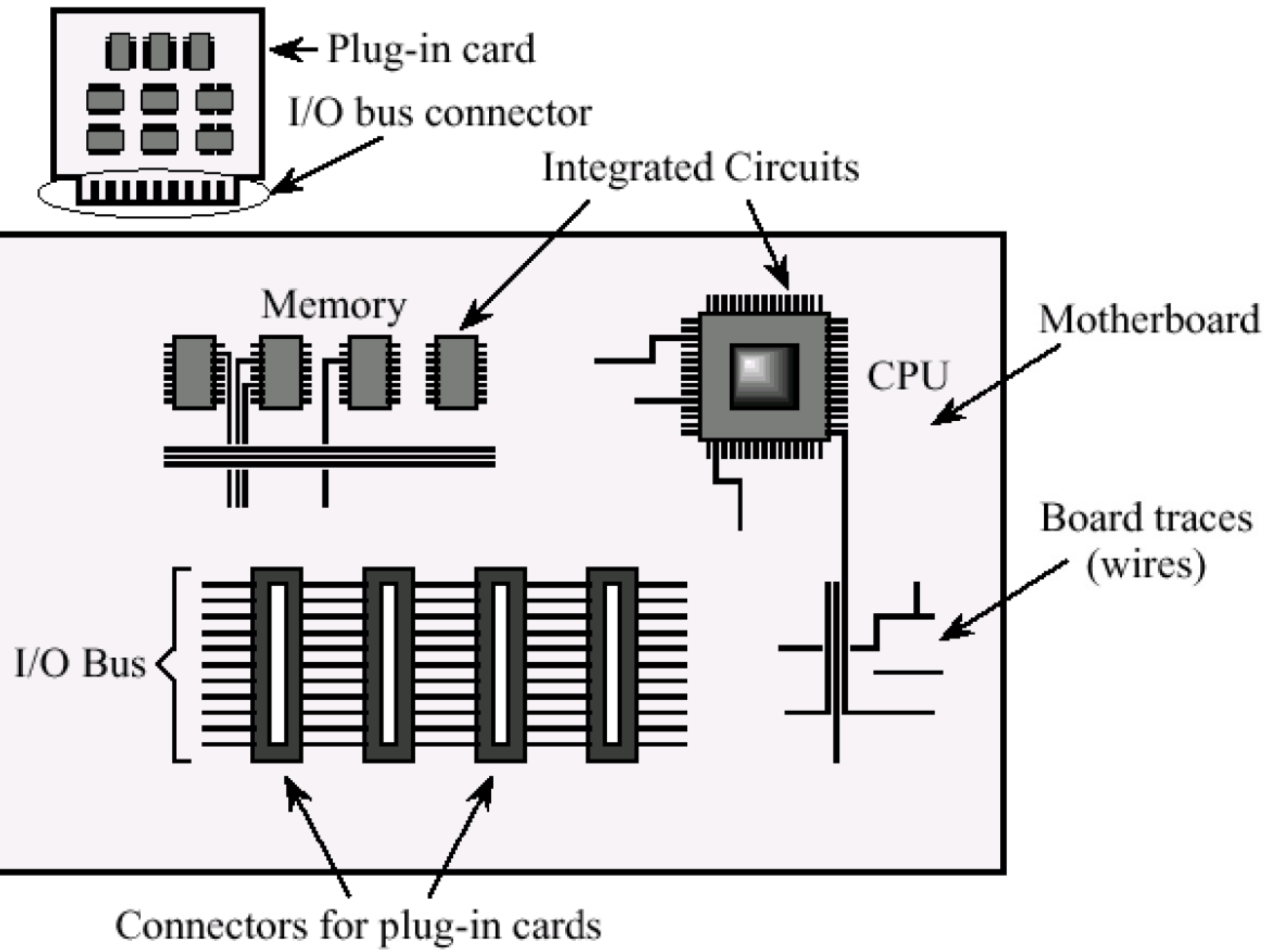
William Stallings
Computer Organization and Architecture

Chapter 3
Instruction Cycle Review
System Buses

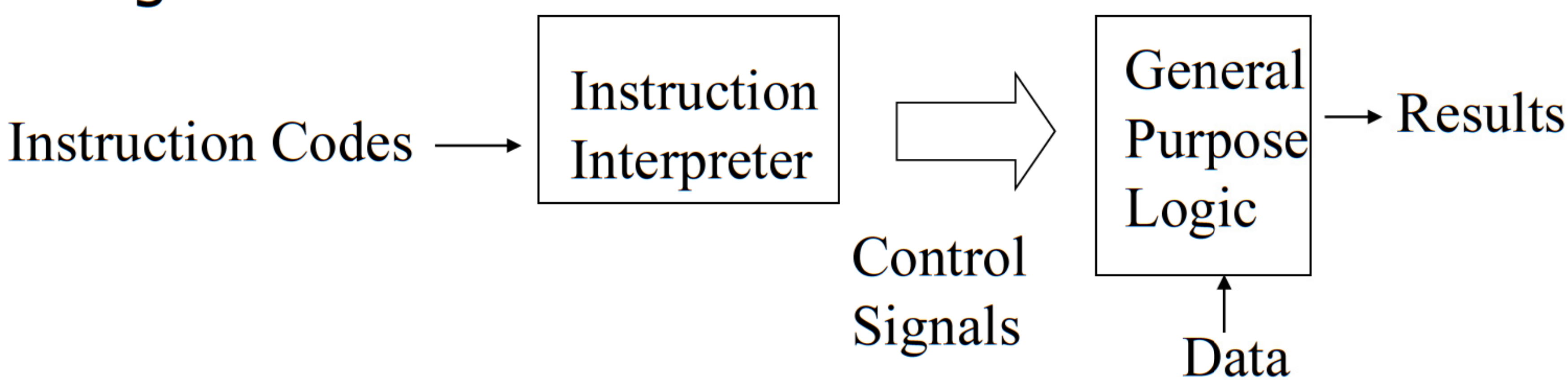
Simple Bus Architecture

- **A simplified motherboard of a personal computer (top view):**



Architecture Review - Program Concept

- Hardwired systems are inflexible
  - Lots of work to re-wire, or re-toggle
- General purpose hardware can do different tasks, given correct control signals
- Instead of re-wiring, supply a new set of control signals



What is a program?

- Software
  - A sequence of steps
  - For each step, an arithmetic or logical operation is done
  - For each operation, a different set of control signals is needed – i.e. an instruction

Function of Control Unit

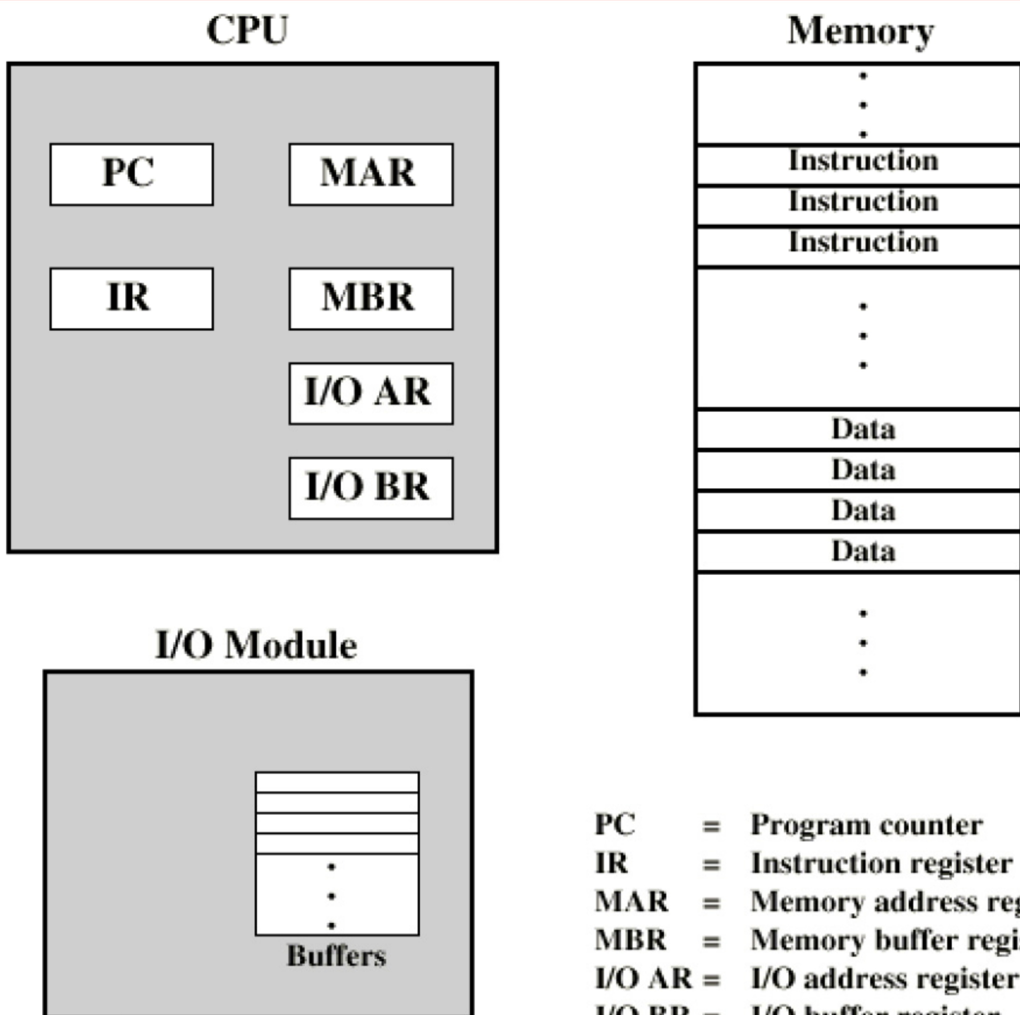
- For each operation a unique code is provided
  - e.g. ADD, MOVE
- A hardware segment accepts the code and issues the control signals

- We have a computer!

