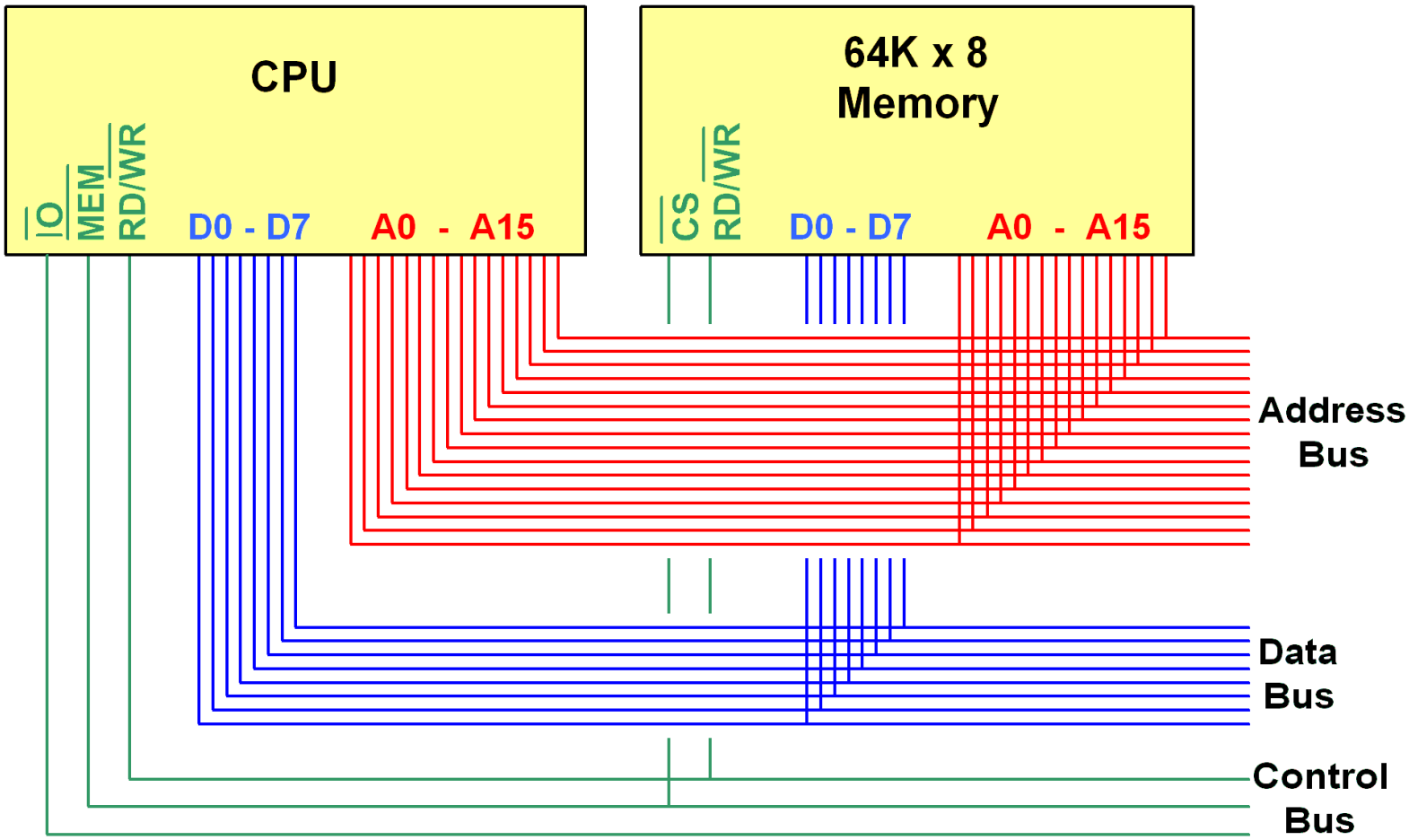
Today's Topics

- **Connecting memory and I/O chips to the system bus**
- **Address decoding for memory and I/O chips**
- **CPU connections to the system bus**
- **Computer system buses**
- **Bus masters vs. bus slaves**
- **Synchronous vs. Asynchronous buses**
- **Bus arbitration**
- **Special bus capabilities**
- **Sample CPU chips: Pentium-II, UltraSPARC II and picoJAVA II**
- **Samples Buses – the ISA, PCI, and AGP buses**
- **Chipsets**
- **The USB Bus**

