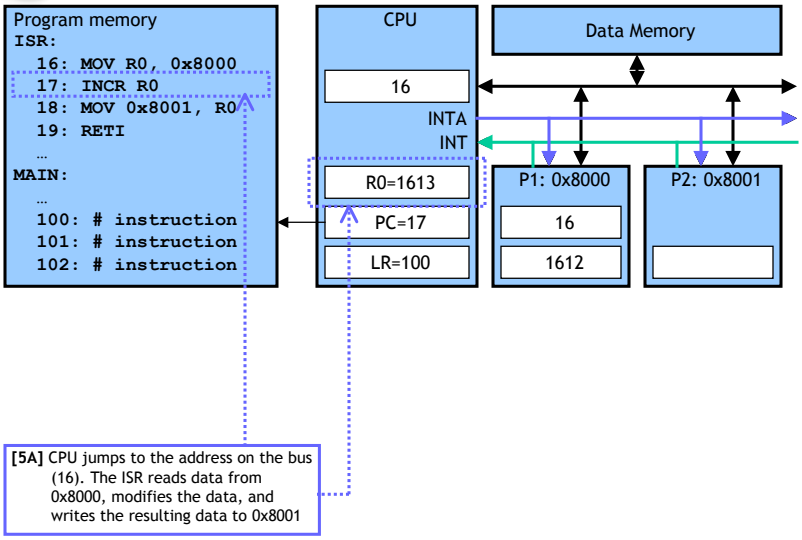
Vectored interrupt



Vectored interrupt

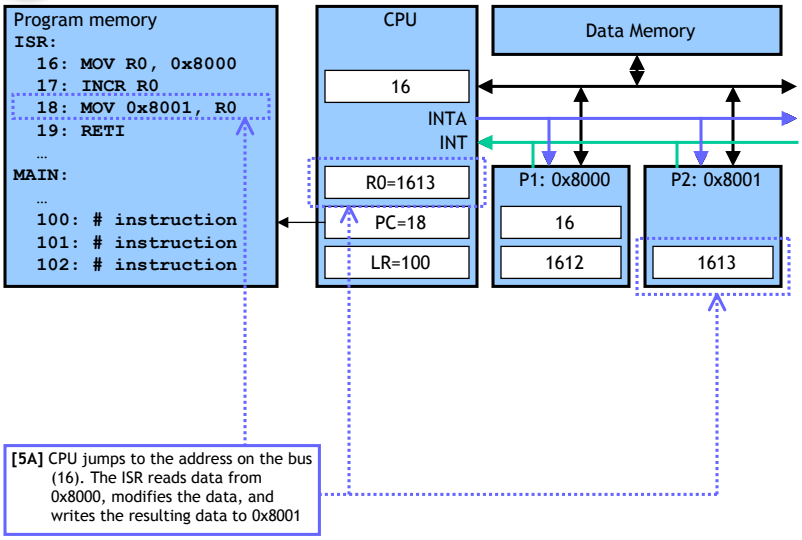


Vectored interrupt



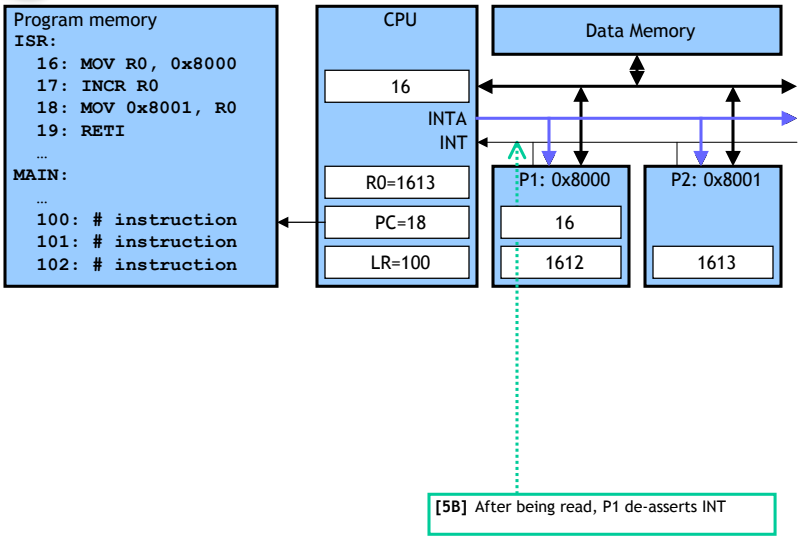
[5A] CPU jumps to the address on the bus (16). The ISR reads data from 0x8000, modifies the data, and writes the resulting data to 0x8001

