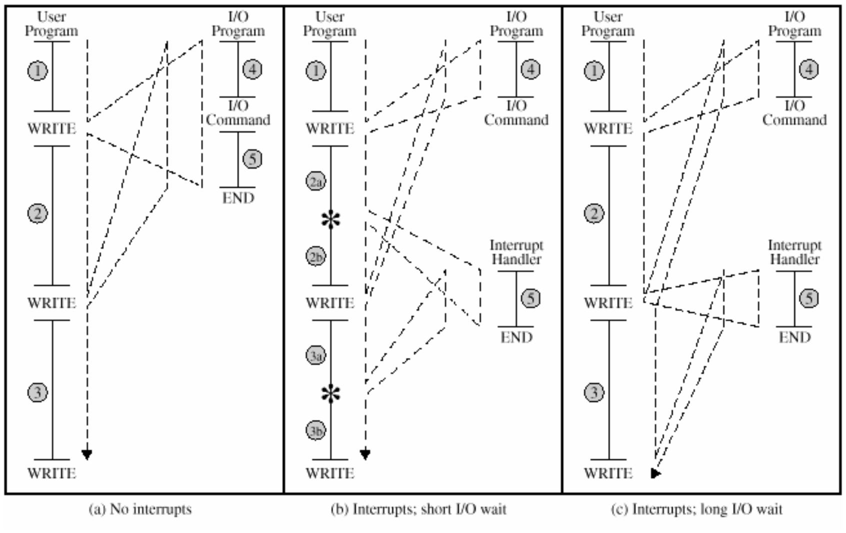
Interrupts, Buses

Chapter 6.2.5, 8.2-8.3

Introduction to Interrupts

- Interrupts are a mechanism by which other modules (e.g. I/O) may interrupt normal sequence of processing
- Four general classes of interrupts
  - Terms: **Traps** when initiated by system, **Interrupt** when initiated by programmer
  - Program - e.g. overflow, division by zero
  - Timer, internal timer, used in pre-emptive multi-tasking
  - I/O - from I/O controller
  - Hardware failure, e.g. memory parity error
- Particularly useful when one module is much slower than another, e.g. disk access (milliseconds) vs. CPU (microseconds or faster)

Interrupt Examples



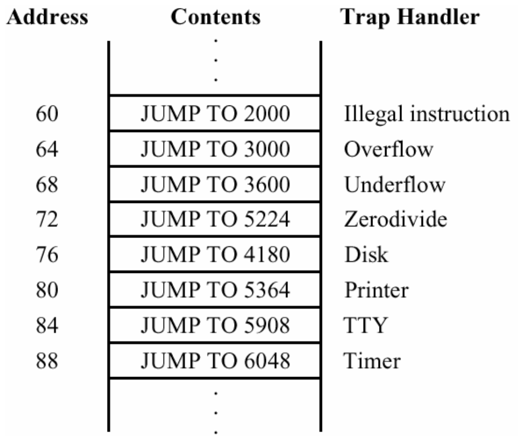
4 = code for pre-write, 5 = code for post-write, * = interrupt

Interrupt Cycle

- Added to instruction cycle
- Processor checks for interrupt
  - Indicated by an interrupt signal
- If no interrupt, fetch next instruction
- If interrupt pending:
  - Suspend execution of current program
  - Save context (what does this mean?)
  - Set PC to start address of interrupt handler routine
  - Process interrupt, i.e. execute interrupt handler code
  - Restore context and continue interrupted program