Components

- Central Processing Unit
  - Control Unit
  - Arithmetic and Logic Unit
- Data and instructions need to get into the CPU and results out
  - Input/Output
- Temporary storage of code and results is needed
  - Main memory

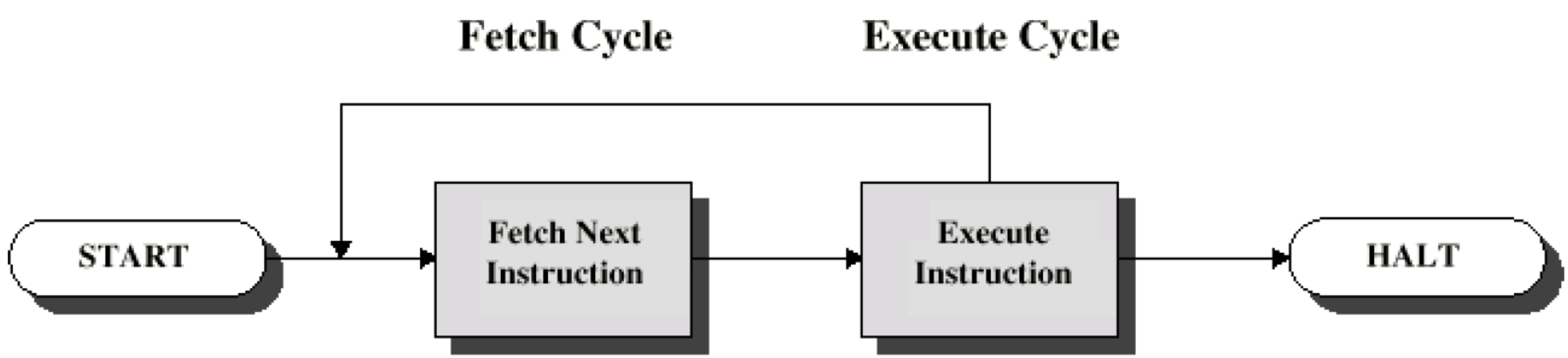
Computer Components: Top Level View



PC = Program counter
IR = Instruction register
MAR = Memory address register
MBR = Memory buffer register
I/O AR = I/O address register
I/O BR = I/O buffer register

Simplified Instruction Cycle

- Two steps:
  - Fetch
  - Execute



Fetch Cycle

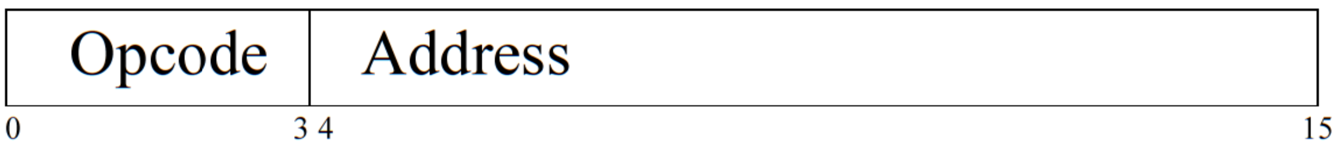
- Program Counter (PC) holds address of next instruction to fetch
- Processor fetches instruction from memory location pointed to by PC
- Increment PC
  - Unless told otherwise
- Instruction loaded into Instruction Register (IR)
- Processor interprets instruction and performs required actions

Execute Cycle

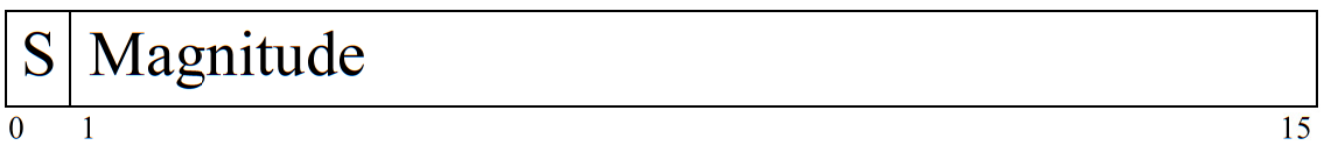
- Processor-memory
  - data transfer between CPU and main memory
- Processor I/O
  - Data transfer between CPU and I/O module
- Data processing
  - Some arithmetic or logical operation on data
- Control
  - Alteration of sequence of operations
  - e.g. jump
- Combination of above

Hypothetical Machine

- Instruction Format - Address range?

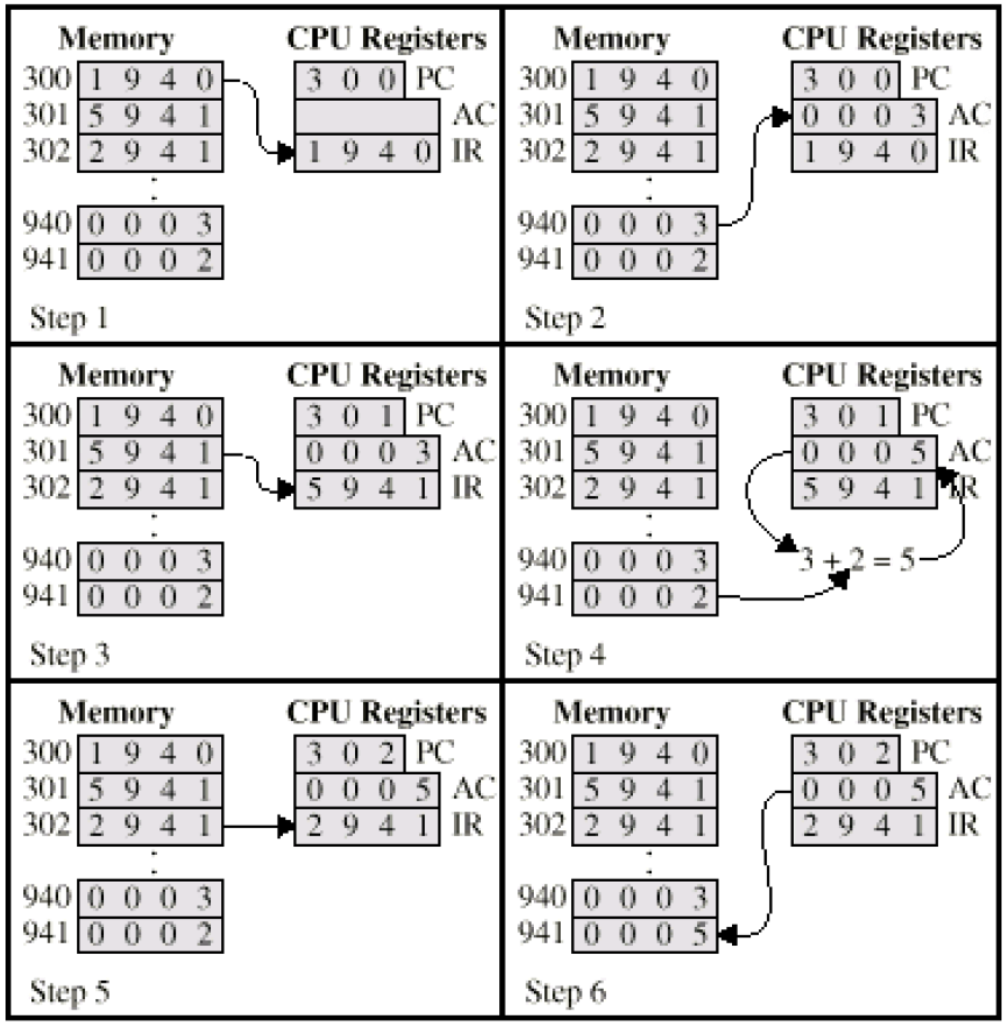


- Integer Format - Data range?