Vectored interrupt

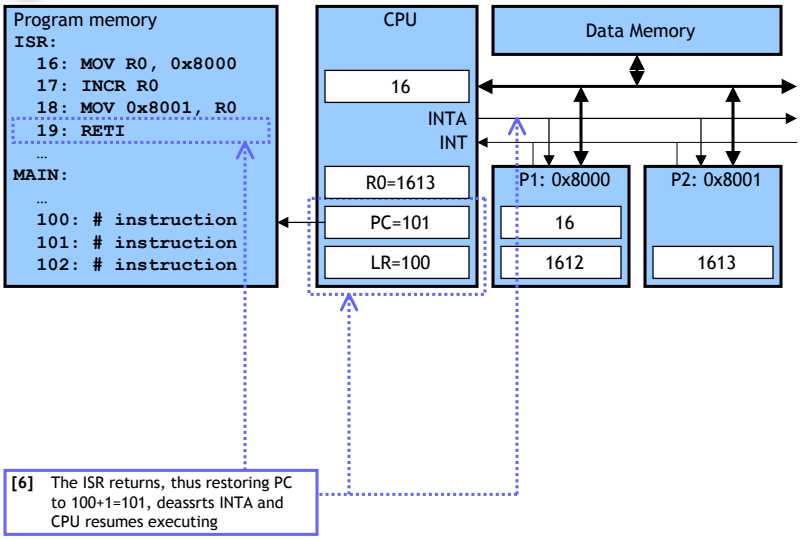


[5A] CPU jumps to the address on the bus (16). The ISR reads data from 0x8000, modifies the data, and writes the resulting data to 0x8001

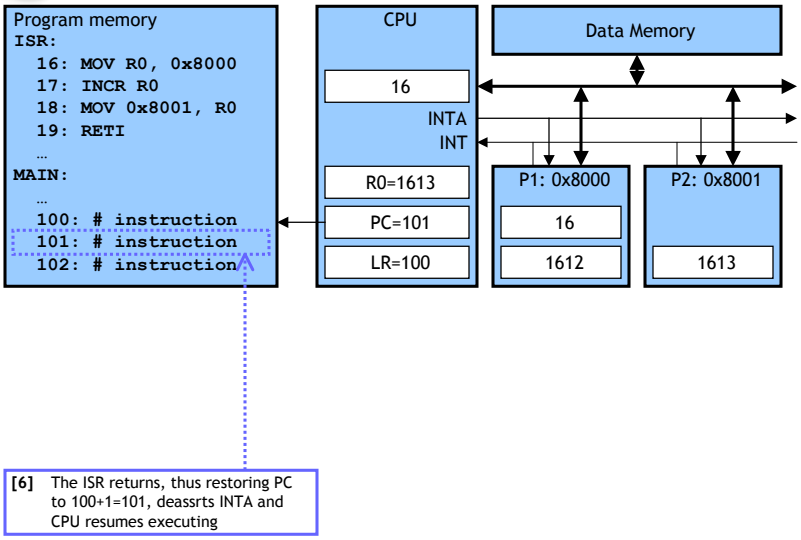
Vectored interrupt



Vectored interrupt



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 (MI) vs. Non-Maskable (NMI) 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

Additional interrupt issues

- 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
  - PC only
    - ISR must not modify registers
    - Or else must save them first
  - Assembly-language programmer must be aware of which registers stored

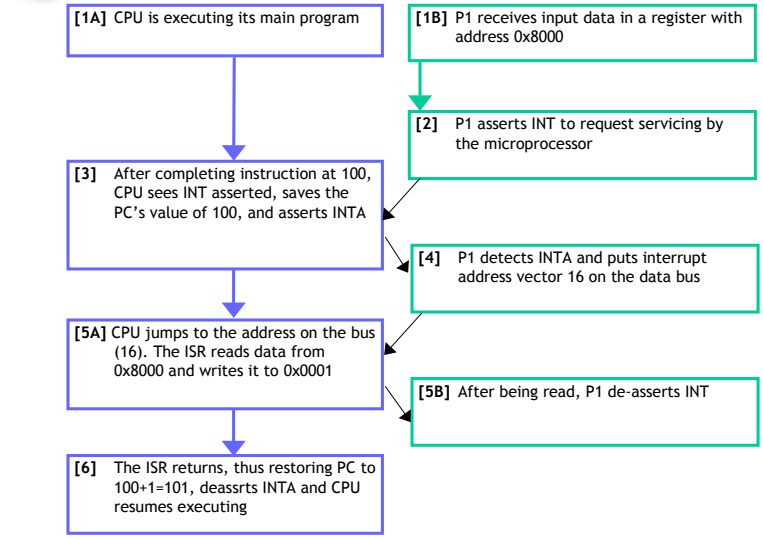
Direct memory access

- 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 (Direct Memory Access) controller is more efficient

