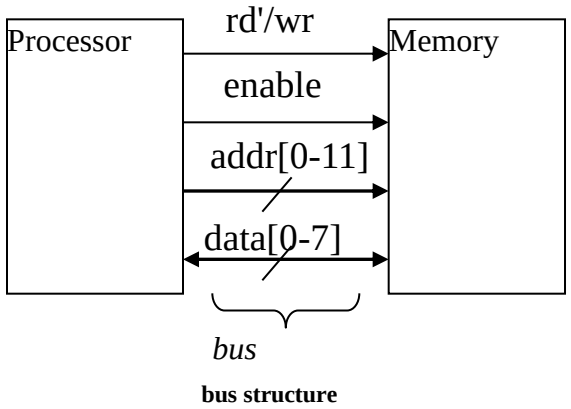
Interfacing

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
      - Read/Write a memory
      - Read/Write peripheral'register
    - Implemented using buses
    - Called *interfacing*

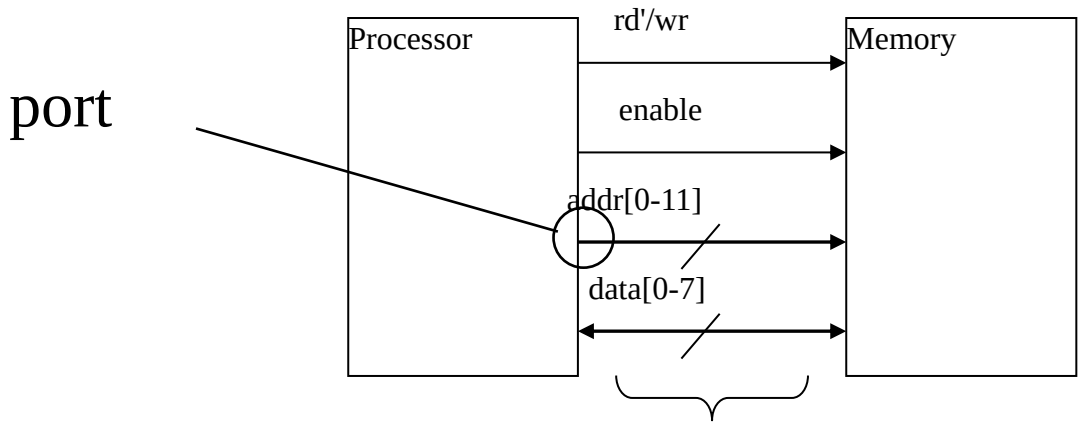
A simple bus

- 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



bus structure

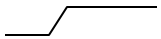
Ports



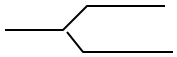
- Conducting device on periphery
- Connects bus to processor or memory
- Often referred to as a *pin*
  - Actual pins on periphery of IC package that plug into socket on printed-circuit board
  - Sometimes metallic balls instead of pins
  - Today, metal "pads" connecting processors and memories within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

Timing Diagrams

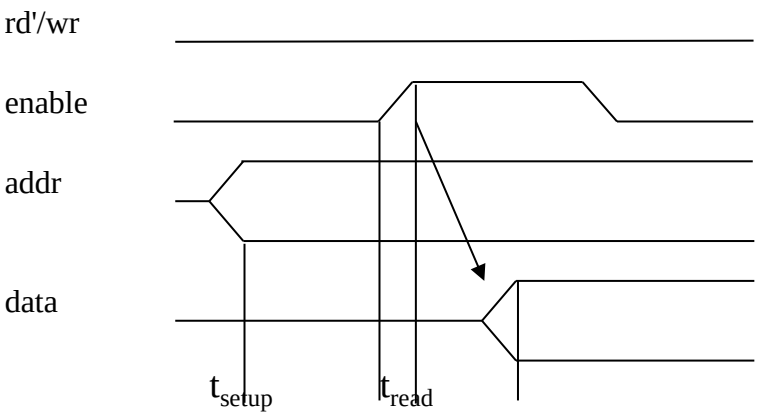
- Most common method for describing a communication protocol
- Time proceeds to the right on x-axis
- Control signal: low or high



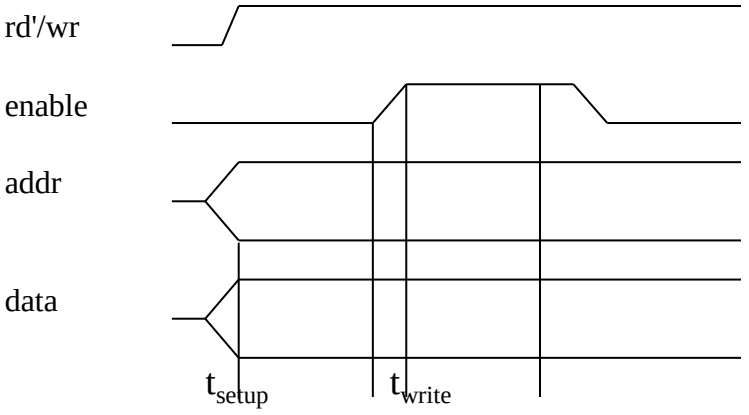
  - May be active low (e.g., go', /go, or go_L)
  - Use terms *assert* (active) and *deassert*
  - Asserting go' means go=0
- Data signal: not valid or valid



- 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 t_{setup} time before *enable* asserted, enable triggers memory to place data on *data* wires by time t_{read}

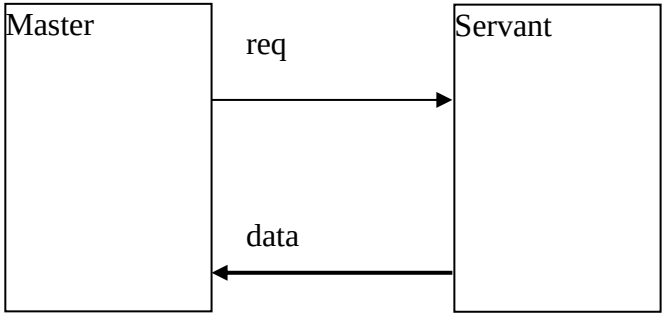


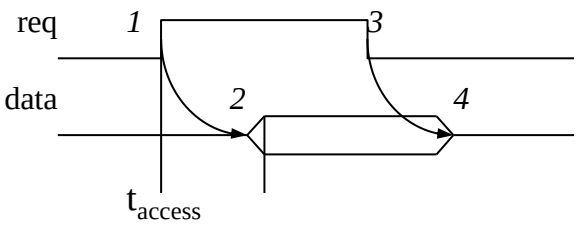
read protocol



write protocol

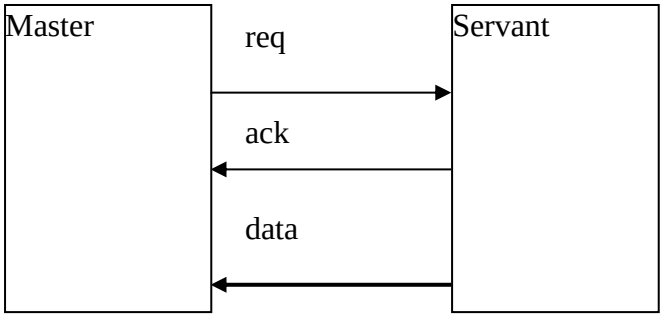
Basic protocol concepts:

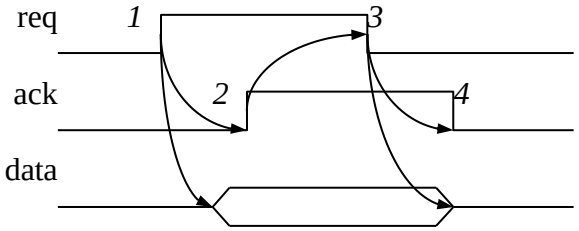




1. Master asserts *req* to receive data
2. Servant puts data on bus **within time t_{access}**
3. Master receives data and deasserts *req*
4. Servant ready for next request

Strobe protocol



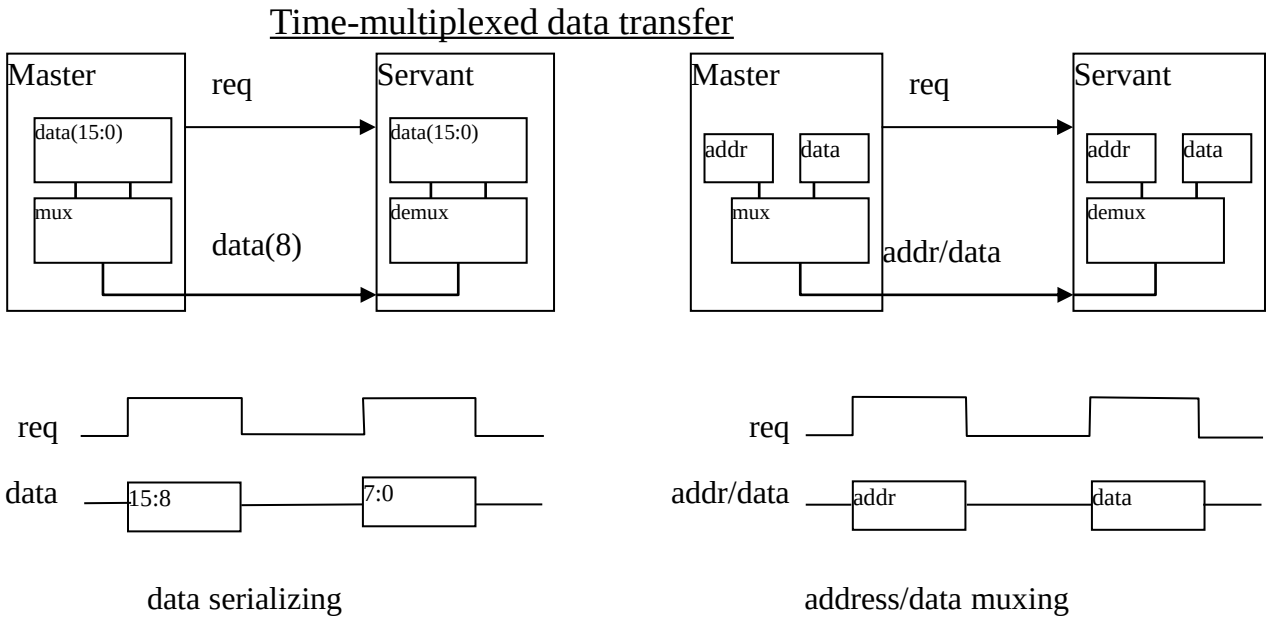


1. Master asserts *req* to receive data
2. Servant puts data on bus **and asserts *ack***
3. Master receives data and deasserts *req*
4. Servant ready for next request