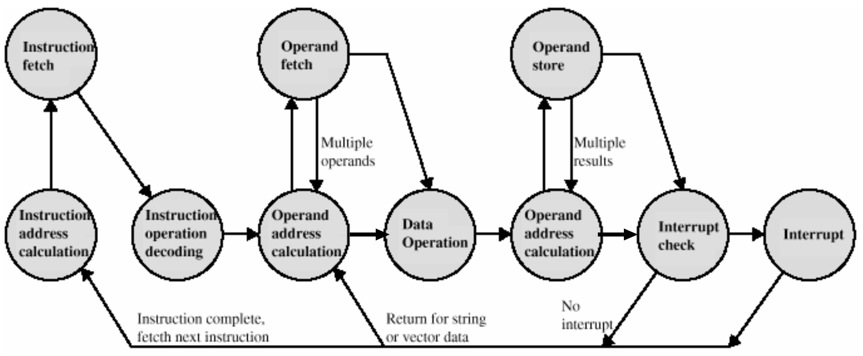
Branch Table for Handlers

How do we know where to go to execute the interrupt handler?
Lookup in a branch table, also called the interrupt vector



Address	Contents	Trap Handler
	. . .	
60	JUMP TO 2000	Illegal instruction
64	JUMP TO 3000	Overflow
68	JUMP TO 3600	Underflow
72	JUMP TO 5224	Zerodivide
76	JUMP TO 4180	Disk
80	JUMP TO 5364	Printer
84	JUMP TO 5908	TTY
88	JUMP TO 6048	Timer
	. . .	

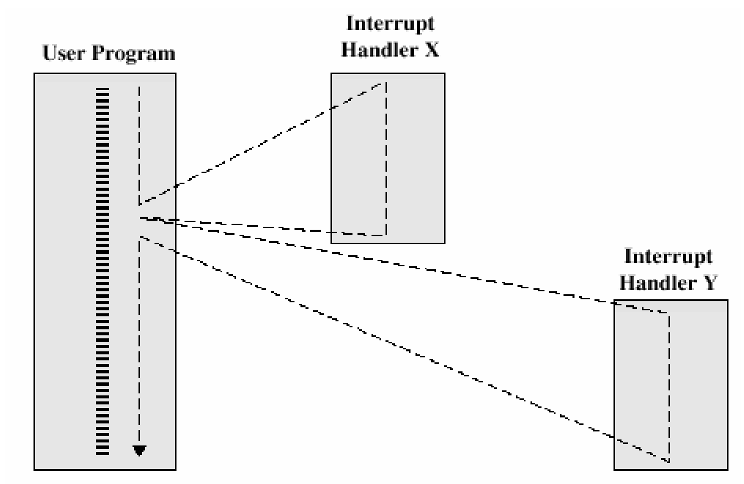
Instruction Cycle (with Interrupts) - State Diagram


Instruction fetch
Operand fetch
Operand store
Multiple operands
Multiple results
Instruction address calculation
Instruction operation decoding
Operand address calculation
Data Operation
Operand address calculation
Interrupt check
Interrupt
No interrupt
Instruction complete, fetcth next instruction
Return for string or vector data

Multiple Interrupts

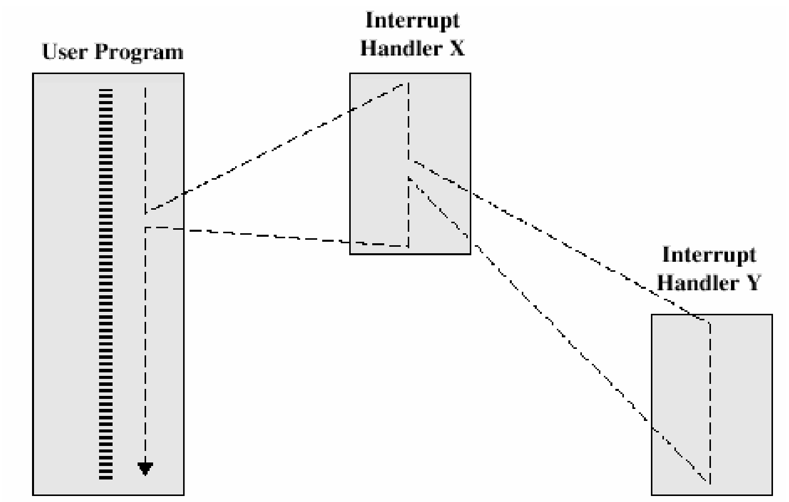
- Disable interrupts – Sequential Processing
  - Processor will ignore further interrupts whilst processing one interrupt
  - Interrupts remain pending and are checked after first interrupt has been processed
  - Interrupts handled in sequence as they occur
- Define priorities – Nested Processing
  - Low priority interrupts can be interrupted by higher priority interrupts
  - When higher priority interrupt has been processed, processor returns to previous interrupt