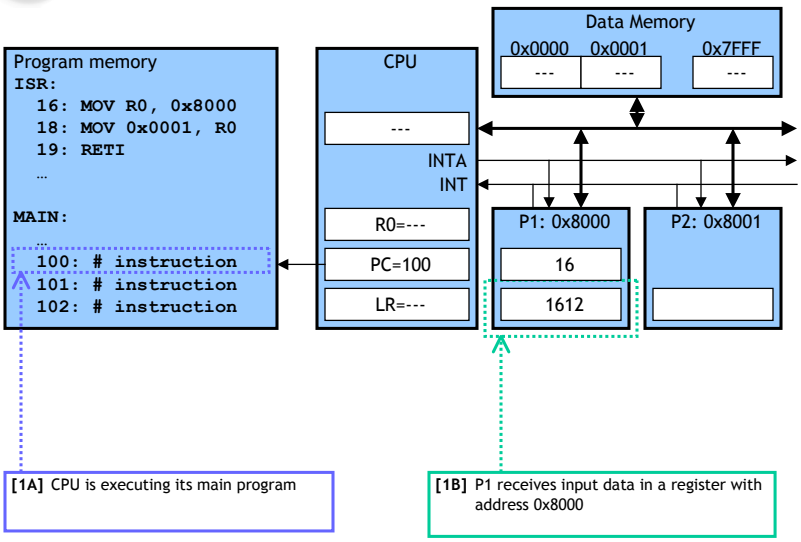
Direct memory access

- Separate single-purpose processor
  - CPU leaves control of system bus to DMA controller
  - CPU can meanwhile execute its regular program
  - No inefficient storing and restoring state due to interrupt service routine call
  - Regular program need not wait unless it requires the system bus
  - For Harvard archictectures
    - Processor can fetch and execute instructions as long as they don't access data memory
    - If they do, processor stalls

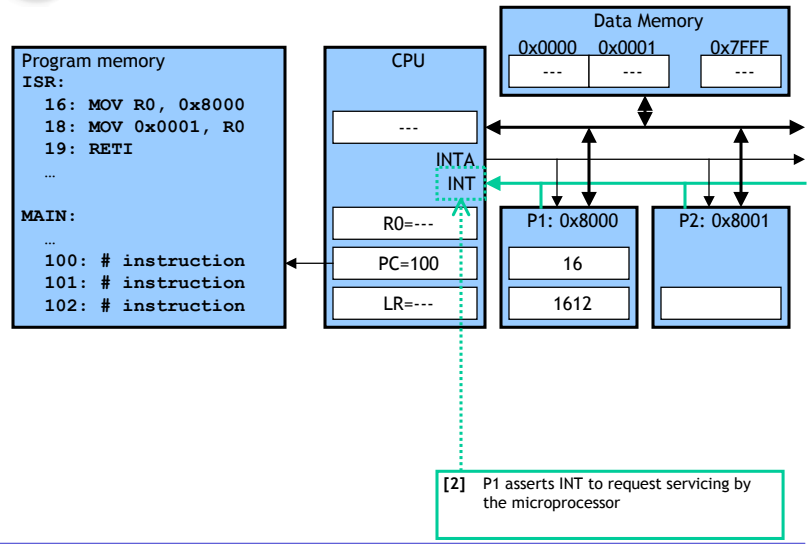
Peripheral to memory transfer with vectored interrupt



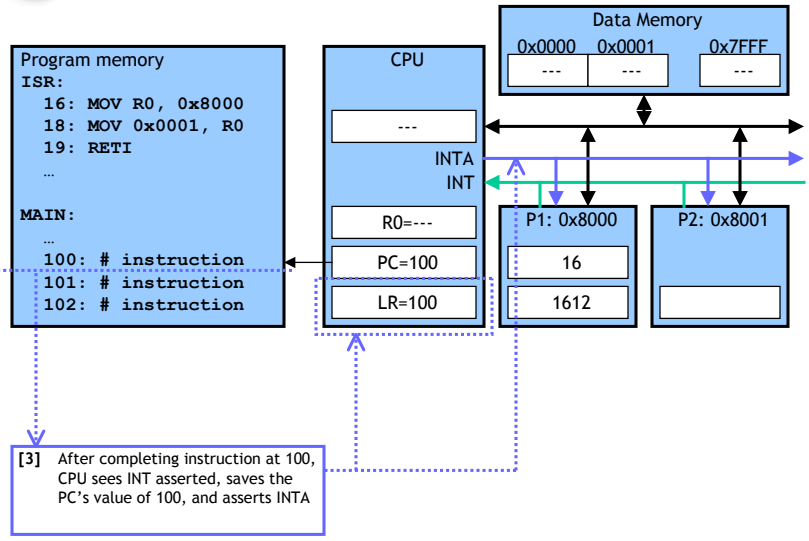
Peripheral to memory transfer with vectored interrupt