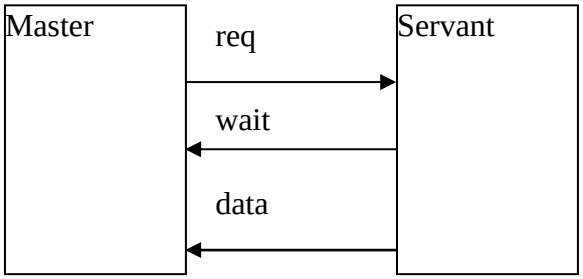
Handshake protocol

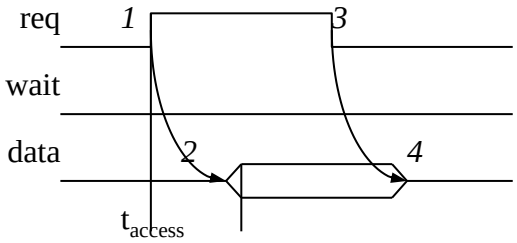
Basic protocol concepts

- 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



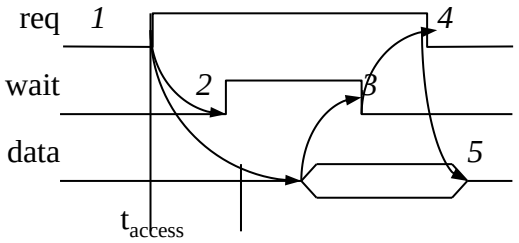
A strobe/handshake compromise





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

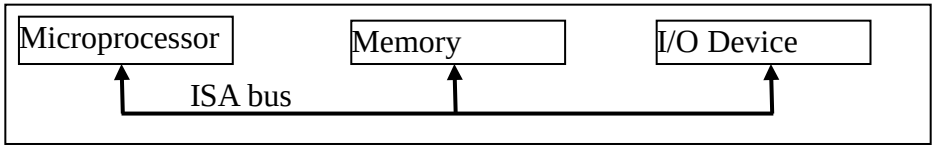


1. Master asserts *req* to receive data
2. Servant can't put data within t_{access}, **asserts *wait*** ack
3. Servant puts data on bus and **deasserts *wait***
4. Master receives data and deasserts *req*
5. Servant ready for next request

Slow-response case

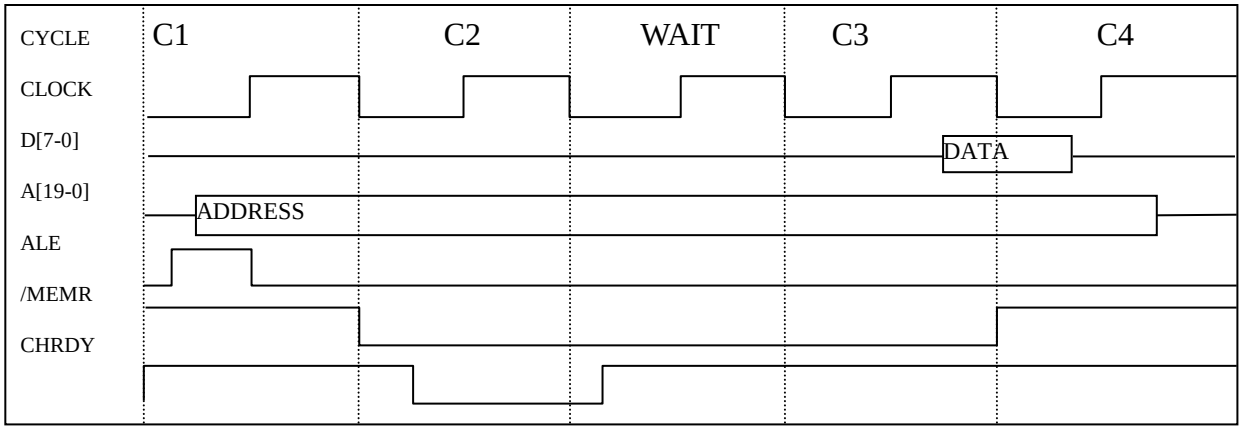
ISA bus protocol – memory access

- 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)

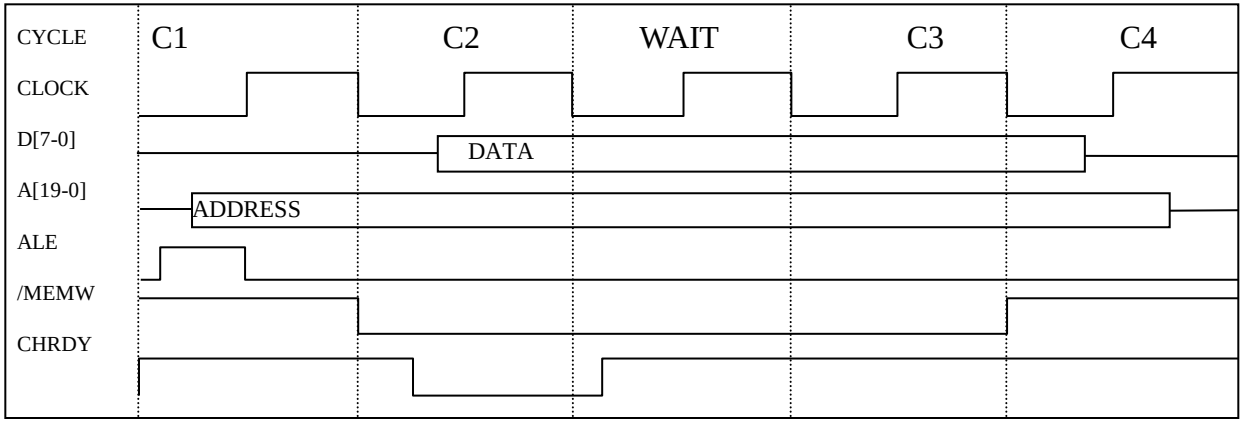


ISA bus protocol – memory access

memory-read bus cycle

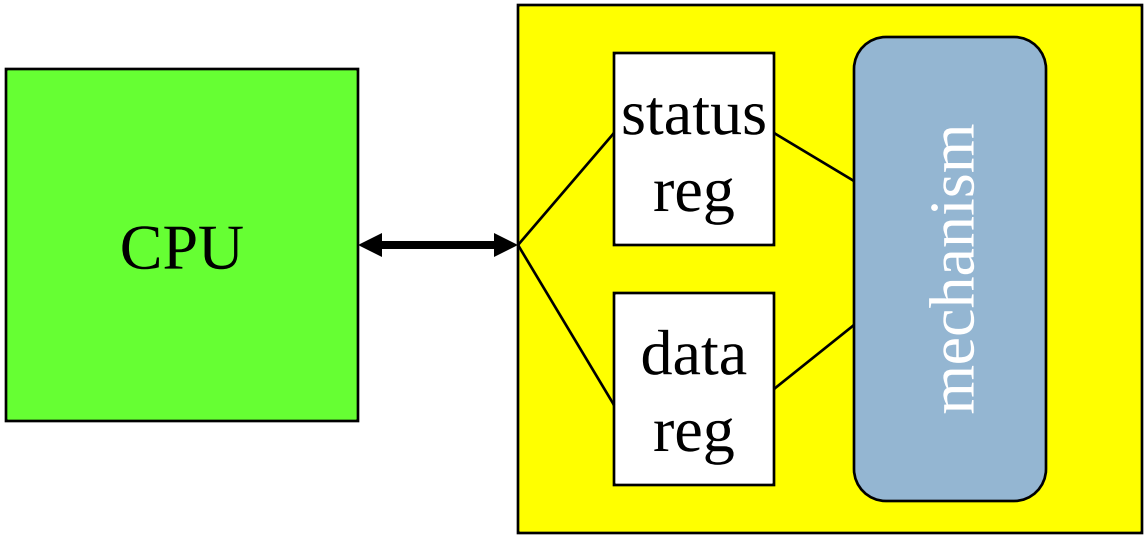


memory-write bus cycle



I/O devices

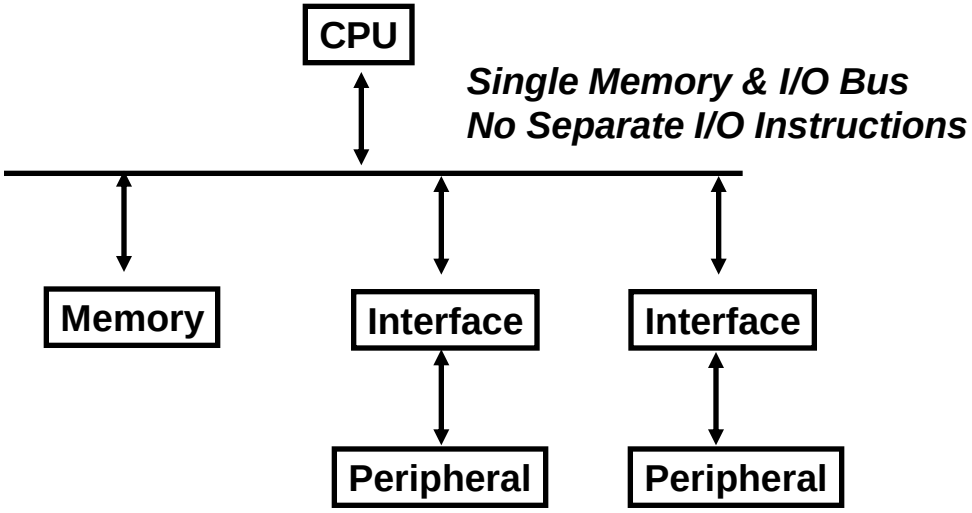
- Usually includes some non-digital component.
- Typical digital interface to CPU:

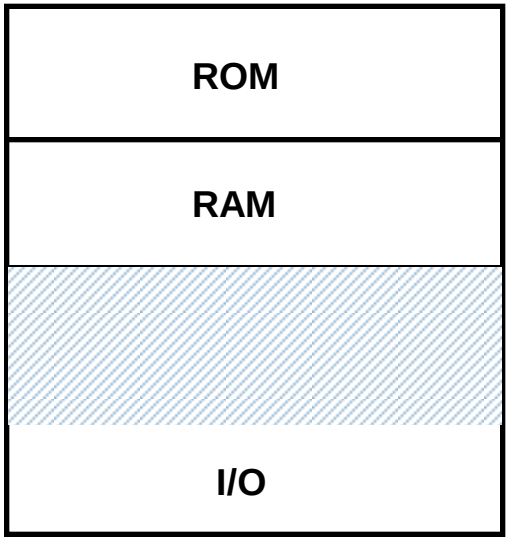


Types of bus-based I/O: memory-mapped I/O and standard I/O

- 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

Memory Mapped I/O

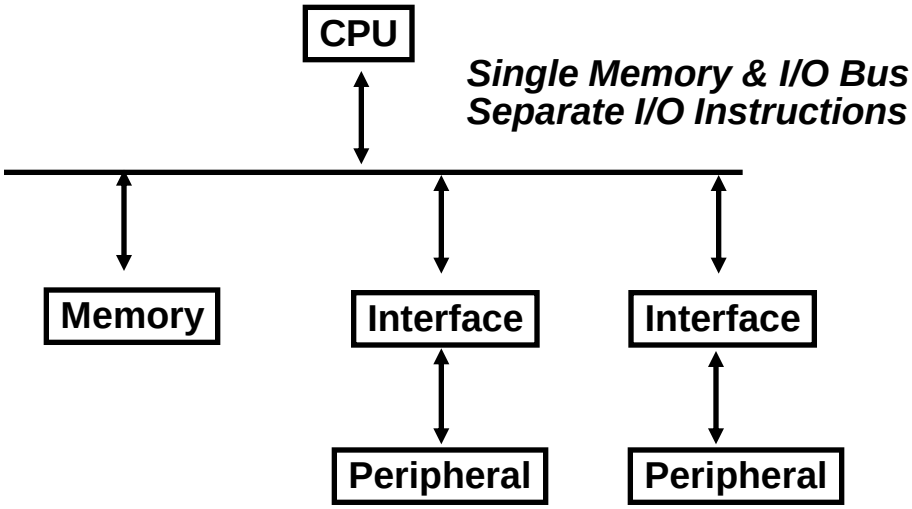


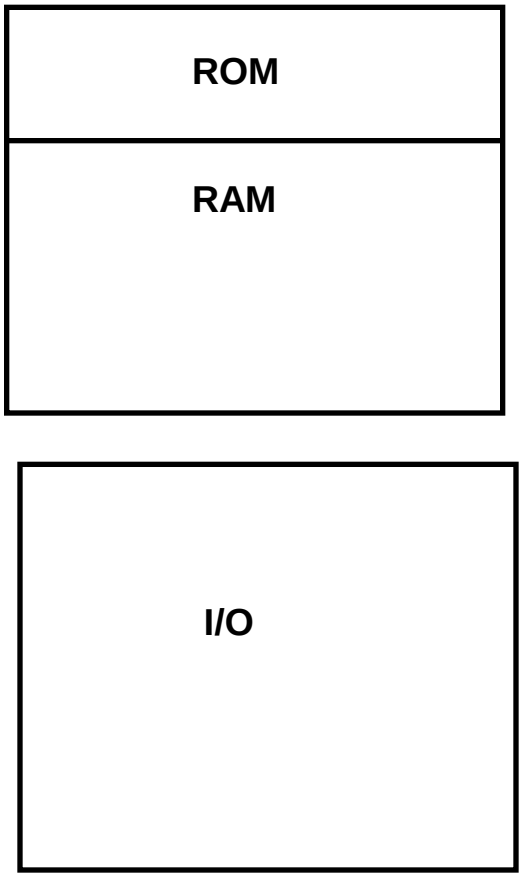


Types of bus-based I/O: memory-mapped I/O and standard I/O

- 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

Standard I/O (I/O-mapped I/O)

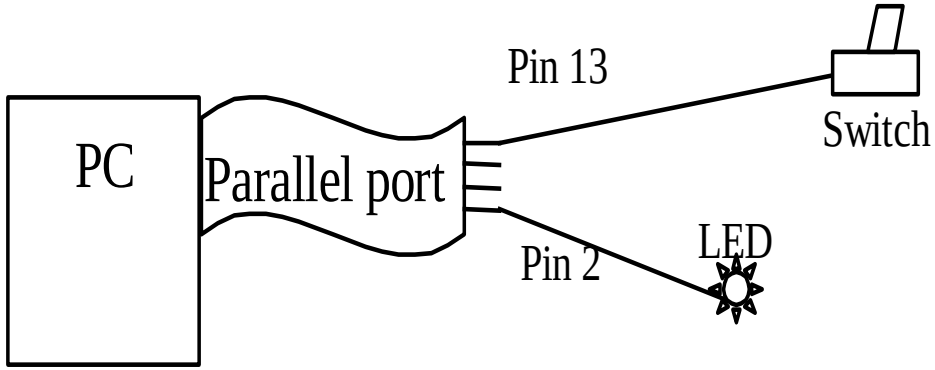




Memory-mapped I/O vs. Standard I/O

- 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
  - Special-purpose I/O instructions
  - Intel x86 provides `in`, `out` instructions.
    - (Es. IN AL, port e OUT port, AL with port= interface address)
  - 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

Intel x86 Standard I/O



LPT Connection Pin	I/O Direction	Register Address
1	Output	0^{th} bit of register #2
2-9	Output	0^{th} bit of register #0
10,11,12,13,15	Input	6,7,5,4,3^{th} bit of register #1
14,16,17	Output	1,2,3^{th} bit of register #2

LPT address = **3BCh**

Intel x86 Standard I/O

```
; This program consists of a sub-routine that reads the state of the input pin,
determining the on/off state of our switch and asserts the output pin, turning
the LED on/off accordingly
.386

CheckPort    proc
    push ax             ; save the content
    push dx             ; save the content
    mov  dx, 3BCh + 1   ; base + 1 for register #1
    in   al, dx         ; read register #1
    and  al, 10h        ; mask out all but bit # 4
    cmp  al, 0          ; is it 0?
    jne  SwitchOn       ; if not, we need to turn the LED on
```

Intel x86 Standard I/O

```
SwitchOff:
    mov dx, 3BCh + 0 ; base + 0 for register #0
    in  al, dx       ; read the current state of the port
    and al, feh      ; clear first bit (masking)
    out dx, al       ; write it out to the port
    jmp Done            ; we are done

SwitchOn:
    mov dx, 3BCh + 0 ; base + 0 for register #0
    in  al, dx       ; read the current state of the port
    or  al, 01h      ; set first bit (masking)
    out dx, al       ; write it out to the port

Done:   pop dx       ; restore the content
        pop ax       ; restore the content
CheckPort   endp
```

ARM memory-mapped I/O

- Define location for device:

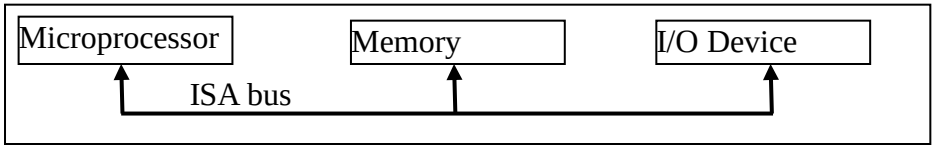
```
DEV1 EQU 0x1000
```

- Read/write code:

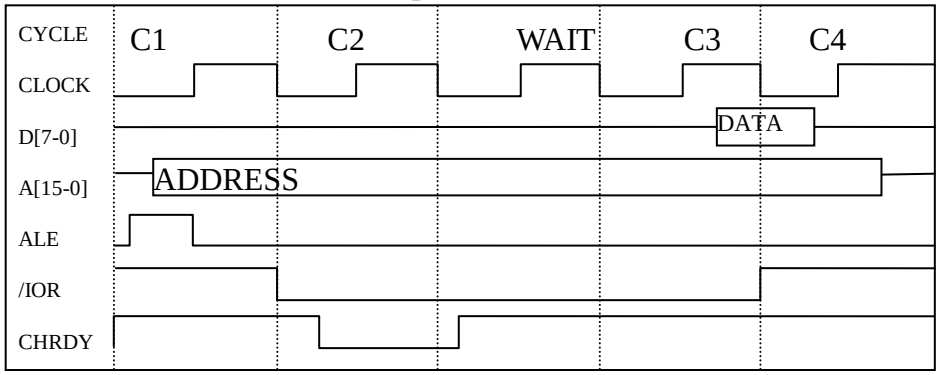
```
LDR r1,#DEV1 ; set up device adrs
LDR r0,[r1] ; read DEV1

LDR r0,#8 ; set up value to write
STR r0,[r1] ; write value to device
```

ISA bus

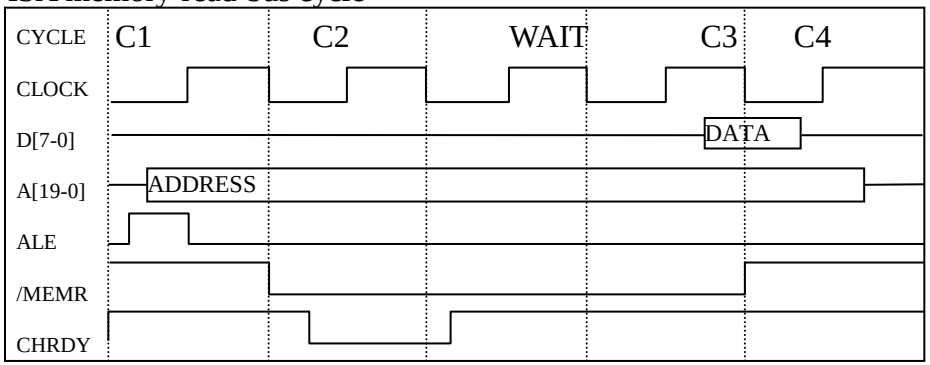
- ISA supports standard I/O
  - /IOR distinct from /MEMR for peripheral read
    - /IOW used for writes
  - 16-bit address space for I/O vs. 20-bit address space for memory
  - Otherwise very similar to memory protocol