- Registers
  - PC = Program Counter, IR = Instruction Register, AC = Accumulator
- Partial List of Opcodes
  - 0001 = Load AC from Memory
  - 0010 = Store AC to Memory
  - 0101 = Add to AC from Memory

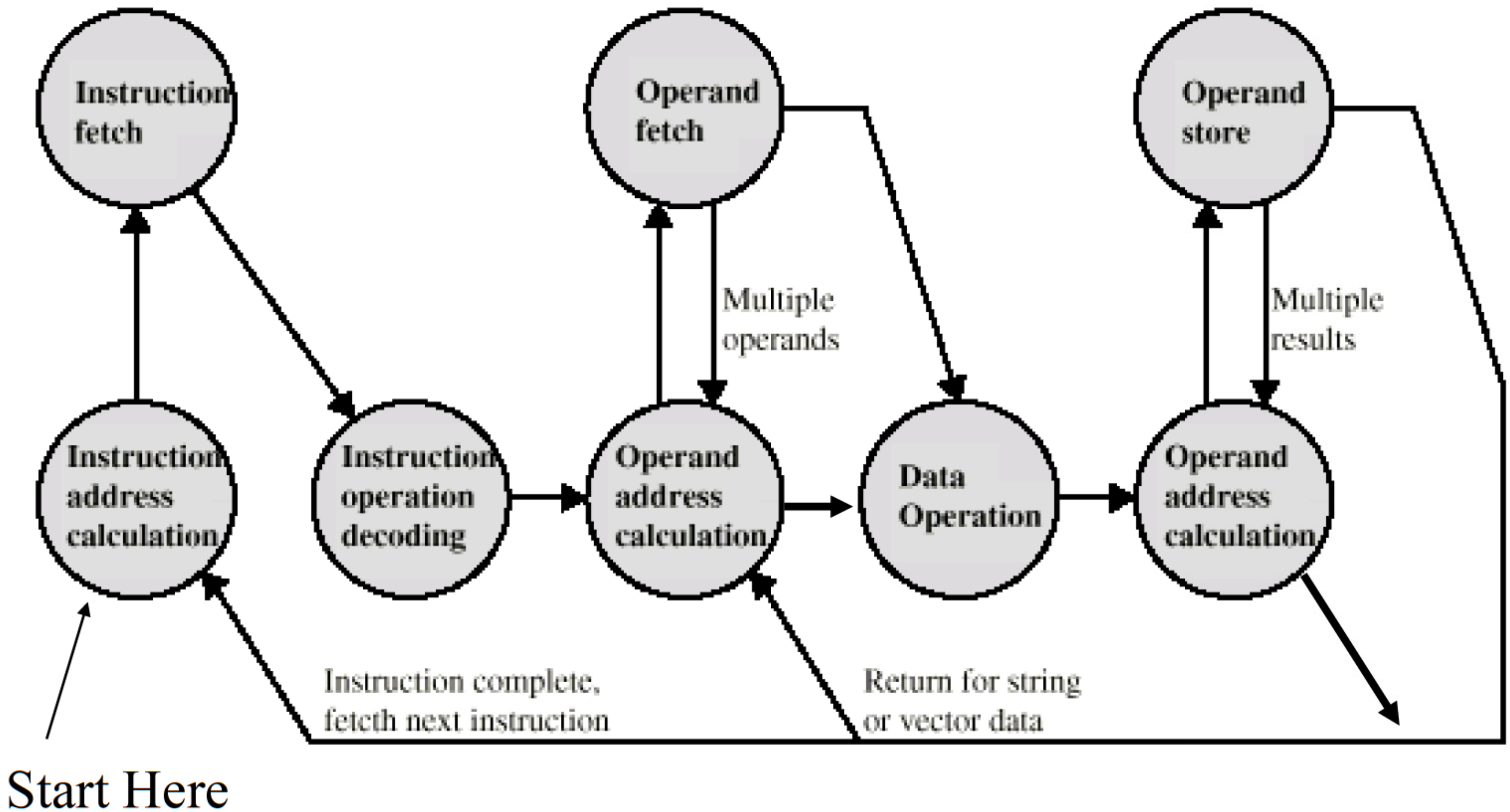
Example of Program Execution



Modifications to Instruction Cycle

- Simple Example
  - Always added one to PC
  - Entire operand fetched with instruction
- More complex examples
  - Might need more complex instruction address calculation
    - Consider a 64 bit processor, variable length instructions
  - Instruction set design might require repeat trip to memory to fetch operand
    - In particular, if memory address range exceeds word size
  - Operand store might require many trips to memory
    - Vector calculation

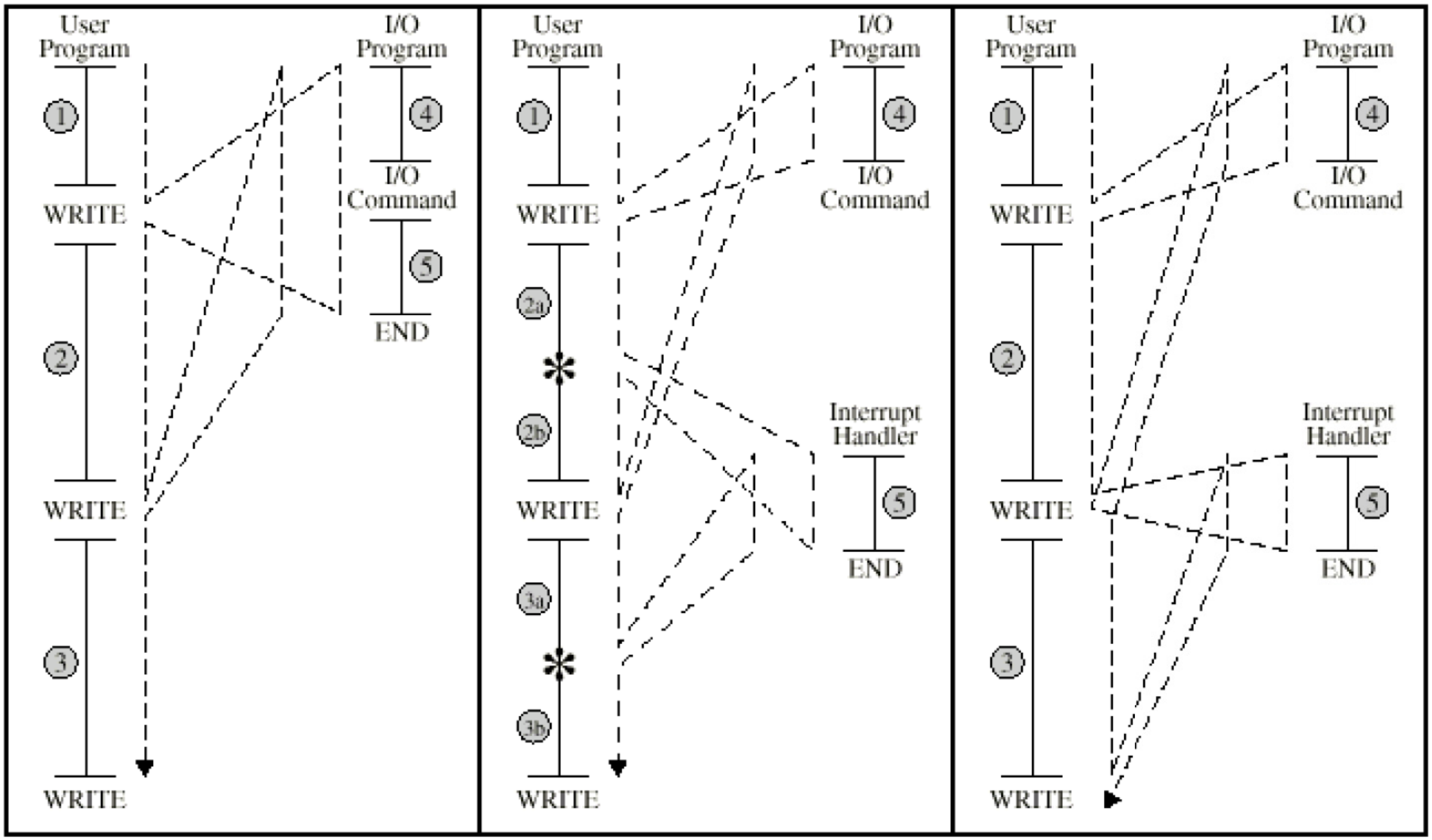
Instruction Cycle - State Diagram



Introduction to Interrupts

- We will have more to say about interrupts later!
- Interrupts are a mechanism by which other modules (e.g. I/O) may interrupt normal sequence of processing
- Four general classes of interrupts
  - Program - e.g. overflow, division by zero
  - Timer
    - Generated by internal processor 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

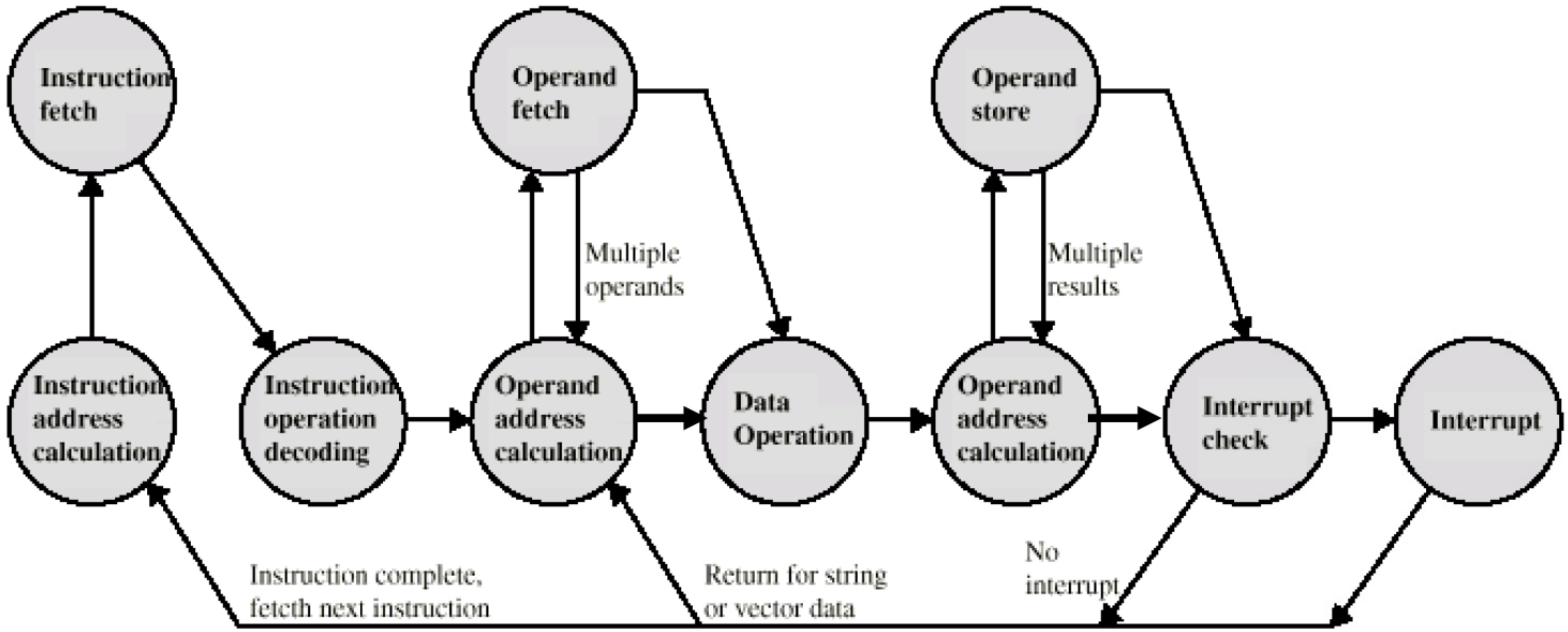


(a) No interrupts (b) Interrupts; short I/O wait (c) Interrupts; long I/O wait

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
  - Restore context and continue interrupted program