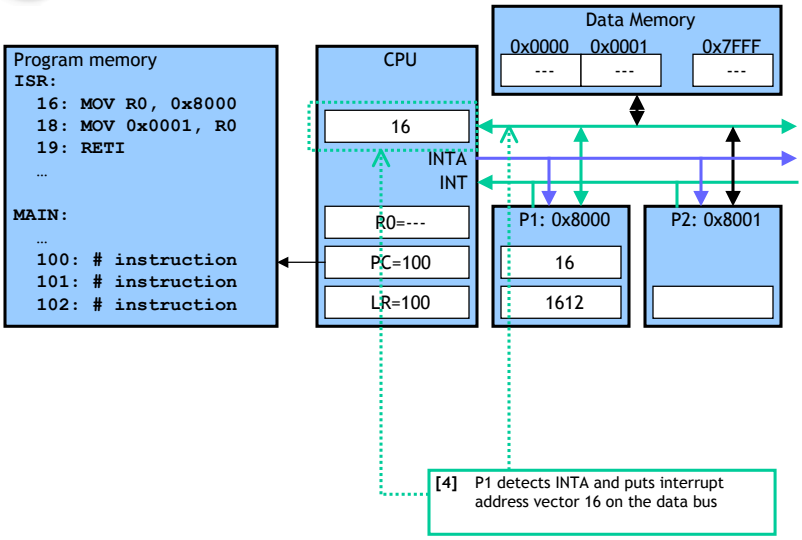
Peripheral to memory transfer with vectored interrupt



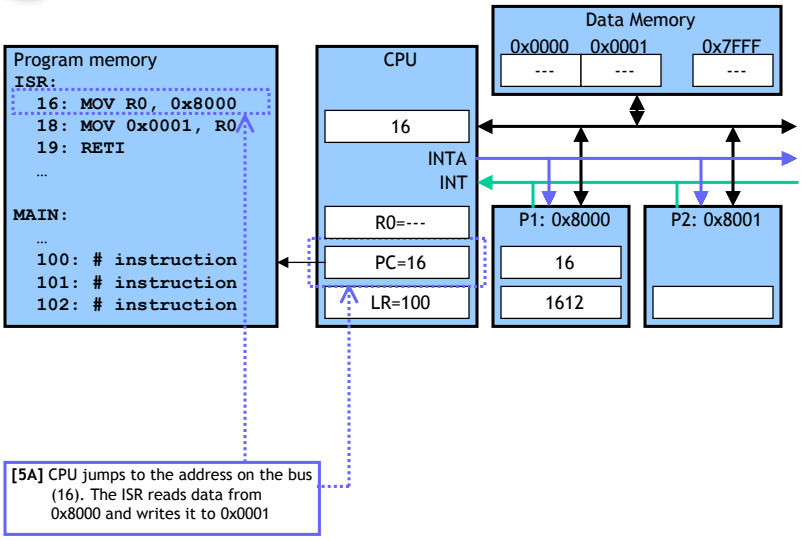
Peripheral to memory transfer with vectored interrupt



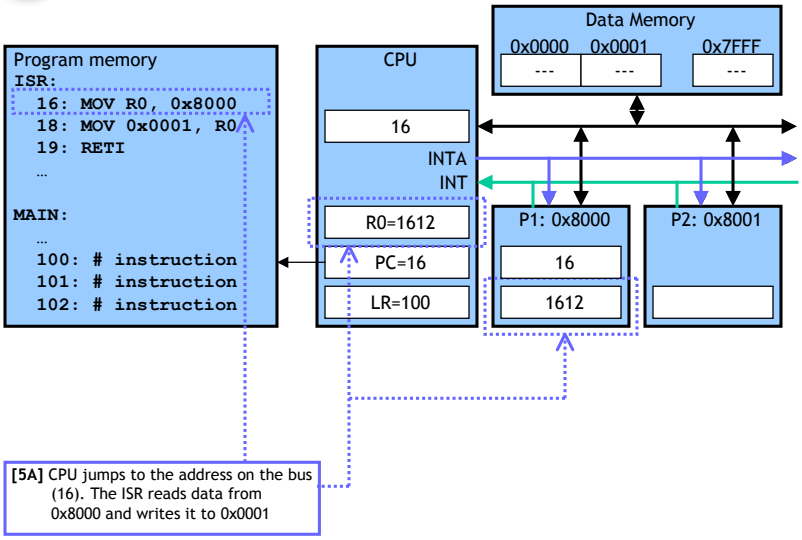
Peripheral to memory transfer with vectored interrupt



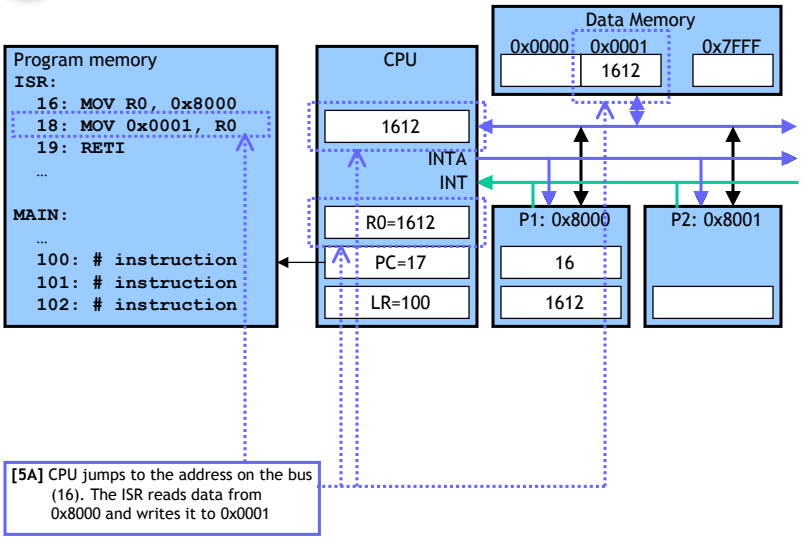
Peripheral to memory transfer with vectored interrupt