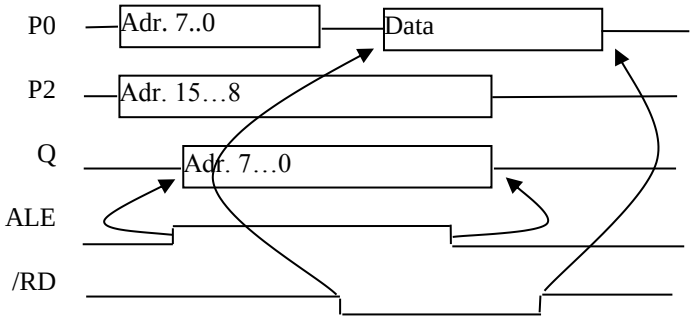
ISA I/O bus read protocol

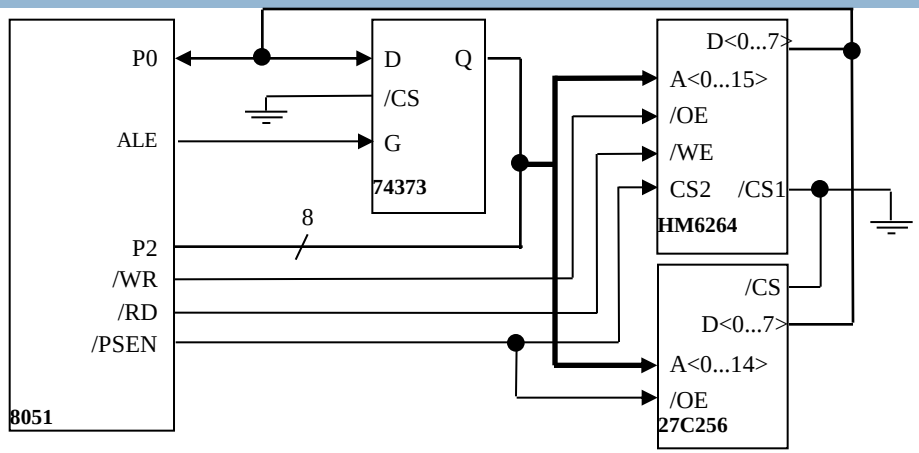


ISA memory-read bus cycle



A basic memory protocol





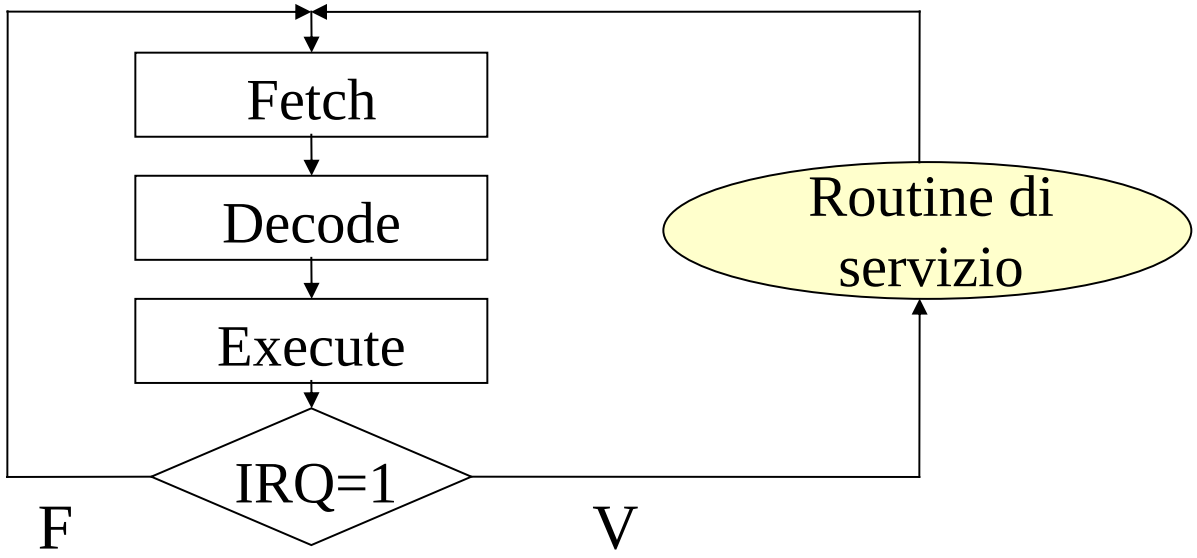
- Interfacing an 8051 to external memory
  - Ports P0 and P2 support port-based I/O when 8051 internal memory being used
  - Those ports serve as data/address buses when external memory is being used
  - 16-bit address and 8-bit data are time multiplexed; low 8-bits of address must therefore be latched with aid of ALE signal

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 – 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!

Sistema di interruzione

- La richiesta di interruzione di un dispositivo di I/O è asincrona rispetto all'esecuzione delle istruzioni
  - Non è associata ad alcuna istruzione e può essere attivata durante l'esecuzione di ogni istruzione
  - Viene valutata solo alla fine dell'esecuzione di ogni istruzione



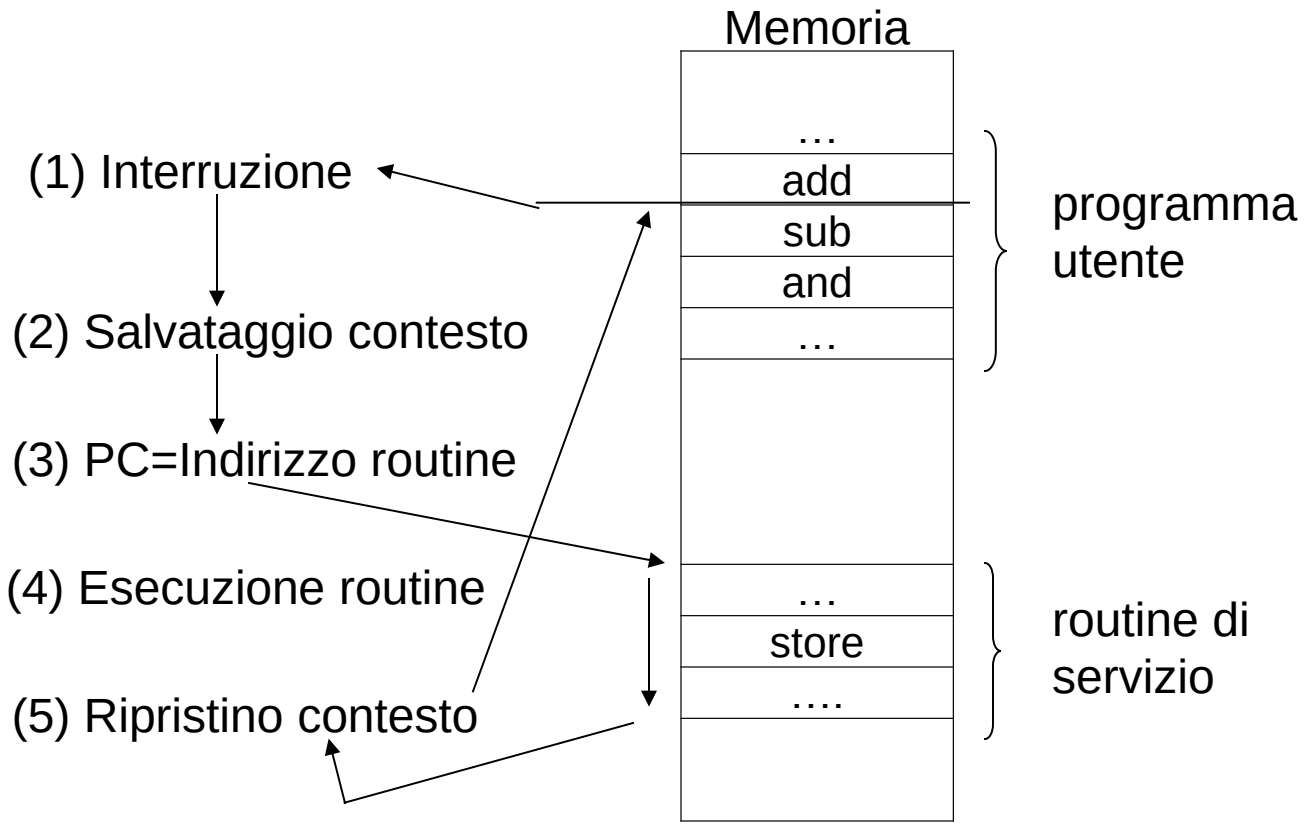
Funzioni del sistema di interruzione

F1. Deve garantire che una interruzione non provochi interferenze sul programma interrotto.

F2. È necessario che il sistema di interruzione riconosca il dispositivo interrompente

F3. Deve provvedere alla gestione delle priorità delle richieste di interruzioni

Cambio di contesto



Salvataggio del contesto

- **Contesto:** Program Counter (PC), Registro di Stato (SR), Registri di uso generale

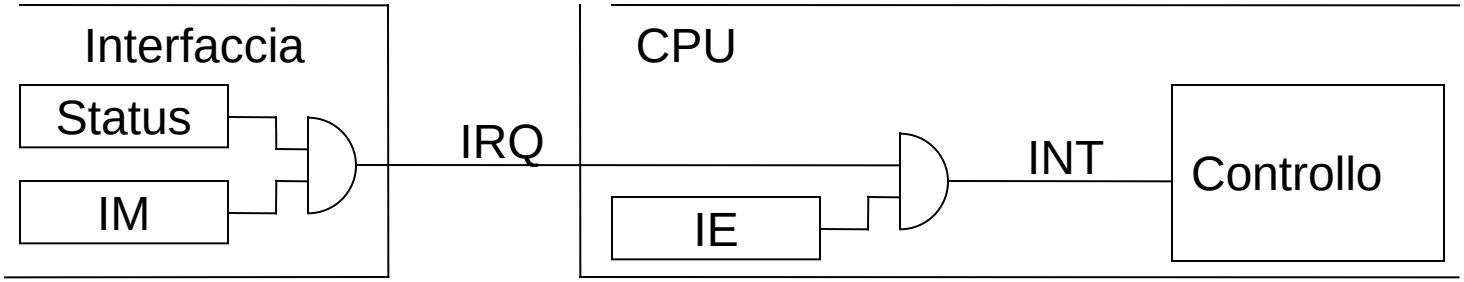
- PC e SR devono essere salvati via hardware.
- Esempio:
  - MEM[SP]=SR; SP--;
  - MEM[SP]=PC; SP--;

- Gli altri registri possono essere salvati via software
  - Vengono salvati solo i registri che verranno utilizzati.
  - Questo compito è demandato alla routine di servizio che lo svolge nel suo preambolo.