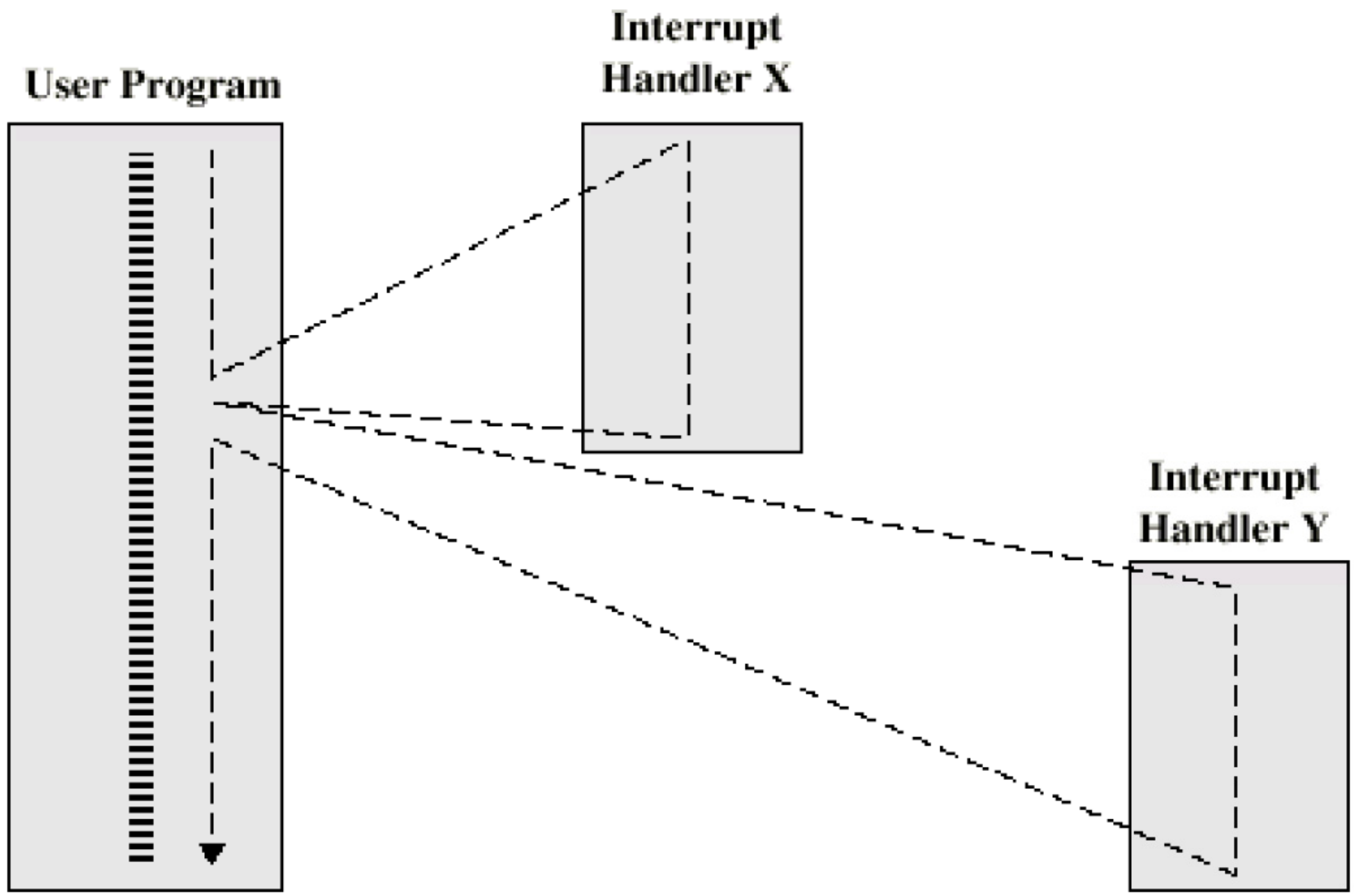
Instruction Cycle (with Interrupts) - State Diagram



Multiple Interrupts

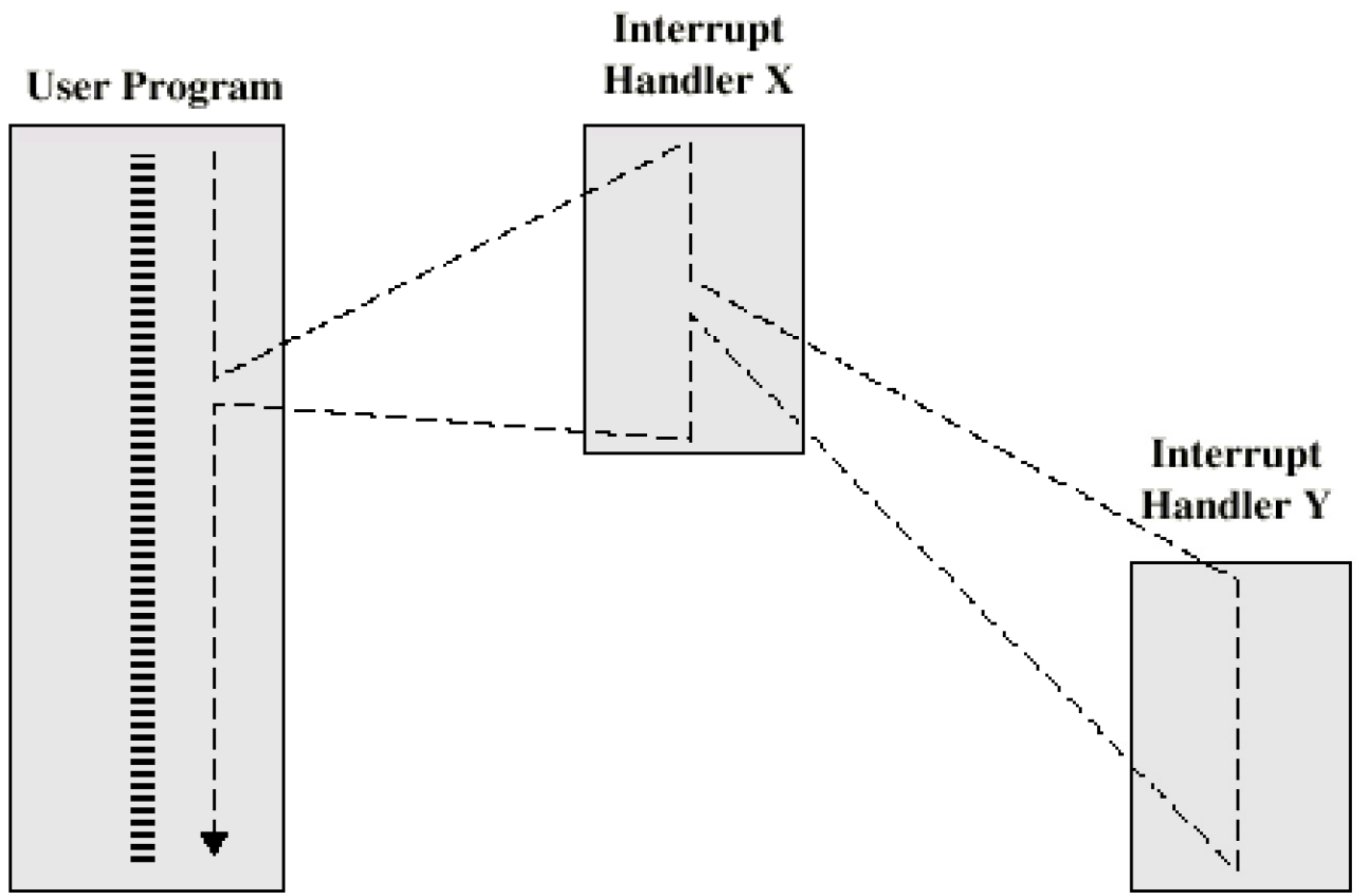
- Disable interrupts – Sequential Processing
  - Processor will ignore further interrupts whilst processing one interrupt
  - Interrupts remain pending and are checked after first interrupt has been processed
  - Interrupts handled in sequence as they occur
- Define priorities – Nested Processing
  - 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