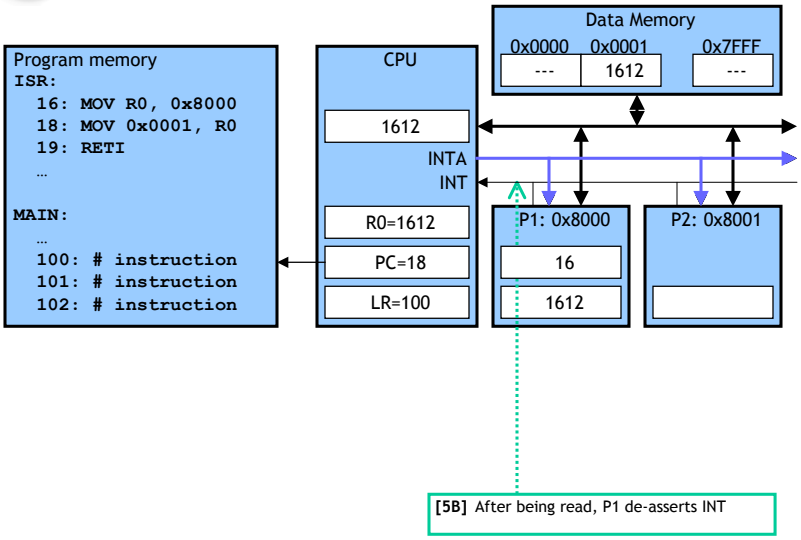
Peripheral to memory transfer with vectored interrupt



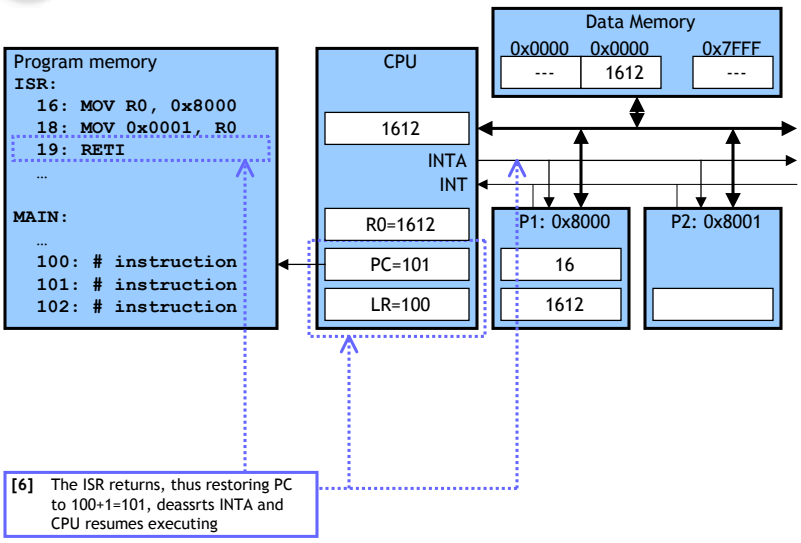
Peripheral to memory transfer with vectored interrupt



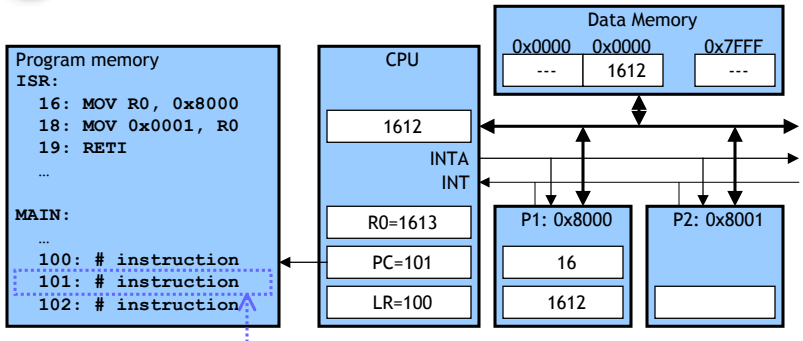
Peripheral to memory transfer with vectored interrupt