Salvataggio del contesto

- Il salvataggio del contesto deve essere non interrompibile per evitare situazioni anomale
  - Il processore viene dotato di un flag IE indicante la interrompibilità del processore.
  - Nel momento in cui viene accettata la richiesta di interruzione IE viene resettato e il processore diventa non interrompibile
  - Per rendere nuovamente interrompibile il processore bisogna usare un'apposita istruzione

Un sistema di interruzione



IM=Interrupt Mask

- Quando il dispositivo è pronto, pone STATUS=1
- Se IM=1 e le interruzioni sono abilitate (IE=1) viene servita la richiesta di interruzione (al termine dell'istruzione corrente).
  - Viene posto IE=0
  - Viene salvato il contesto
  - PC=Indirizzo della Routine di servizio

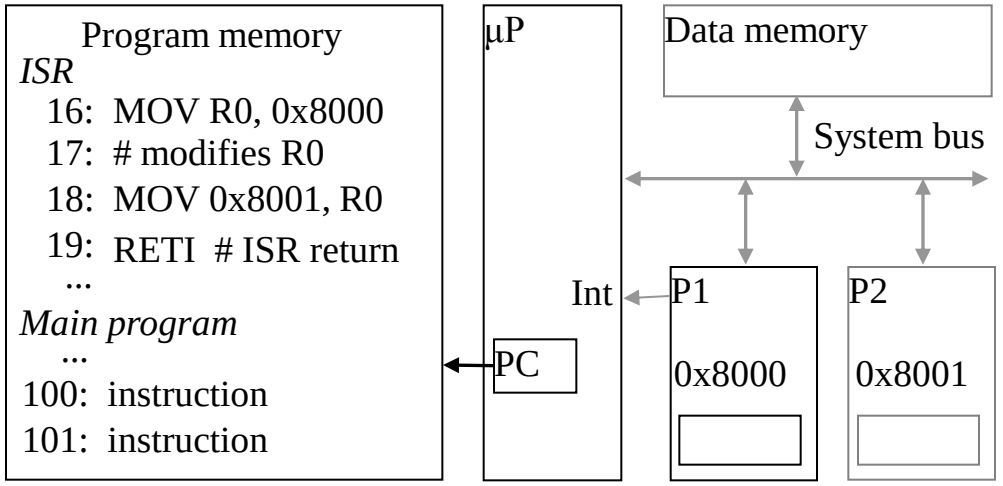
Ripristino del contesto

- Via software, nell'epilogo della routine di servizio, vengono ripristinati i valori dei registri salvati nello stack (POP)
- L'uscita dalla routine di servizio avviene mediante un'apposita istruzione di ritorno da interruzione (RTI) che ripristina la parte di contesto salvata via hardware
- Esempio
  - SR=MEM[SP]; SP++;
  - PC=MEM[SP]; SP++;
- In alcuni processori la RTI riabilita anche il flip flop IE (IE=1), in altri è necessario utilizzare un'apposita istruzione;

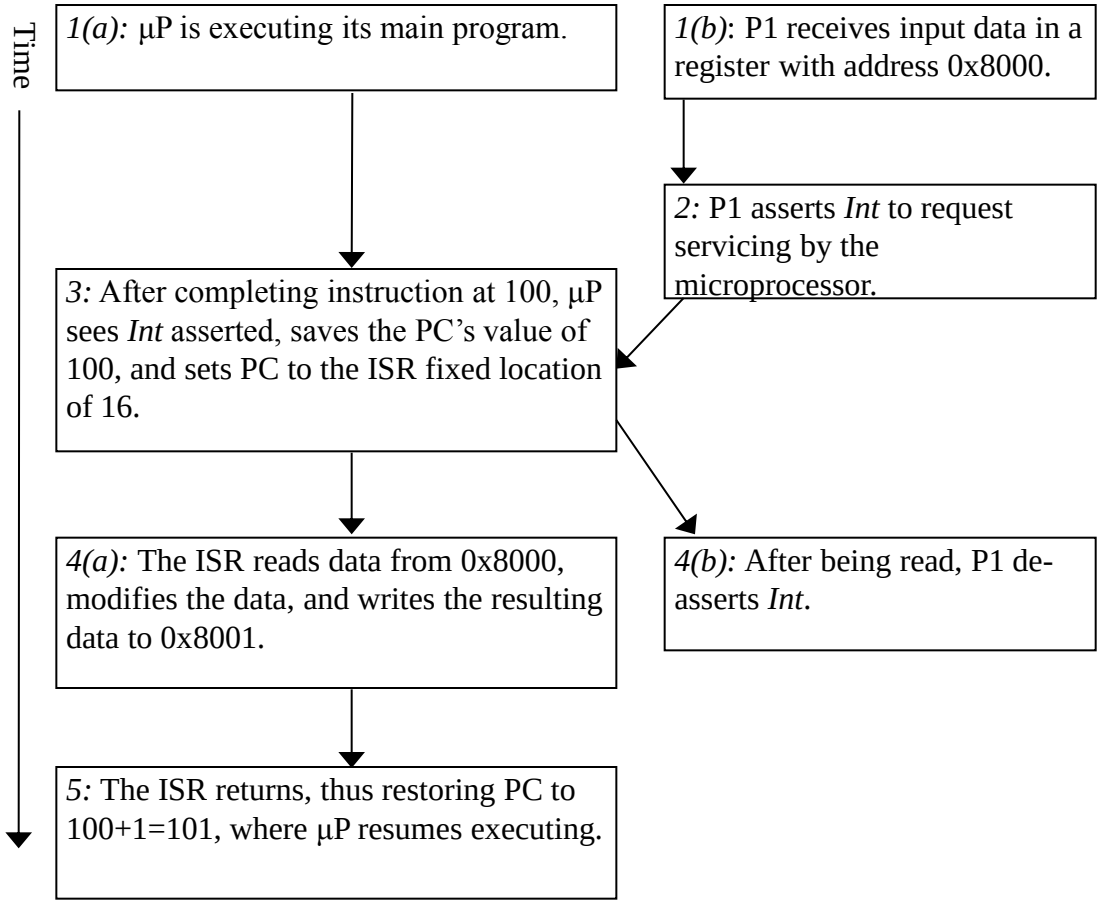
Microprocessor interfacing: interrupts

- What is the address of the Interrupt Service Routine?
  - 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
    - Multiple Int pins to support multiple peripherals or one pin and polling of the peripherals
  - Vectored interrupt
    - Peripheral must provide the address
    - Common when microprocessor has multiple peripherals connected by a system bus
  - Compromise: interrupt address table

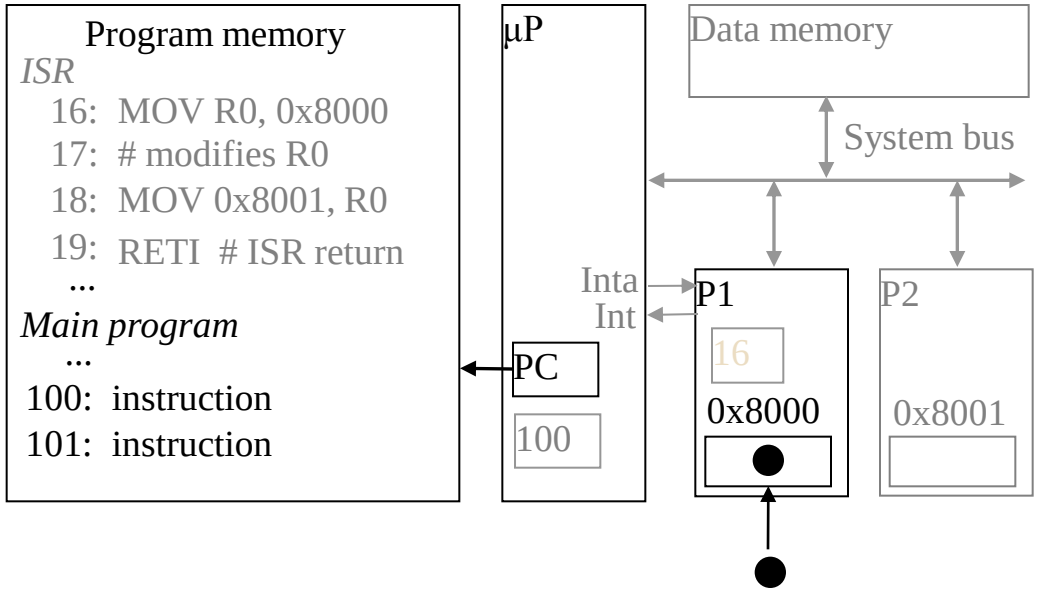
Interrupt-driven I/O using fixed ISR location



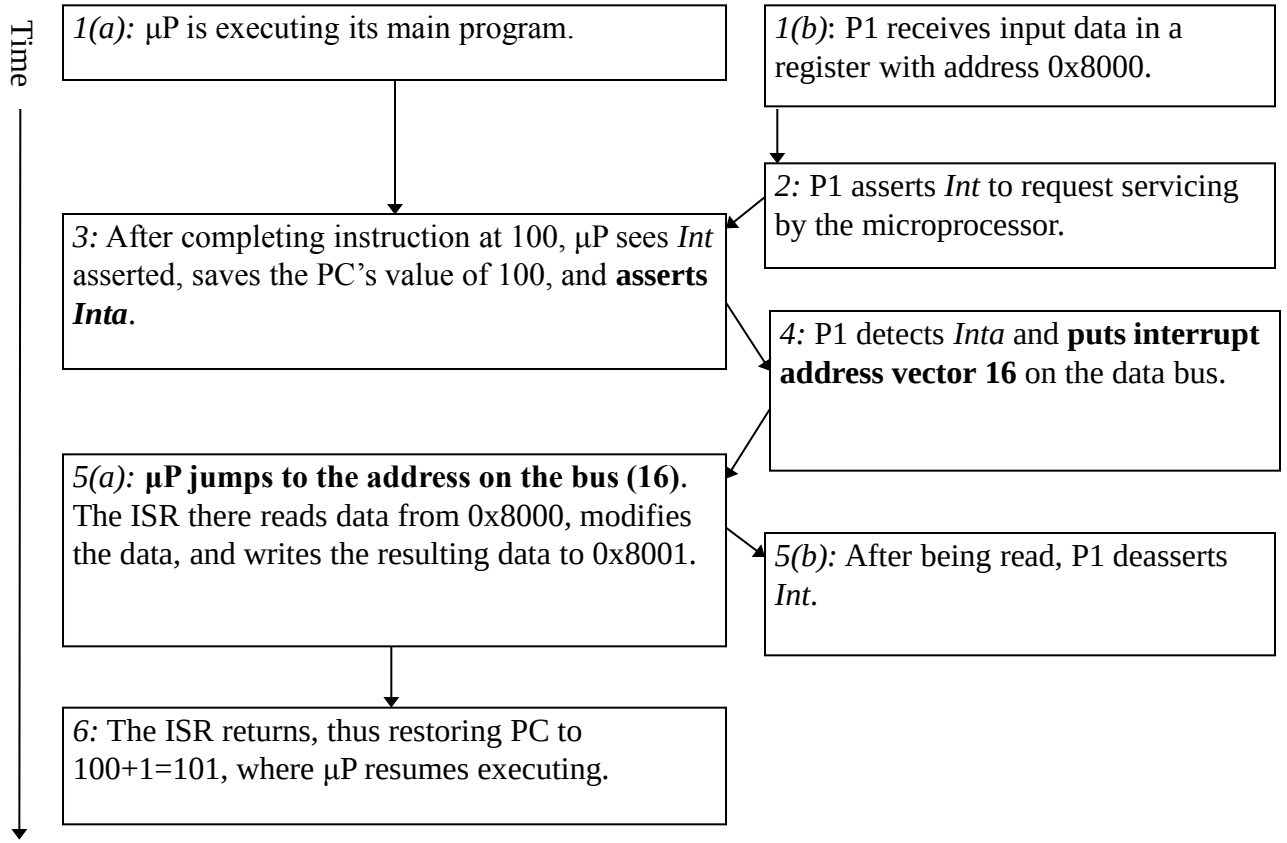
Interrupt-driven I/O using fixed ISR location