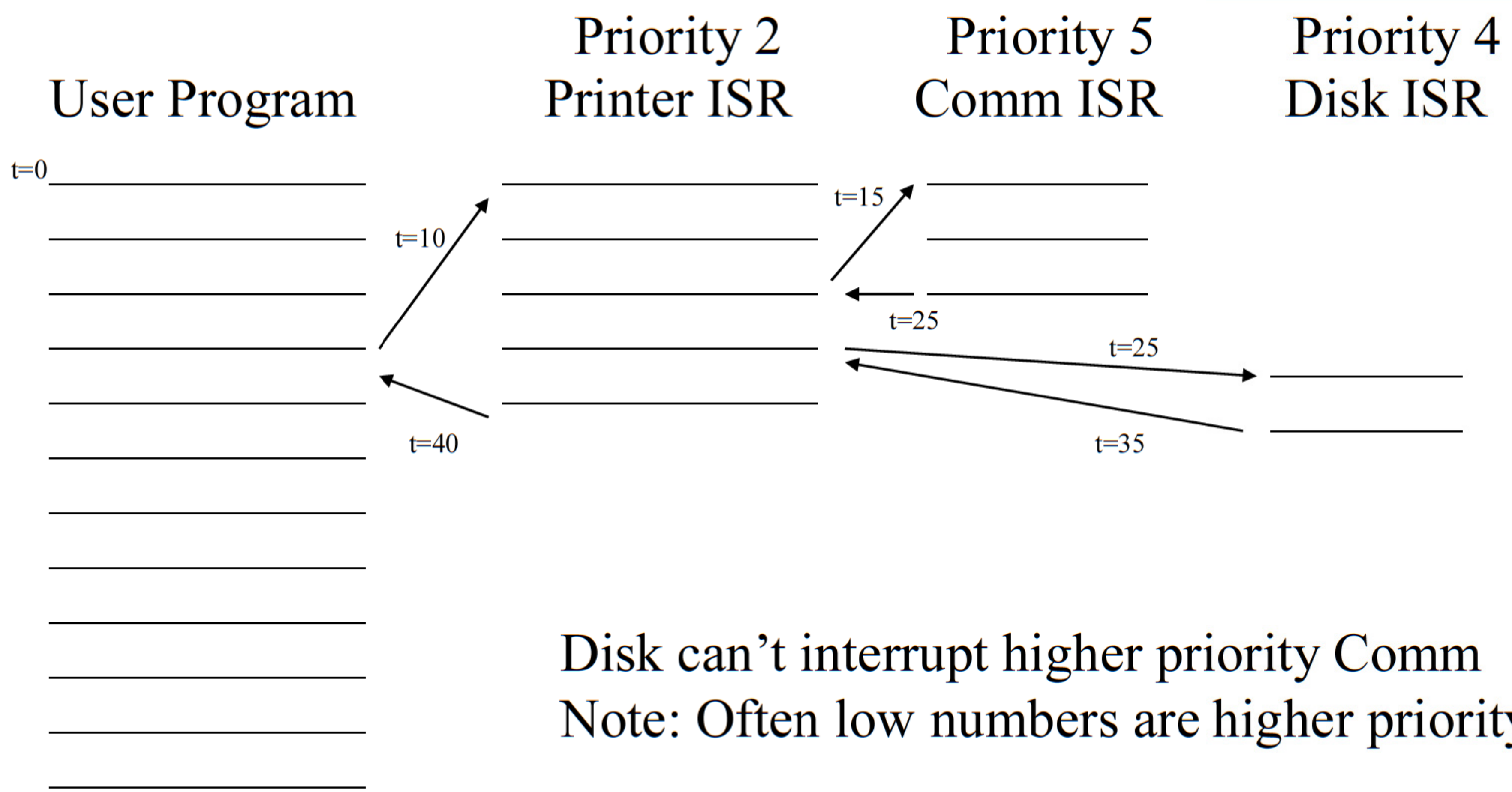
Disabled Interrupts – Nice and Simple

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

Connecting

- All the units must be connected
- Different type of connection for different type of unit
  - Memory
  - Input/Output
  - CPU

Memory Connection

- Memory typically consists of N words of equal length addressed from 0 to N-1
- Receives and sends data
  - To Processor
  - To I/O Device
- Receives addresses (of locations)
- Receives control signals
  - Read
  - Write
  - Timing

Input/Output Connection(1)

- Functionally, similar to memory from internal viewpoint
- Instead of N words as in memory, we have M ports
- Output
  - Receive data from computer
  - Send data to peripheral
- Input
  - Receive data from peripheral
  - Send data to computer

Input/Output Connection(2)

- Receive control signals from computer
- Send control signals to peripherals
  - e.g. spin disk
- Receive addresses from computer
  - e.g. port number to identify peripheral
- Send interrupt signals (control)

CPU Connection

- Sends control signals to other units
- Reads instruction and data
- Writes out data (after processing)
- Receives (& acts on) interrupts

Buses

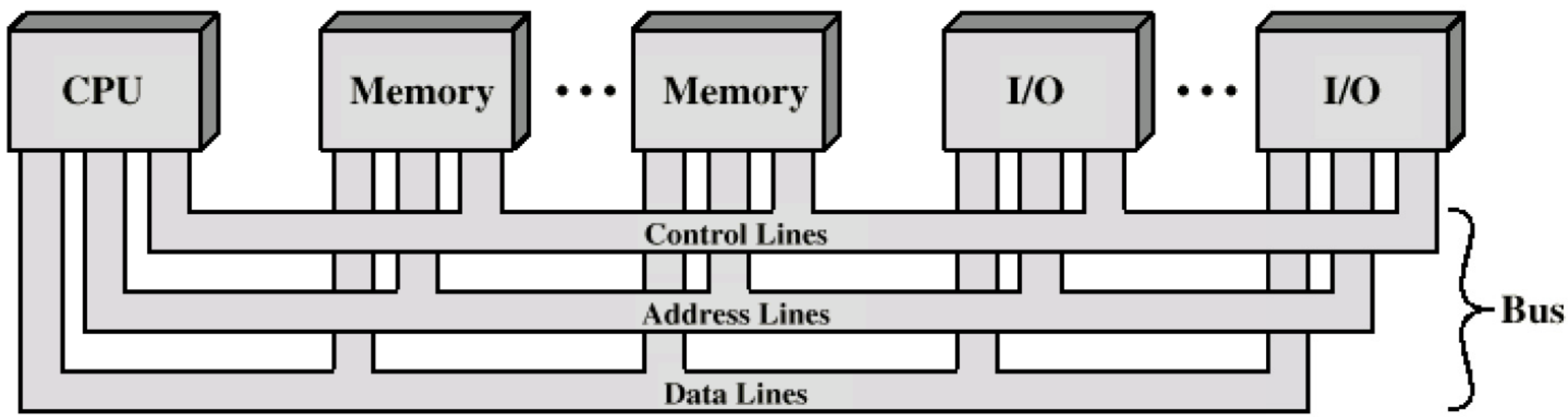
- There are a number of possible interconnection systems. The most common structure is the **bus**

- Single and multiple BUS structures are most common
- e.g. Control/Address/Data bus (PC)
- e.g. Unibus (DEC-PDP) – replaced the Omnibus

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 bit
  - What if the data bus is 8 bits wide but instructions are 16 bits long?
  - What if the data bus is 64 bits wide but instructions are 16 bits long?

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
    - RAM: 0 – BFFFF, ROM: C0000 - FFFFF
    - DOS only allowed first 640K to be used, remaining memory for BIOS, hardware controllers. Needed High-Memory Manager to "break the 640K barrier"

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
- Typical control lines
  - Memory read
  - Memory write
  - I/O read
  - I/O write
  - Interrupt request
  - Interrupt ACK
  - Bus Request
  - Bus Grant
  - Clock signals