Multiple Interrupts - Sequential



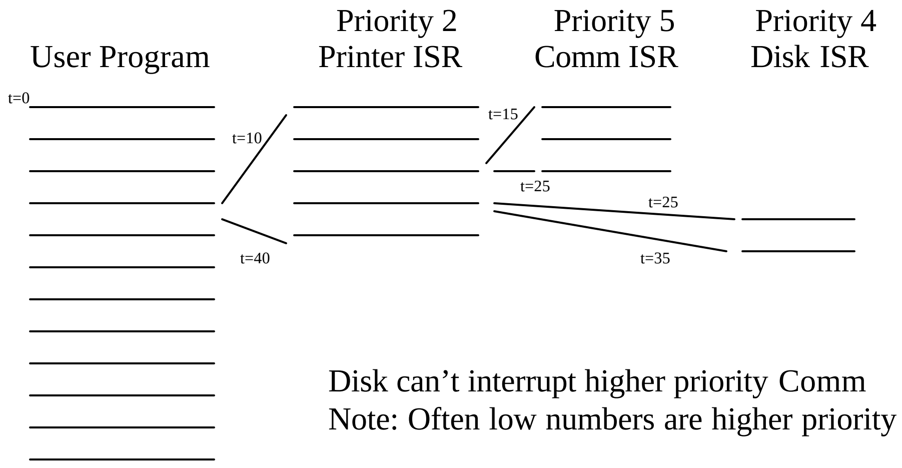
Disabled Interrupts – Nice and Simple

Multiple Interrupts - Nested



How to handle state with an arbitrary number of interrupts?

Sample Time Sequence of Multiple Interrupts



Disk can’t interrupt higher priority Comm
Note: Often low numbers are higher priority

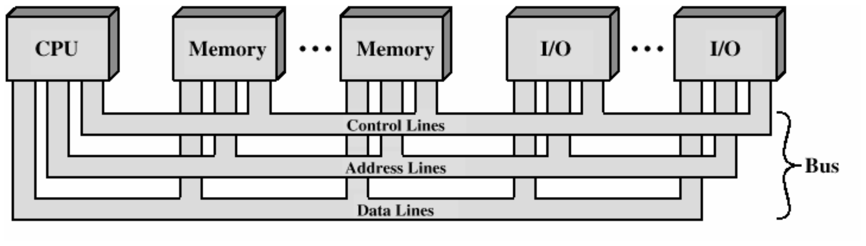
Buses

- There are a number of possible interconnection systems. The most common structure is the **bus**
- Single and multiple BUS structures are most common
- e.g. Control/Address/Data bus (PC)

What is a Bus?

- A communication pathway connecting two or more devices
- Usually broadcast
  - Everyone listens, must share the medium
  - Master – can read/write exclusively, only one master
  - Slave – everyone else. Can monitor data but not produce
- Often grouped
  - A number of channels in one bus
  - e.g. 32 bit data bus is 32 separate single bit channels
- Power lines may not be shown
- Three major buses: data, address, control

Bus Interconnection Scheme



Data Bus

- Carries data
  - Remember that there is no difference between "data" and "instruction" at this level
- Width is a key determinant of performance
  - 8, 16, 32, 64, 128 bit
  - Generally we like the data bus width to match the register word size
    - Famous non-examples include 8088, 386SX

Address bus

- Identify the source or destination of data
  - In general, the address specifies a specific memory address or a specific I/O port
- e.g. CPU needs to read an instruction (data) from a given location in memory
- Bus width determines maximum memory capacity of system
  - 8086 has 20 bit address bus but 16 bit word size for 64k directly addressable address space
  - But it could address up to 1MB using a segmented memory model