Interrupt-driven I/O using vectored interrupt



Interrupt-driven I/O using vectored interrupt

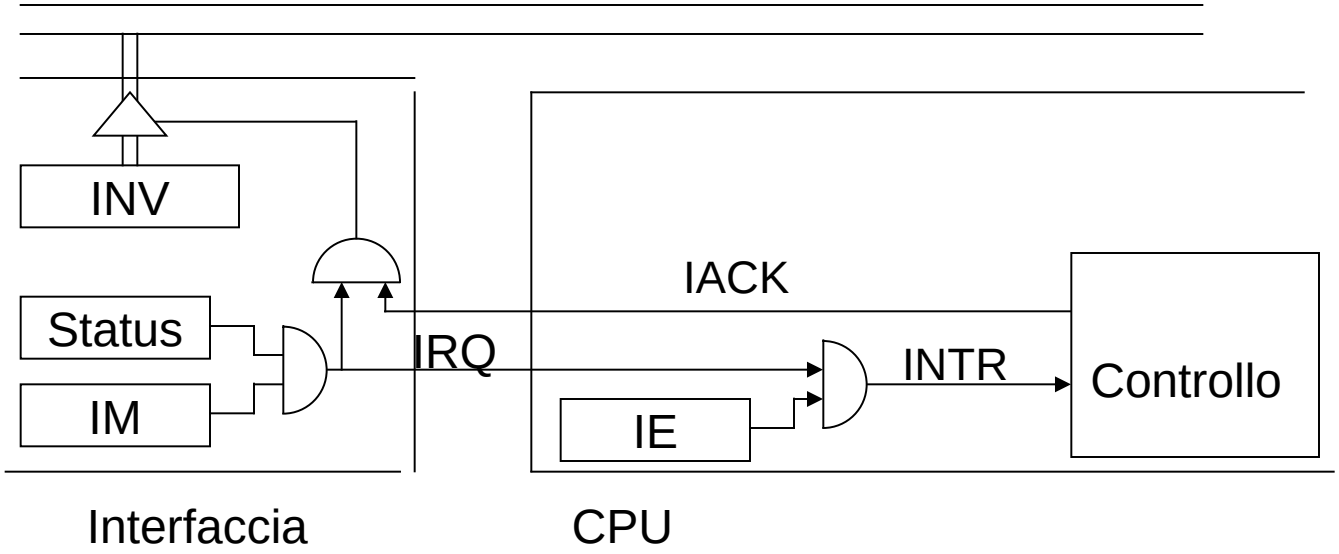


Interrupt address table

- 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

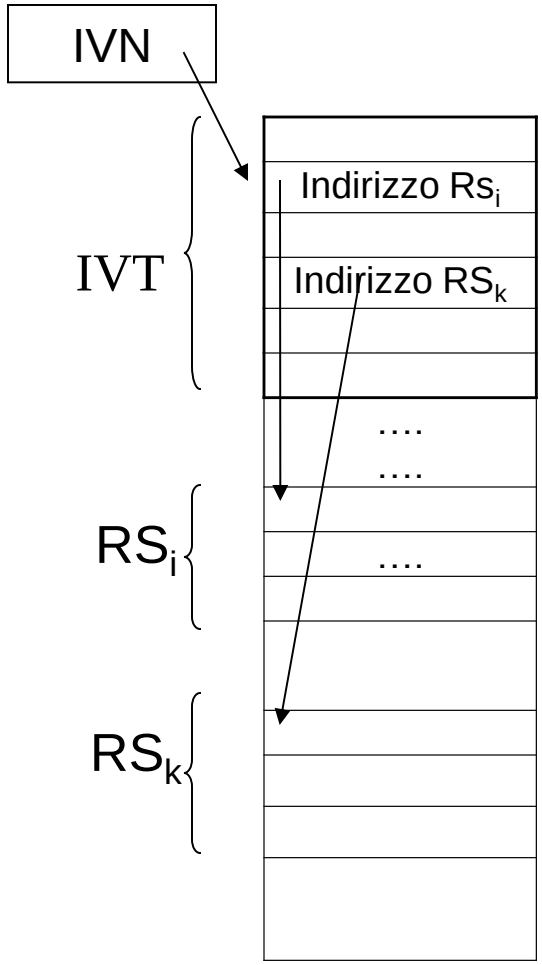
Riconoscimento mediante vettore delle interruzioni

- Nell'interfaccia è presente un registro in cui è memorizzato il codice identificativo del dispositivo (INV).
- In seguito alla richiesta di interruzione IRQ=1, se il processore è interrompibile (IE=1), attiva in risposta il segnale IACK.
- Il codice identificativo viene inviato sul bus dati in risposta al segnale IACK generato dal processore.



Riconoscimento mediante vettore delle interruzioni

- In memoria è presente una tabella, ***Interrupt Vector Table (IVT)***, che contiene gli indirizzi delle routine di servizio dei dispositivi.
- Il codice inviato dal dispositivo di I/O, ***Interrupt Vector Number (IVN)***, rappresenta l'indice della tabella corrispondente alla routine di servizio.
- La IVT di norma è memorizzata a partire dalle prime posizioni della memoria in modo da codificare l'IVN con pochi bit.
- Il riempimento della IVT (o di parte di essa) viene eseguita ad opera del programmatore



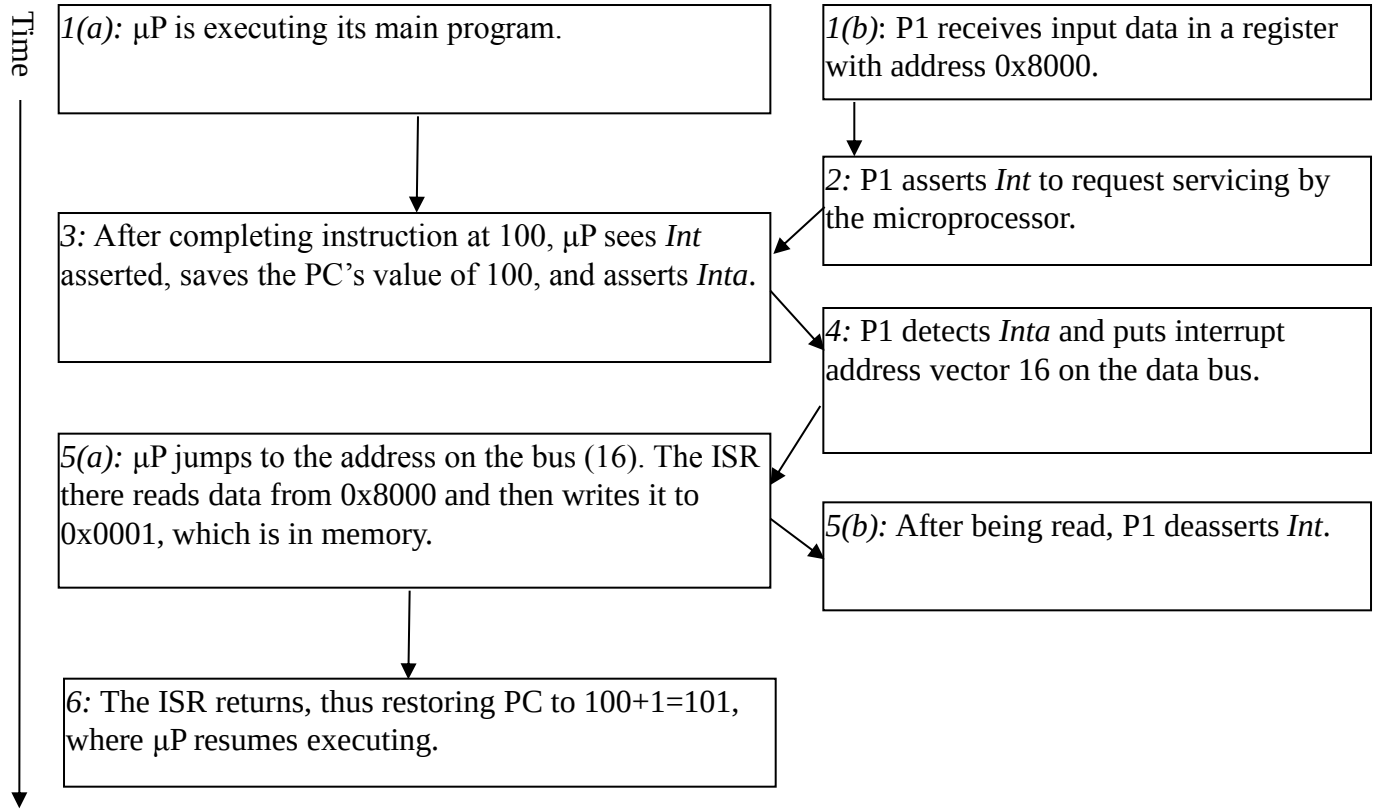
Additional interrupt issues

- 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
      - Harvard archictecture – processor can fetch and execute instructions as long as they don't access data memory – if they do, processor stalls