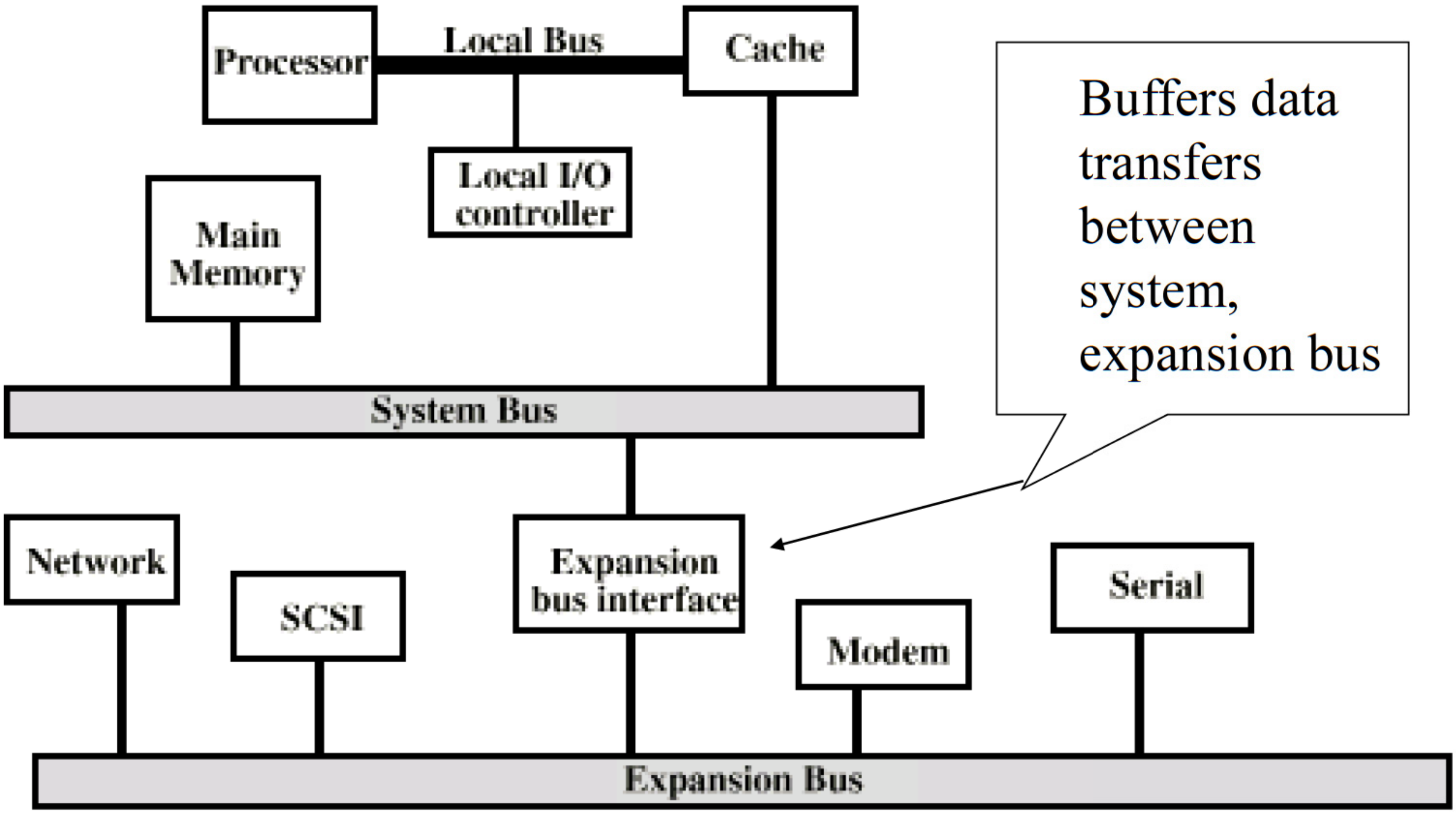
Big and Yellow?

- 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

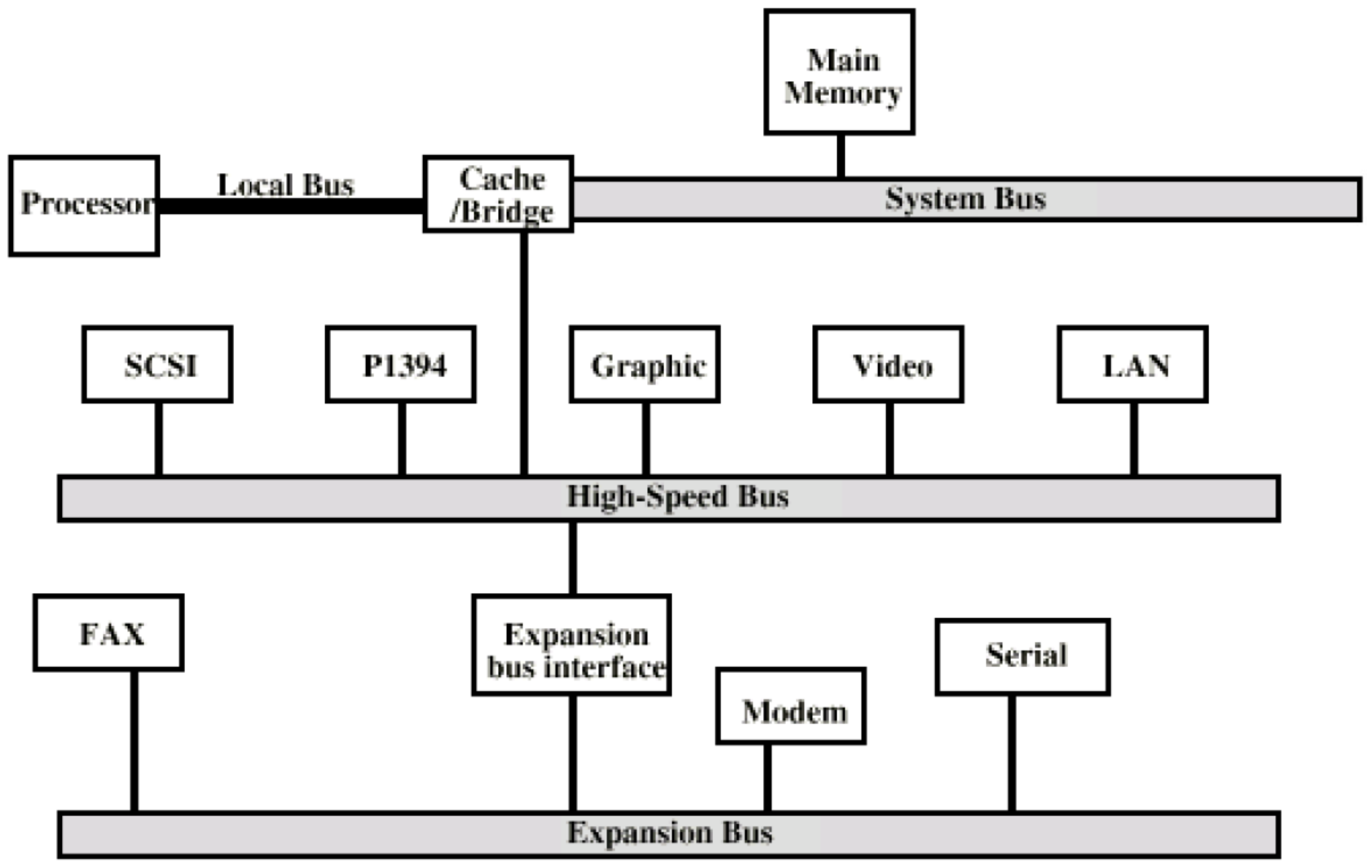
Traditional (ISA) (with cache)



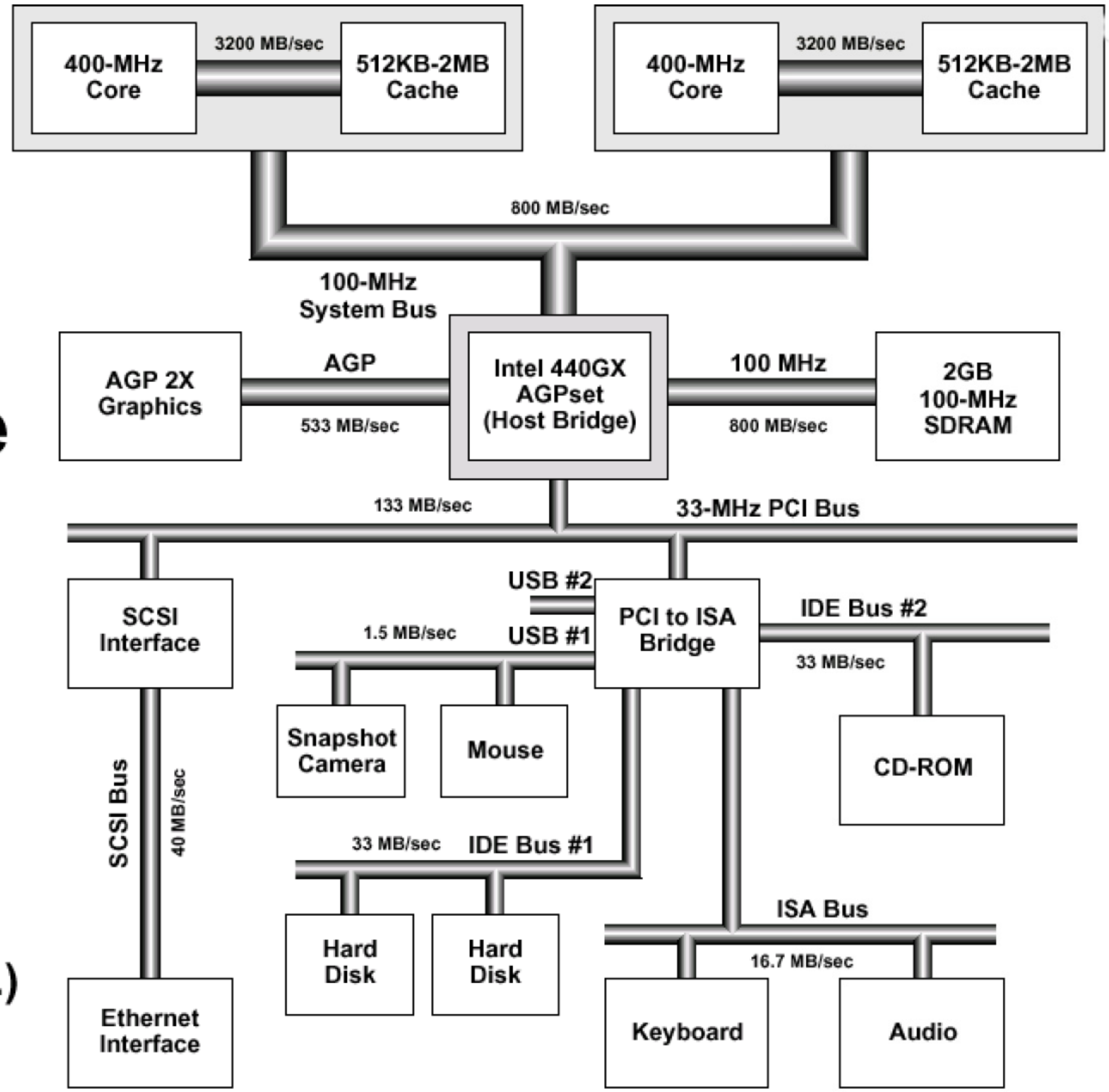
This approach breaks down as I/O devices need higher performance