Peripheral to memory transfer with vectored interrupt

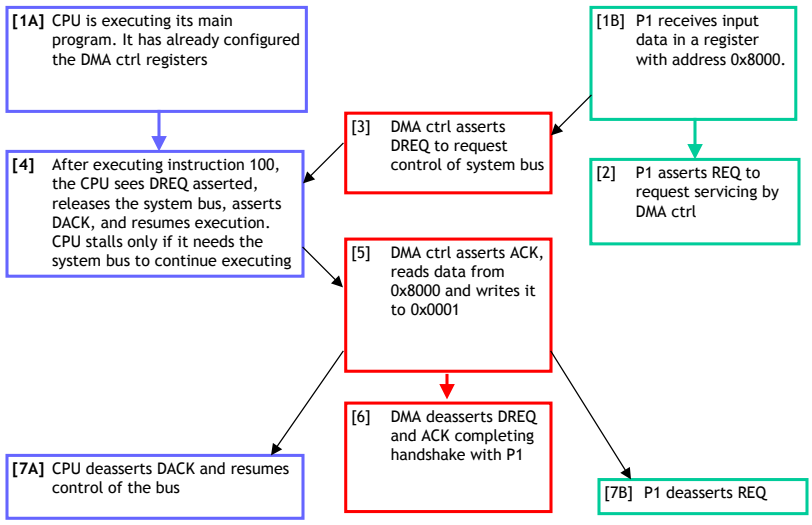


Peripheral to memory transfer with vectored interrupt

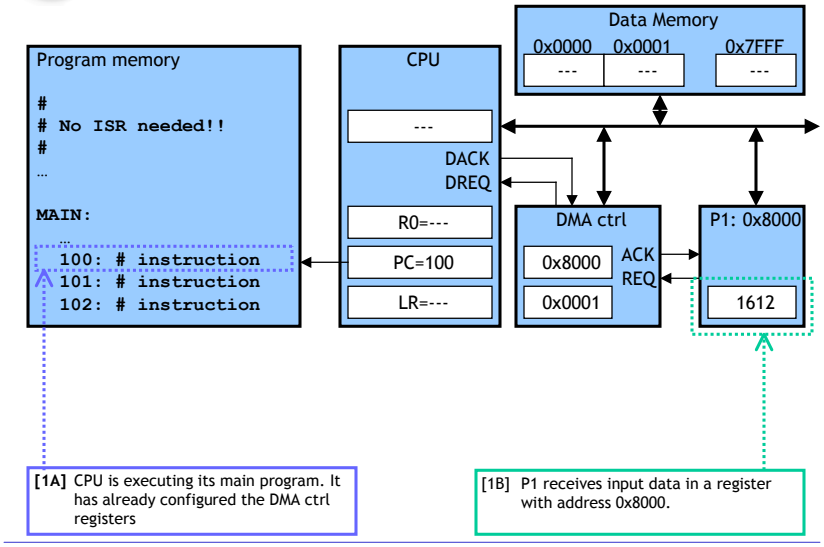


[6] The ISR returns, thus restoring PC to 100+1=101, deassrts INTA and CPU resumes executing