Control Bus

- Control and timing information
  - Determines what modules can use the data and address lines
  - If a module wants to send data, it must (1) obtain permission to use the bus, and (2) transfer data – which might be a request for another module to send data
    - We will skip how arbitration for control is performed
- Typical control lines

– Memory read	- Memory write
– I/O read	- I/O write
– Interrupt request	- Interrupt ACK
– Bus Request	- Bus Grant
– Clock signals	

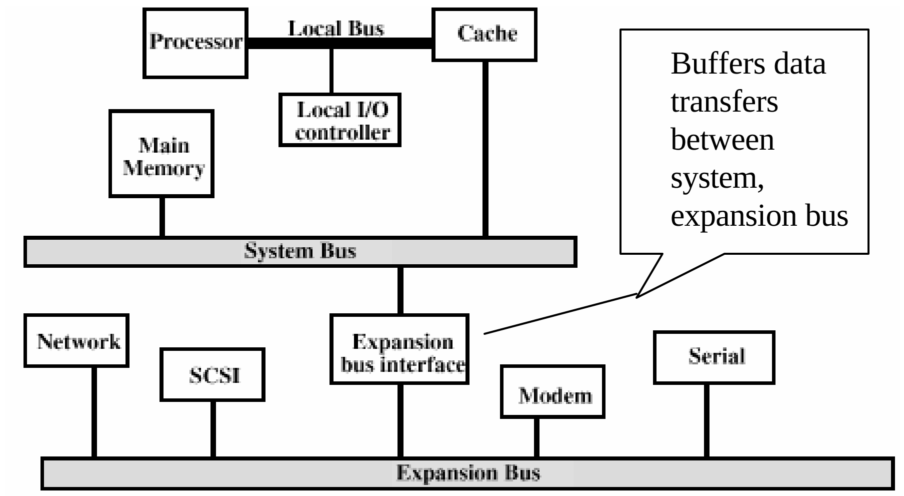
Big and Yellow?

- What do buses look like?
  - Parallel lines on circuit boards
  - Ribbon cables
  - Strip connectors on mother boards
    - e.g. PCI
  - Sets of wires
- Limited by physical proximity – time delays, fan out, attenuation are all factors for long buses

Single Bus Problems

- Lots of devices on one bus leads to:
  - Propagation delays
    - Long data paths mean that co-ordination of bus use can adversely affect performance – **bus skew,** data arrives at slightly different times
    - If aggregate data transfer approaches bus capacity. Could increase bus width, but expensive
  - Device speed
    - Bus can't transmit data faster than the slowest device
    - Slowest device may determine bus speed!
      - Consider a high-speed network module and a slow serial port on the same bus; must run at slow serial port speed so it can process data directed for it
  - Power problems
- Most systems use multiple buses to overcome these problems