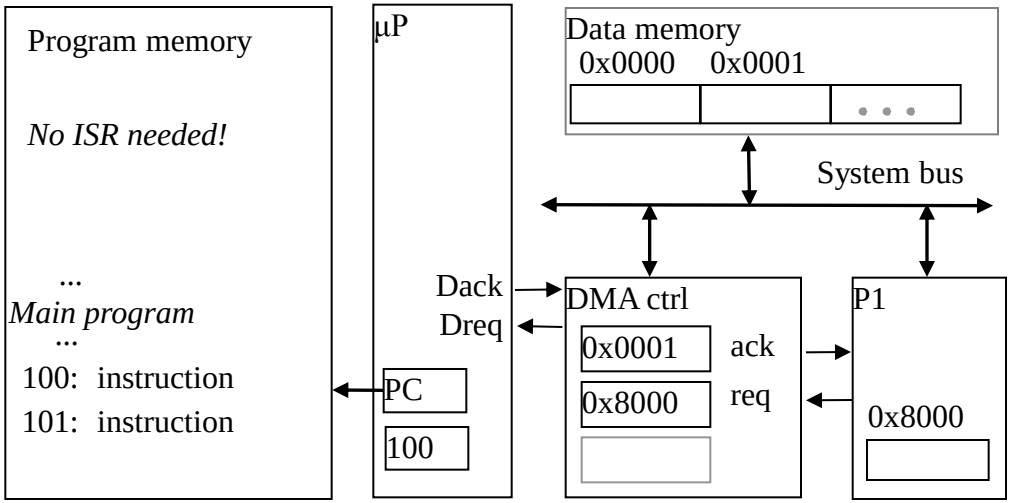
Peripheral to memory transfer *without* DMA, using vectored interrupt



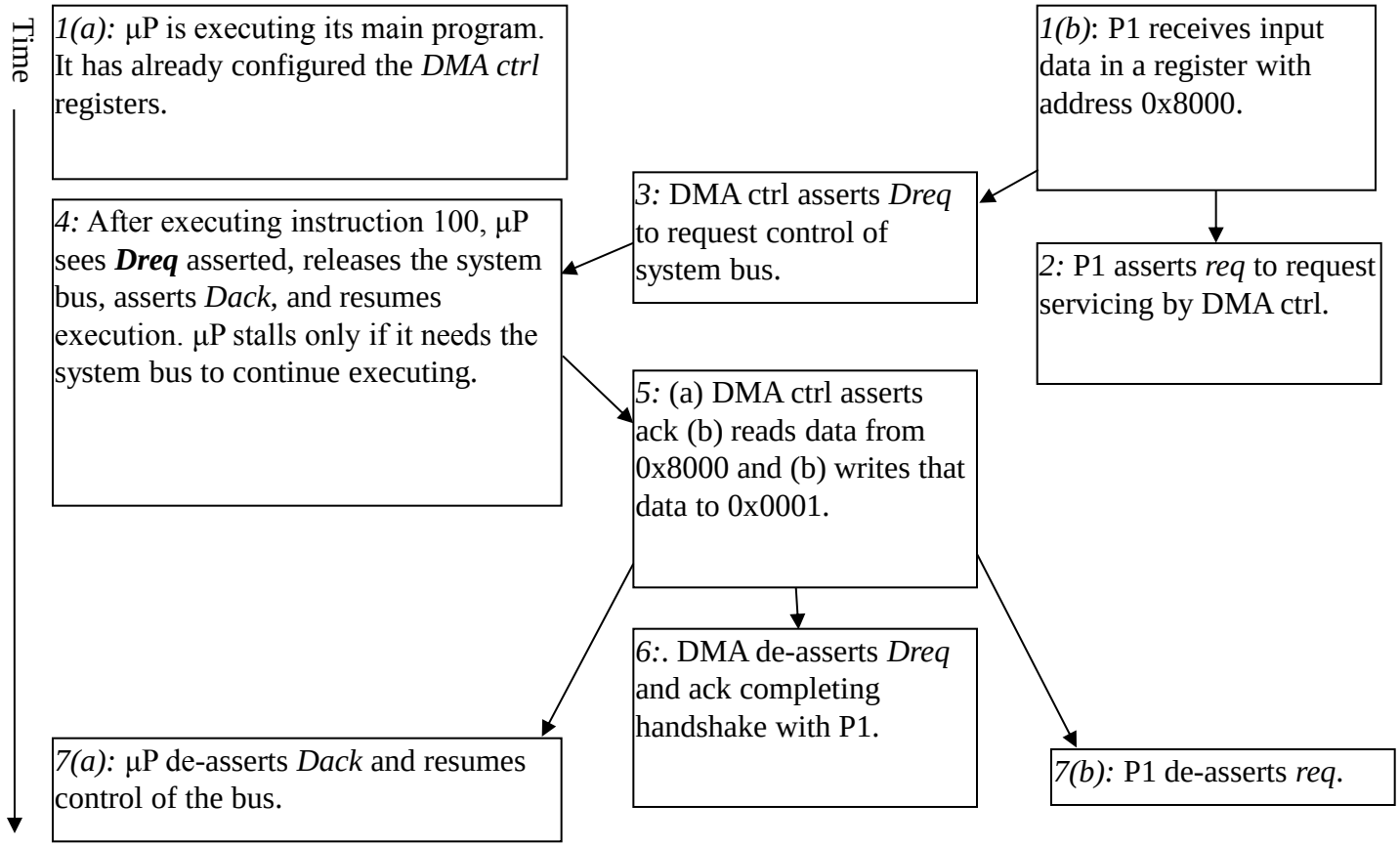
Peripheral to memory transfer with DMA (cont')

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.

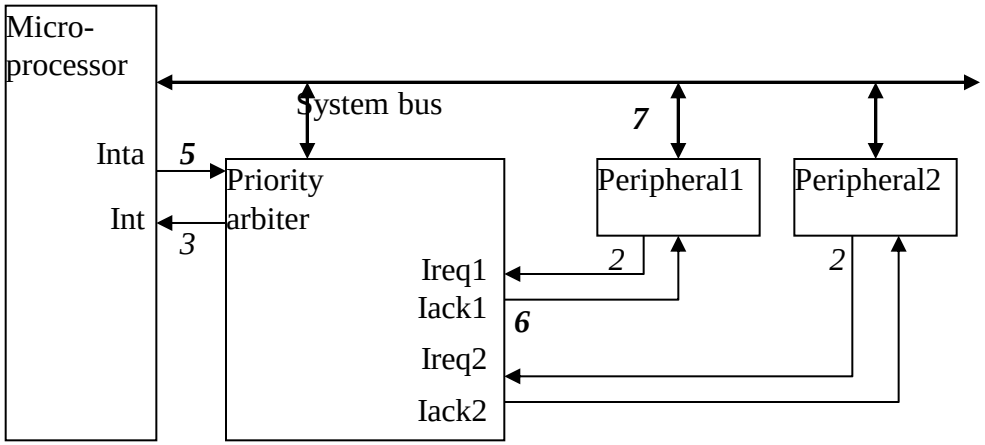


Peripheral to memory transfer with DMA

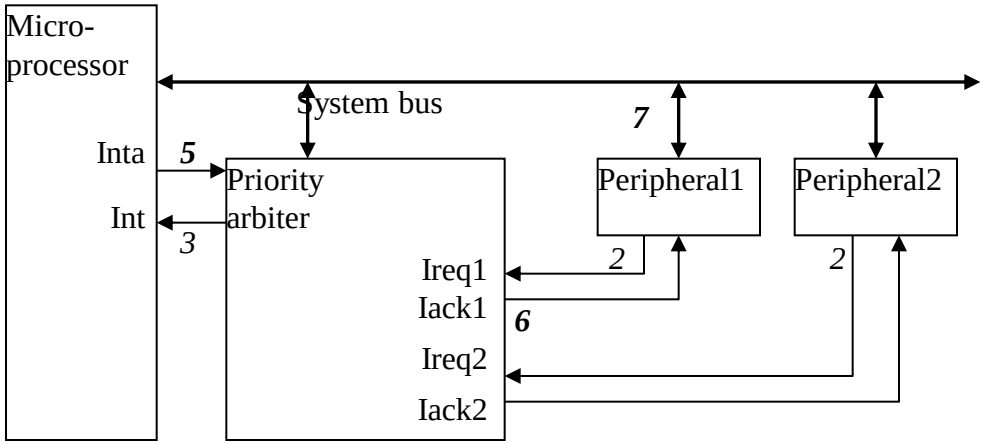


Arbitration: Priority arbiter

- 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 using a priority arbiter



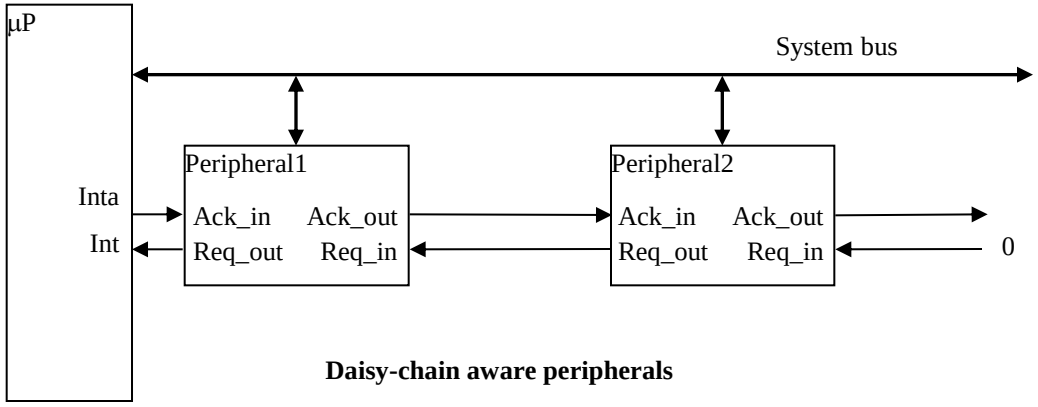
1. 1. Microprocessor is executing its program.
2. 2. Peripheral1 needs servicing so asserts *Ireq1*. Peripheral2 also needs servicing so asserts *Ireq2*.
3. 3. Priority arbiter sees at least one *Ireq* input asserted, so asserts *Int*.
4. 4. Microprocessor stops executing its program and stores its state.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