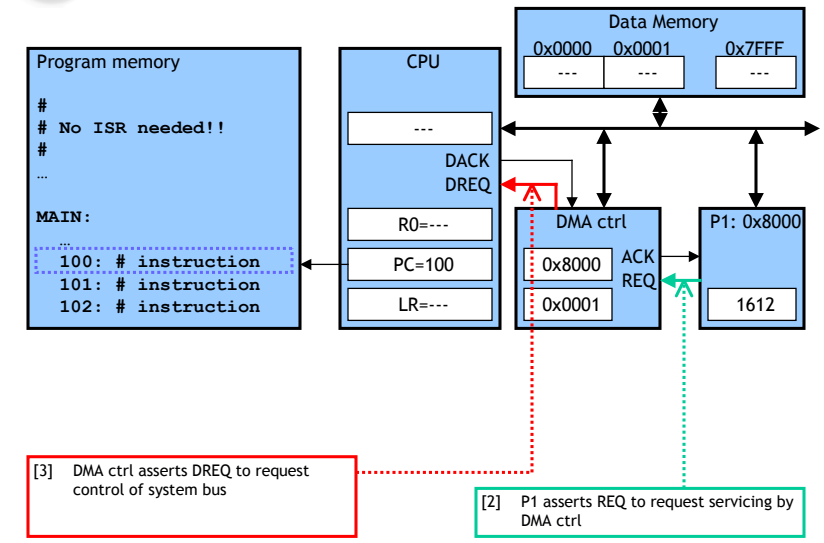
Peripheral to memory transfer with DMA



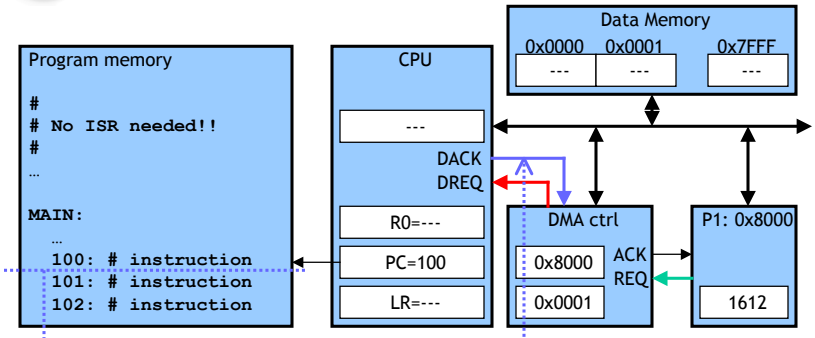
Peripheral to memory transfer with DMA



Peripheral to memory transfer with DMA

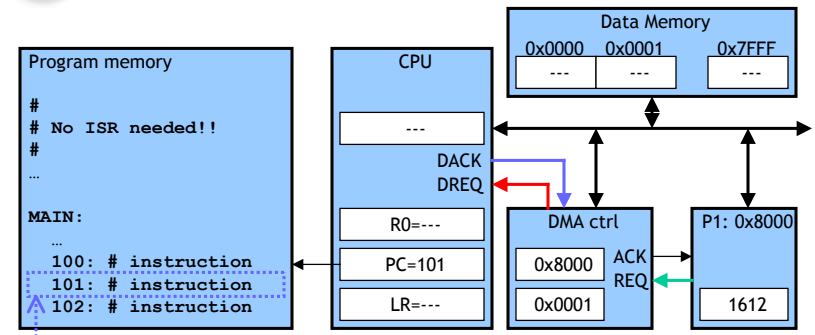


Peripheral to memory transfer with DMA



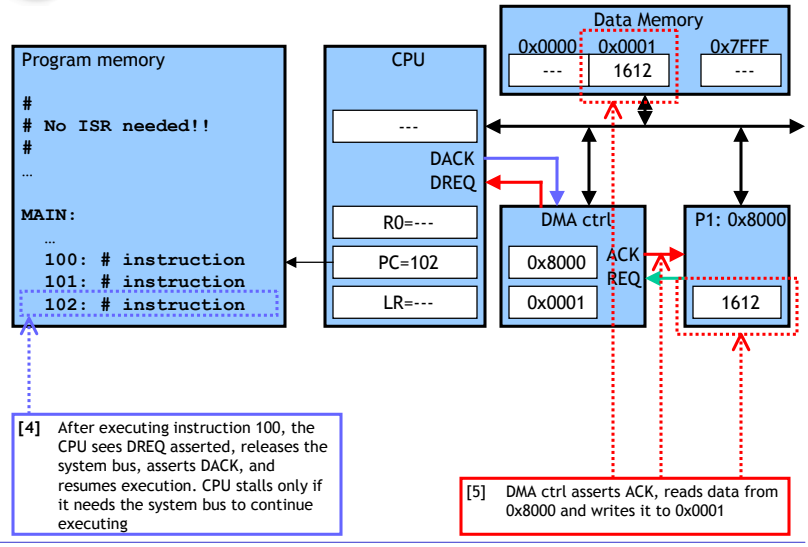
[4] After executing instruction 100, the CPU sees DREQ asserted, releases the system bus, asserts DACK, and resumes execution. CPU stalls only if it needs the system bus to continue executing

Peripheral to memory transfer with DMA

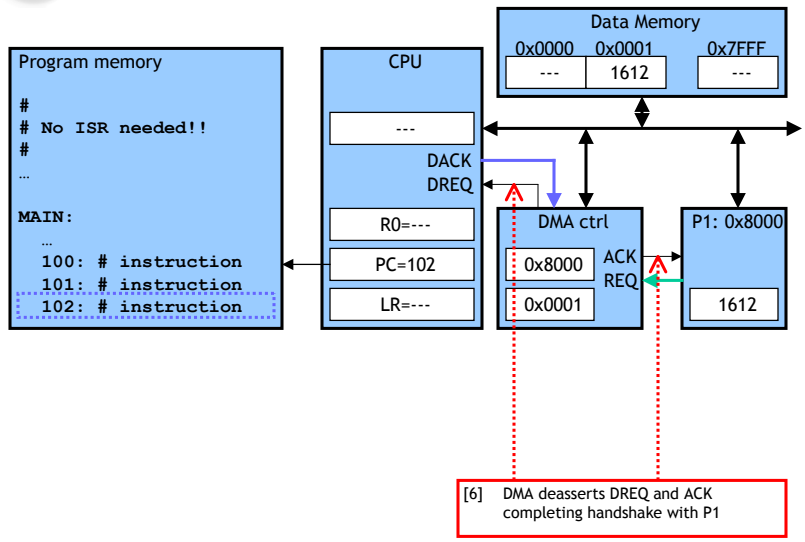


[4] After executing instruction 100, the CPU sees DREQ asserted, releases the system bus, asserts DACK, and resumes execution. CPU stalls only if it needs the system bus to continue executing