High Performance Bus – Mezzanine Architecture

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



Bridge Based Bus Architecture

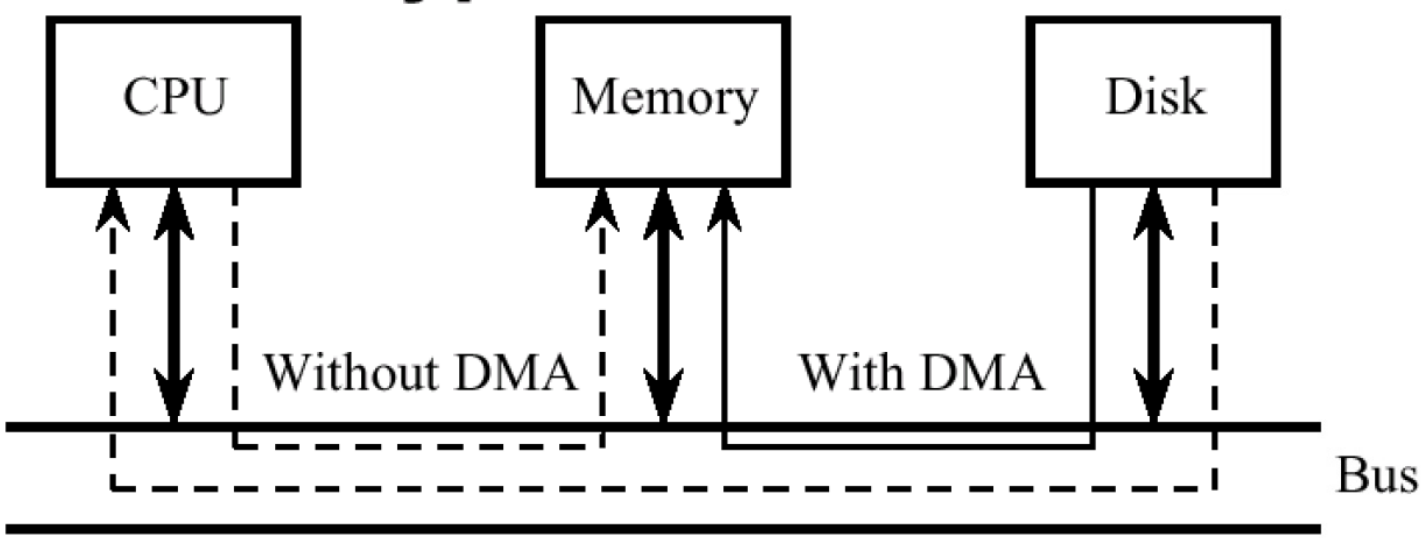


- Bridging with dual Pentium II Xeon processors on Slot 2.

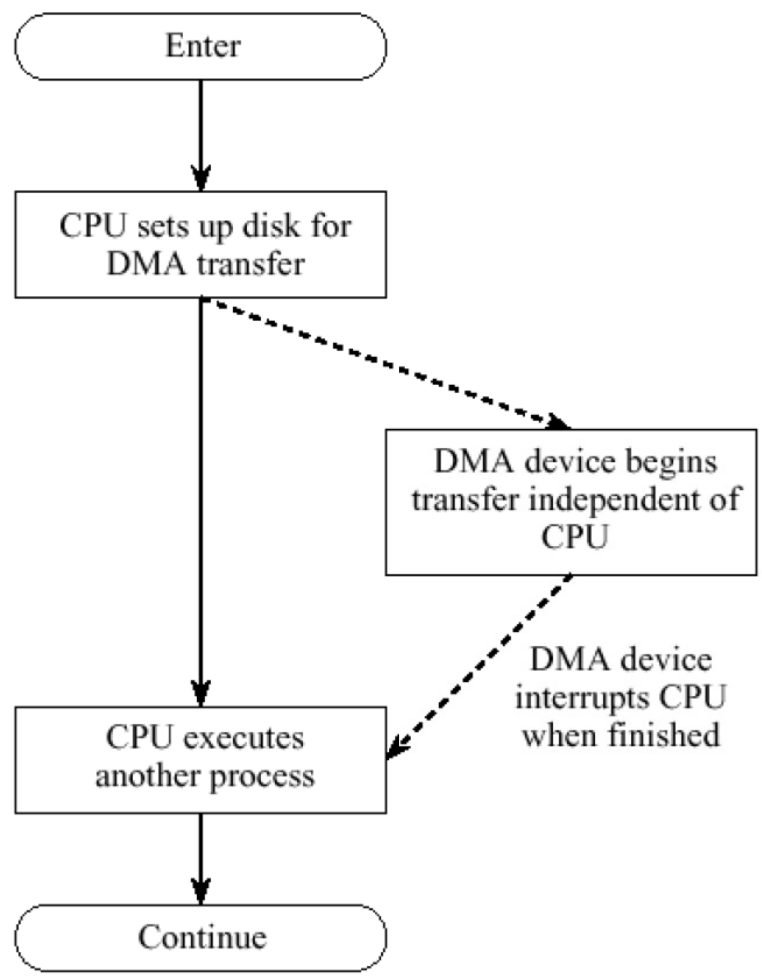
(Source: http://www.intel.com.)

Direct Memory Access

DMA Transfer from Disk to Memory Bypasses the CPU



DMA Flowchart for a Disk Transfer



Bus Types

- Dedicated
  - Separate data & address lines
- Multiplexed
  - Shared lines
  - Consider shared address, data lines
    - Need separate Address valid or Data valid control line
    - Time division multiplexing in this case
  - Advantage - fewer lines
  - Disadvantages
    - More complex control
    - Ultimate performance

Bus Arbitration

- More than one module may want to control the bus
  - e.g. I/O module may need to send data to memory and to the CPU
- But only one module may control bus at one time
  - Arbitration decides who gets to use the bus
  - Arbitration must be fast or I/O devices might lose data
- Arbitration may be centralized or distributed

Centralized Arbitration

- Single hardware device is responsible for allocating bus access
  - Bus Controller
  - Arbiter
- May be part of CPU or separate

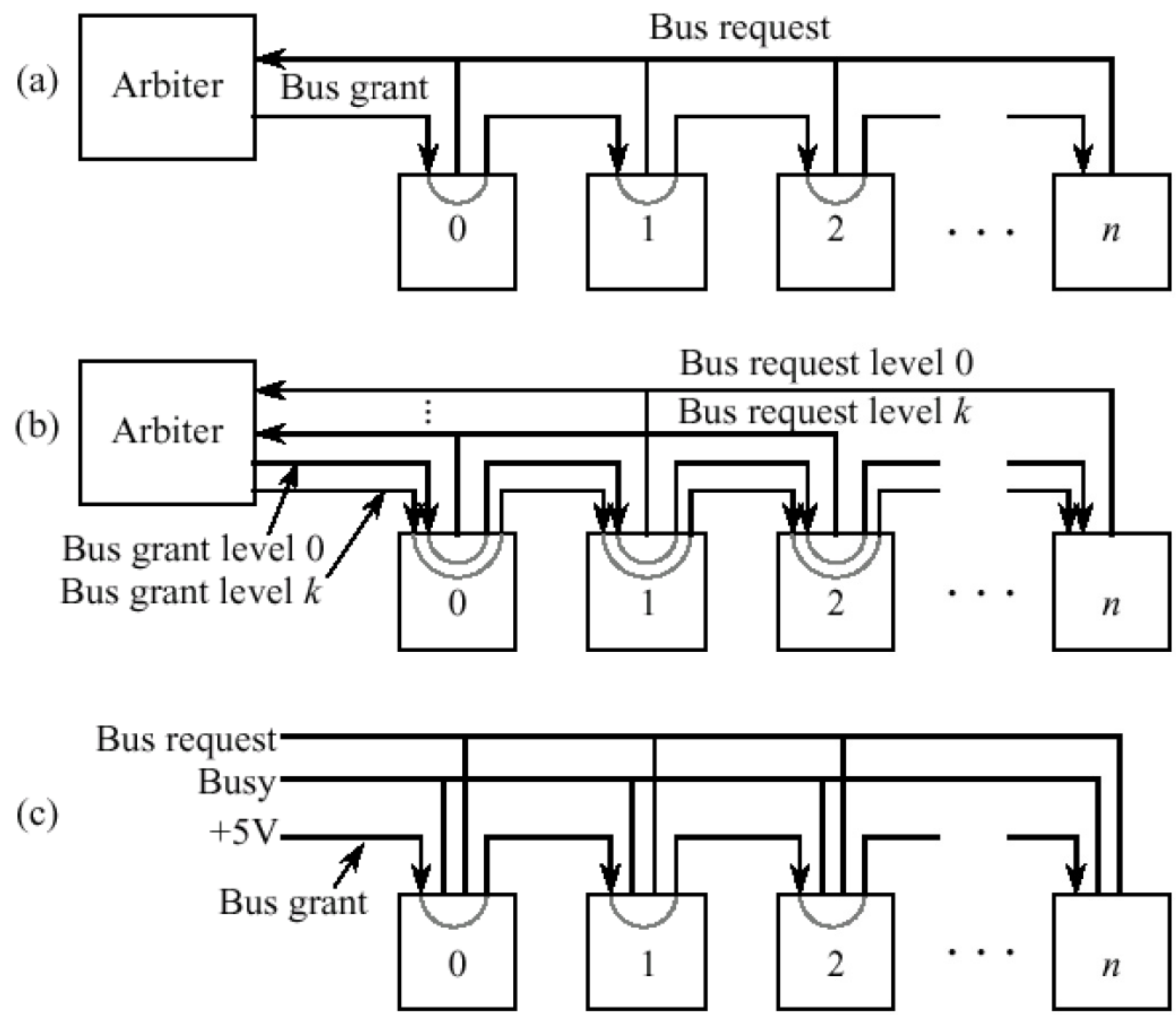
Distributed Arbitration

- No single arbiter
- Each module may claim the bus
- Proper control logic on all modules so they behave to share the bus

- Purpose of both distributed and centralized is to designate the master
- The recipient of a data transfer is the slave

- Many types of arbitration algorithms: round-robin, priority, etc.