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

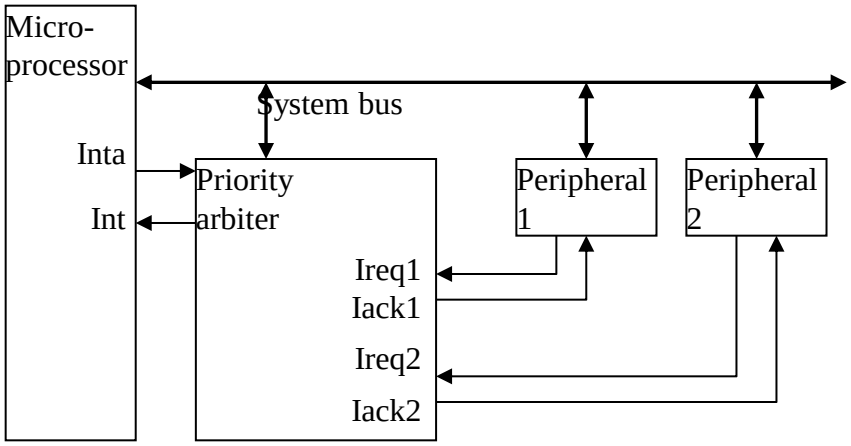
- Arbitration done by peripherals
  - Built into peripheral or external logic added
    - *req* input and *ack* output added to each peripheral
- Peripherals connected to each other in daisy-chain manner
  - One peripheral connected to resource, all others connected "upstream"
  - Peripheral's *req*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

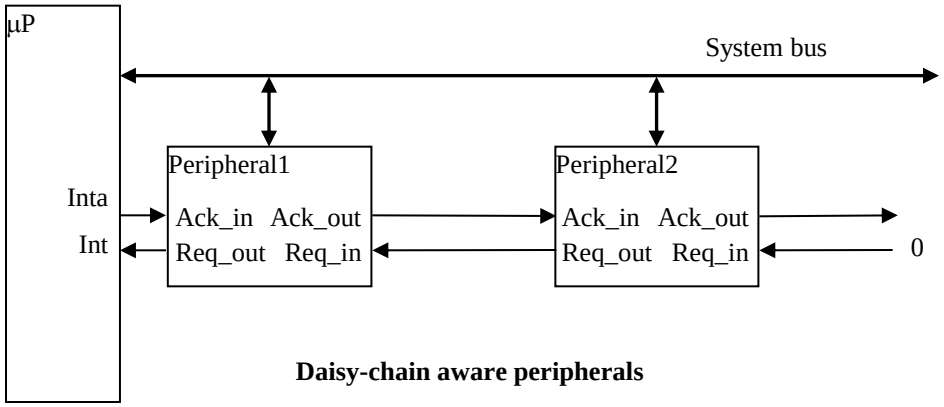


Daisy-chain aware peripherals

Arbitration: Daisy-chain arbitration

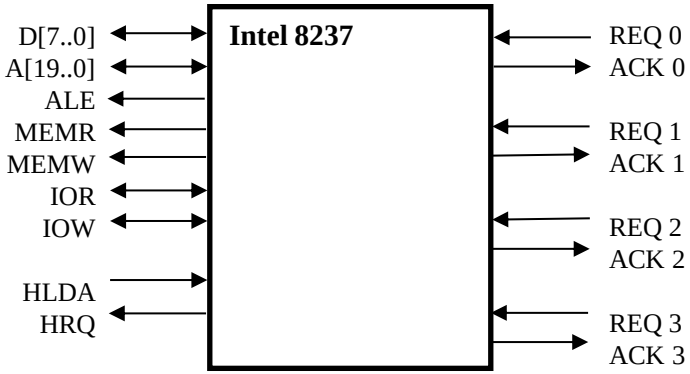
- Pros/cons
  - Easy to add/remove peripheral - no system redesign needed
  - Does not support rotating priority
  - One broken peripheral can cause loss of access to other peripherals





Daisy-chain aware peripherals

Intel 8237 DMA controller



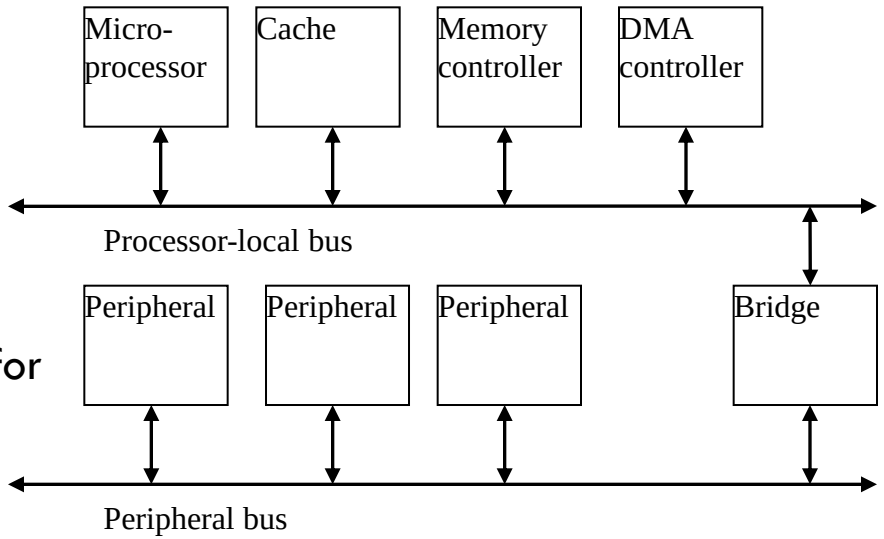
D[7..0]	These wires are connected to the system bus (ISA) and are used by the microprocessor to write to the internal registers of the 8237.
A[19..0]	These wires are connected to the system bus (ISA) and are used by the DMA to issue the memory location where the transferred data is to be written to. The 8237 is also addressed by the micro-processor through the lower bits of these address lines.
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.	

Multilevel bus architectures

- A microprocessor-based embedded system can have numerous types of communications
- High speed communications between processor and memory
- Less frequent communications between processors and its peripherals can require less speed
- 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

Multilevel bus architectures

- 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

- 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
- 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

Parallel communication

- Multiple data, control, and possibly power wires
  - One bit per wire
- High data throughput with short distances
- Typically used when connecting devices on same IC or same circuit board
  - Bus must be kept short
    - long parallel wires result in high capacitance values which requires more time to charge/discharge
    - Data misalignment between wires increases as length increases
- Higher cost, bulky

Serial communication

- Single data wire, possibly also control and power wires
- Words transmitted one bit at a time
- Higher data throughput with long distances
  - Less average capacitance, so more bits per unit of time
- Cheaper, less bulky
- More complex interfacing logic and communication protocol
  - Sender needs to decompose word into bits
  - Receiver needs to recompose bits into word
  - Control signals often sent on same wire as data increasing protocol complexity

Wireless communication

- Infrared (IR)
  - Electronic wave frequencies just below visible light spectrum
  - Diode emits infrared light to generate signal
  - Infrared transistor detects signal, conducts when exposed to infrared light
  - Cheap to build
  - Need line of sight, limited range
- Radio frequency (RF)
  - Electromagnetic wave frequencies in radio spectrum
  - Analog circuitry and antenna needed on both sides of transmission
  - Line of sight not needed, transmitter power determines range

Error detection and correction

- Often part of bus protocol
- Error detection: ability of receiver to detect errors during transmission
- Error correction: ability of receiver and transmitter to cooperate to correct problem
  - Typically done by acknowledgement/retransmission protocol
- Bit error: single bit is inverted
- Burst of bit error: consecutive bits received incorrectly
- 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

Serial protocols: CAN

- CAN (Controller area network)
  - Protocol for real-time applications
  - Developed by Robert Bosch GmbH
  - Originally for communication among components of cars
  - Applications now using CAN include:
    - elevator controllers, copiers, telescopes, production-line control systems, and medical instruments
  - Data transfer rates up to 1 Mbit/s and 11-bit addressing
  - Common devices interfacing with CAN:
    - 8051-compatible 8592 processor and standalone CAN controllers
  - Actual physical design of CAN bus not specified in protocol
    - Requires devices to transmit/detect dominant and recessive signals to/from bus
    - e.g., '1' = dominant, '0' = recessive if single data wire used
    - Bus guarantees dominant signal prevails over recessive signal if asserted simultaneously

Serial protocols: FireWire

- FireWire (a.k.a. I-Link, Lynx, IEEE 1394)
  - High-performance serial bus developed by Apple Computer Inc.
  - Designed for interfacing independent electronic components
    - e.g., Desktop, scanner
  - Data transfer rates from 12.5 to 400 Mbits/s, 64-bit addressing
  - Plug-and-play capabilities
  - Packet-based layered design structure
  - Applications using FireWire include:
    - disk drives, printers, scanners, cameras
  - Capable of supporting a LAN similar to Ethernet
    - 64-bit address:
      - 10 bits for network ids, 1023 subnetworks
      - 6 bits for node ids, each subnetwork can have 63 nodes
      - 48 bits for memory address, each node can have 281 terabytes of distinct locations

Serial protocols: USB

- USB (Universal Serial Bus)
  - Easier connection between PC and monitors, printers, digital speakers, modems, scanners, digital cameras, joysticks, multimedia game equipment
  - 2 data rates:
    - 12 Mbps for increased bandwidth devices
    - 1.5 Mbps for lower-speed devices (joysticks, game pads)
  - Tiered star topology can be used
    - One USB device (hub) connected to PC
      - hub can be embedded in devices like monitor, printer, or keyboard or can be standalone
    - Multiple USB devices can be connected to hub
    - Up to 127 devices can be connected like this
  - USB host controller
    - Manages and controls bandwidth and driver software required by each peripheral
    - Dynamically allocates power downstream according to devices connected/disconnected

Parallel protocols: PCI Bus

- PCI Bus (Peripheral Component Interconnect)
  - High performance bus originated at Intel in the early 1990's
  - Standard adopted by industry and administered by PCISIG (PCI Special Interest Group)
  - Interconnects chips, expansion boards, processor memory subsystems
  - Data transfer rates of 127.2 to 508.6 Mbits/s and 32-bit addressing
    - Later extended to 64-bit while maintaining compatibility with 32-bit schemes
  - Synchronous bus architecture
  - Multiplexed data/address lines

Parallel protocols: ARM Bus

- ARM Bus
  - Designed and used internally by ARM Corporation
  - Interfaces with ARM line of processors
  - Many IC design companies have own bus protocol
  - Data transfer rate is a function of clock speed
    - If clock speed of bus is X, transfer rate = 16 x X bits/s
  - 32-bit addressing