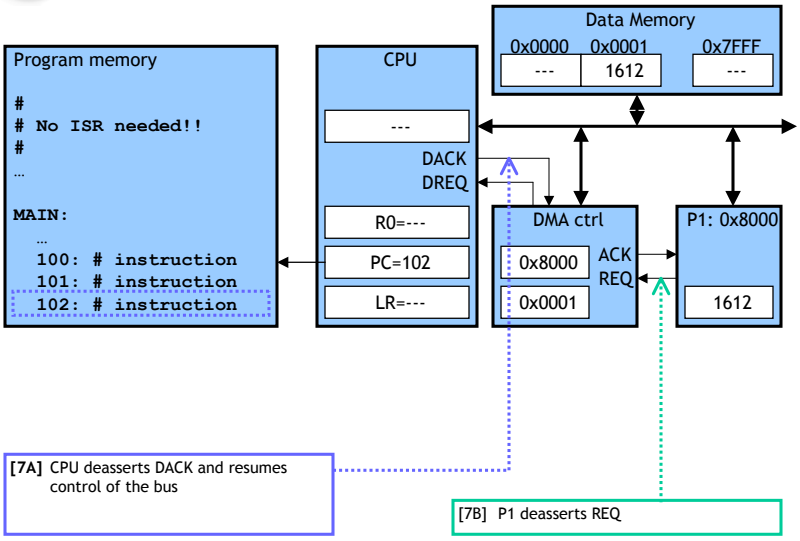
Peripheral to memory transfer with DMA



Peripheral to memory transfer with DMA



Peripheral to memory transfer with DMA

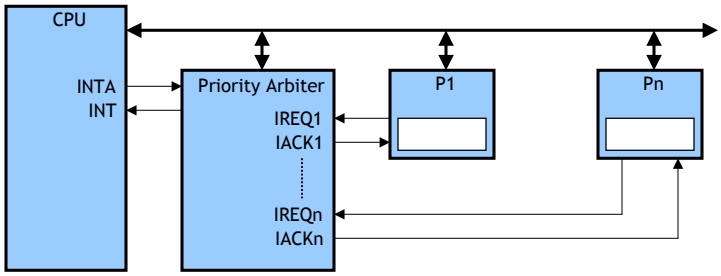


Arbitration

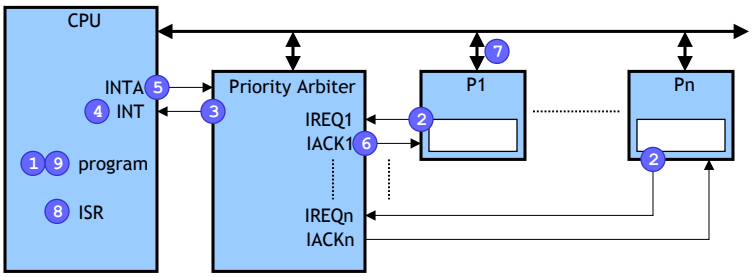
- Multiple peripherals may request services from single resource simultaneously
  - Microprocessor, DMA controller, …
- Who is served first?
  - Need an arbiter mechanism
- Different approaches
  - Priority arbiter
  - Daisy-chain arbiter

Arbitration using a priority arbiter

- Single-purpose processor
  - Peripherals make requests to arbiter
  - Arbiter makes requests to resource
- Arbiter is connected to system bus
  - For configuration purposes only



Arbitration using a priority arbiter



1. Microprocessor is executing its program
2. Peripheral P1 needs servicing so asserts IREQ1; peripheral Pn also needs servicing so asserts IREQn
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 peripheral P1
7. Peripheral P1 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 using a priority arbiter

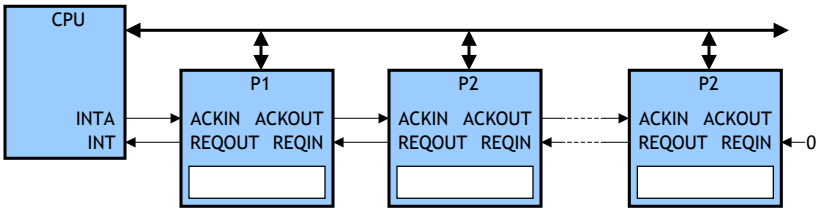
- Different types of priority
- Fixed priority
  - Each peripheral has unique rank
  - Highest rank is chosen first with simultaneous request
  - Preferred when there is a clear difference in rank between peripherals
- Rotating priority (round-robin)
  - Priority is changed based on history of servicing
  - Better distribution of servicing
    - Especially among peripherals with similar priority demands

Arbitration using daisy-chain

- 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
  - 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 using daisy-chain

- Advantages
  - Easy to add/remove peripheral
  - No system redesign needed
- Disadvantages
  - Does not support rotating priority
  - One broken peripheral causes loss of access to others



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
  - This can cause collisions
    - Data must be resent
    - Don't want to start sending again at same time, so statistical methods are 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
  - Some peripherals need
    - High-speed bus interface
    - Processor-specific bus interface
  - Too many peripherals slows down bus
- A solution consists in using a multilevel bus architecture consisting of
  - Processor-local bus
  - Peripheral bus
  - Bridge

Multilevel bus architectures

- Processor-local bus
  - High speed and wide
  - Used for most frequent communication
  - Connects microprocessor, cache, memory controllers
- Peripheral bus
  - Lower speed, narrower
  - Used for less frequent communication
  - Typically industry standard bus for portability
    - ISA, PCI, SCSI
- Bridge
  - Converts communication between busses

Multilevel bus architectures