Bus Arbitration



- **(a)Simple centralized bus arbitration; (b) centralized arbitration with priority levels; (c) decentralized bus arbitration. (Adapted from [Tanenbaum, 1999]).**

Daisy Chaining
of devices
What if a device breaks?

Devices to left higher priority

Bus Arbitration Implementations – Centralized

- Centralized
  - If a device wants the bus, assert bus request
  - Arbiter decides whether or not to send bus grant
  - Bus grant travels through daisy-chain of devices
  - If device wants the bus, it uses it and does not propagate bus grant down the line. Otherwise it propagates the bus grant.
  - Electrically close devices to arbiter get first priority
- Centralized with Multiple Priority Levels
  - Can add multiple priority levels, grants, for more flexible system. Arbiter can issue bus grant on only highest priority line

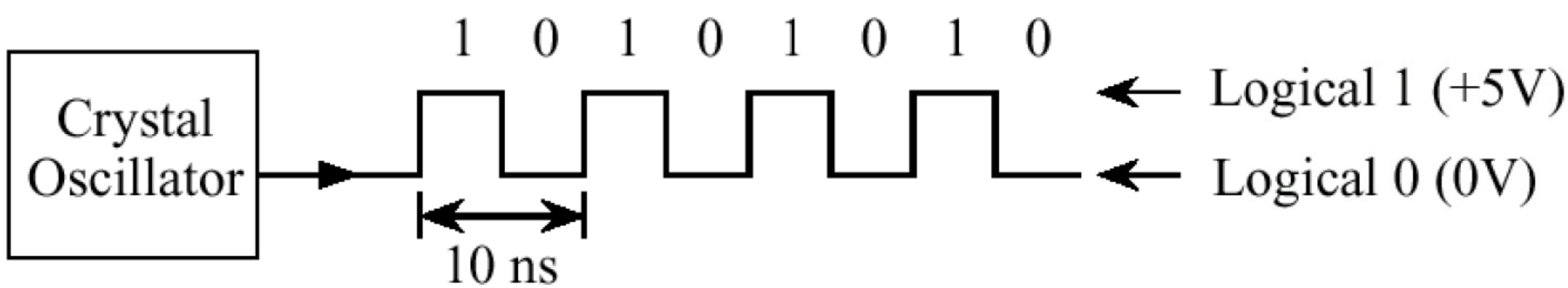
Bus Arbitration Implementation - Decentralized

- Decentralized
  - If don't want the bus, propagate bus grant down the line
  - To acquire bus, see if bus is idle and bus grant is on
    - If bus grant is off, may not become master, propagate negative bus grant
    - If bus grant is on, propagate negative bus grant
  - When dust settles, only one device has bus grant
  - Asserts busy on and begins transfer
  - Leftmost device that wants the bus gets it

Timing

- Co-ordination of events on bus
- Synchronous
  - Events determined by clock signals
  - Control Bus includes clock line
  - A single 1-0 is a bus cycle
  - All devices can read clock line
  - Usually sync on leading edge
  - Usually a single cycle for an event

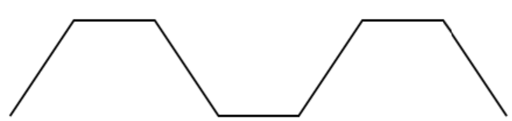
100 MHz Bus Clock



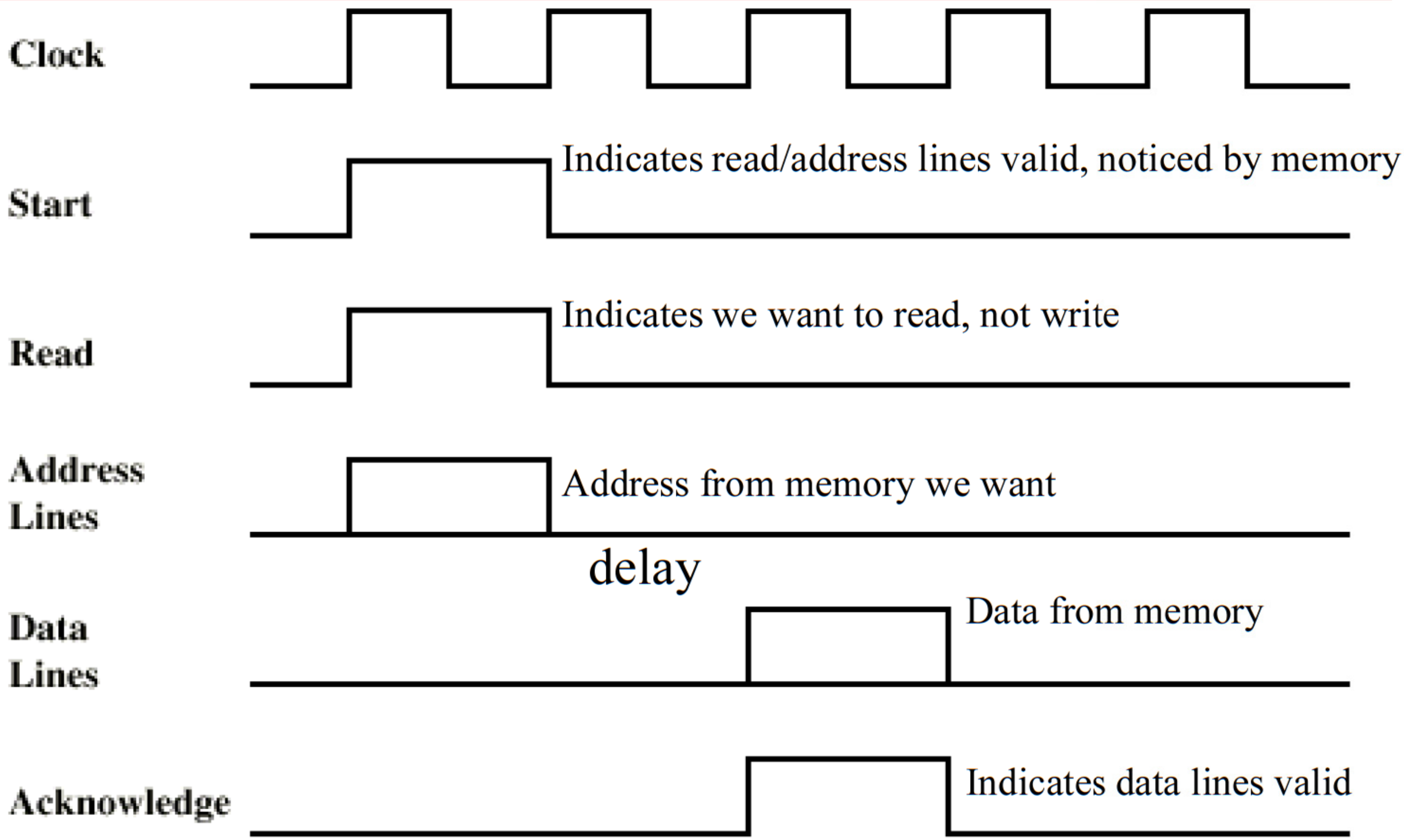
100 million cycles per second

1 cycle in (1/100,000,000) seconds = 0.0000001s = 10 ns

In reality, the clock is a bit more sawtoothed



Synchronous Timing Diagram
Read Operation Timing



Synchronous - Disadvantages

- Although synchronous clocks are simple, there are some disadvantages
  - Everything done in multiples of clock, so something finishing in 3.1 cycles takes 4 cycles
  - With a mixture of fast and slow devices, we have to wait for the slowest device
    - Faster devices can't run at their capacity, all devices are tied to a fixed clock rate
    - Consider memory device speed faster than 10ns, no speedup increase for 100Mhz clock
- One solution: Use asynchronous bus

Asynchronous Bus

- No clock
- Occurrence of one event on the bus follows and depends on a previous event
- Requires tracking of state, hard to debug, but potential for higher performance

- Also used with networking
  - Problem with “drift” and loss of synchronization
  - Some use self-clocking codes, e.g. Ethernet

Asynchronous Timing Diagram