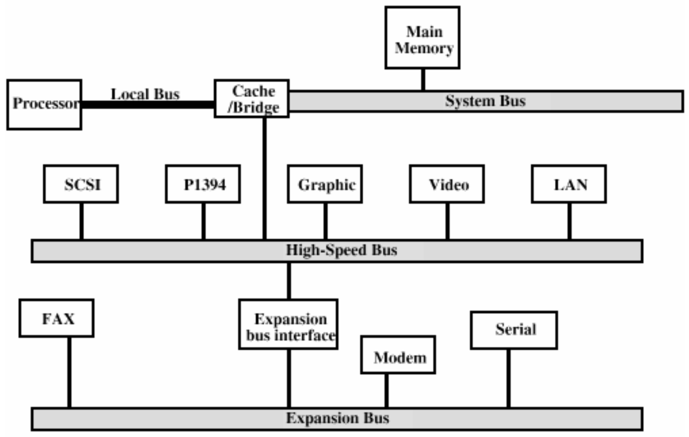
Traditional (ISA) with cache



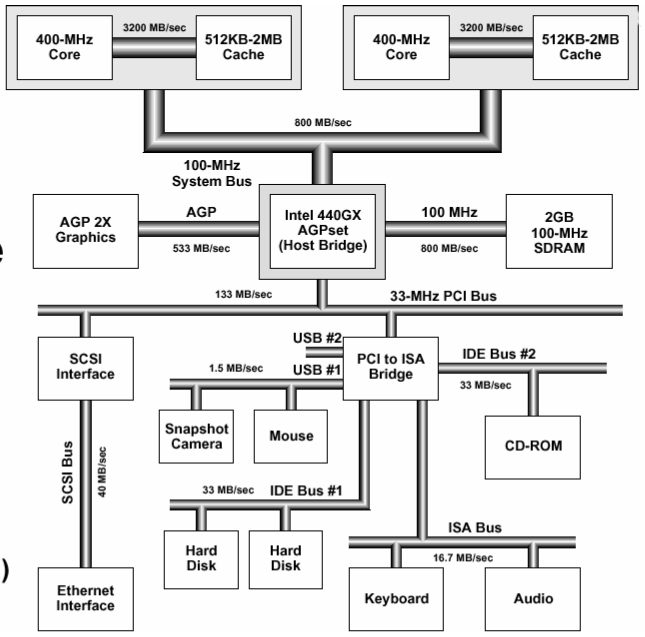
This approach breaks down as I/O devices need higher performance

High Performance Bus – Mezzanine Architecture

Addresses higher speed I/O devices by moving up in the hierarchy



Bridge Based Bus Architecture



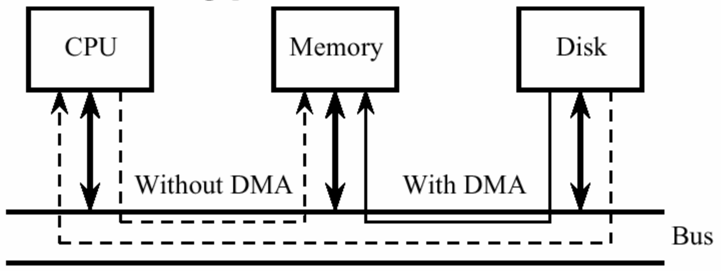
- Bridging with dual Pentium II Xeon processors on Slot 2.

(Source: http://www.intel.com.)

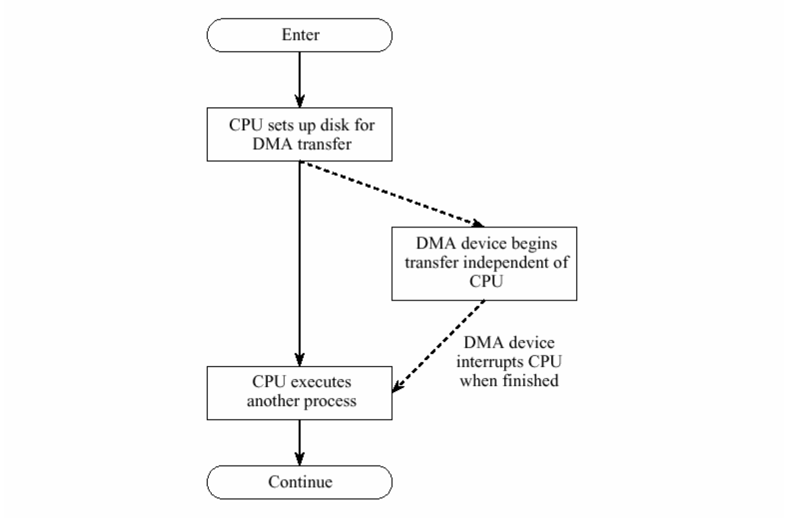
Direct Memory Access

Tying together buses with interrupts!

DMA Transfer from Disk to Memory Bypasses the CPU



DMA Flowchart for a Disk Transfer


Enter
CPU sets up disk for DMA transfer
DMA device begins transfer independent of CPU
DMA device interrupts CPU when finished
CPU executes another process
Continue