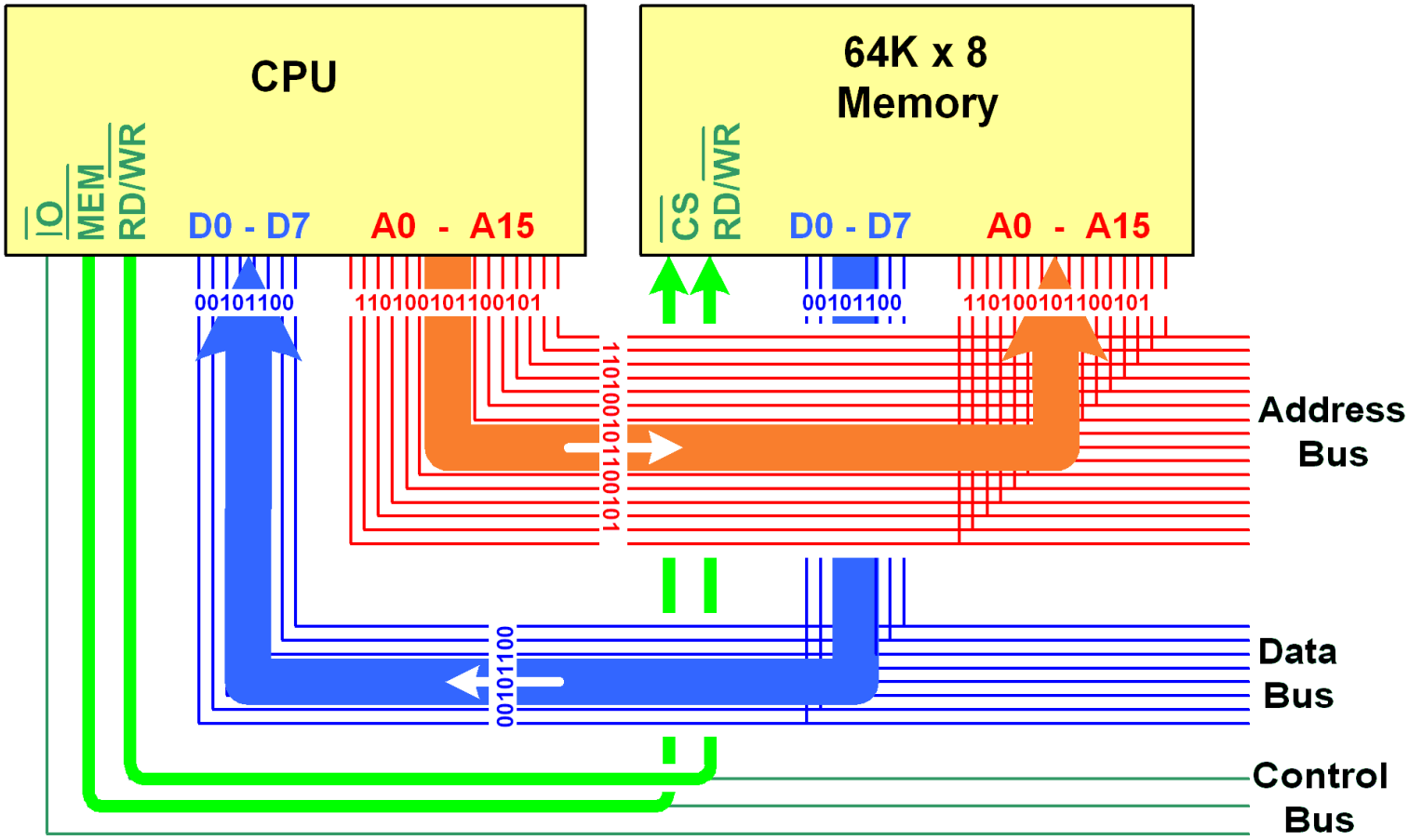
Connecting the CPU to Memory

The CPU has address, data and control pins which are connected to the system bus. The corresponding memory pins are also connected to the bus so that it can respond to read or write requests from the CPU. For simplicity, our examples will use a CPU chip with 16 address lines and 8 data lines (similar to the 8051):



Reading Data from Memory

To read data, the CPU asserts the $\overline{\text{MEM}}$ and RD/$\overline{\text{WR}}$ signals and sends the requested address over the bus. The memory sees a $\overline{\text{CS}}$ (Chip Select) request and then responds by sending the data at the requested address back over the bus to the CPU.



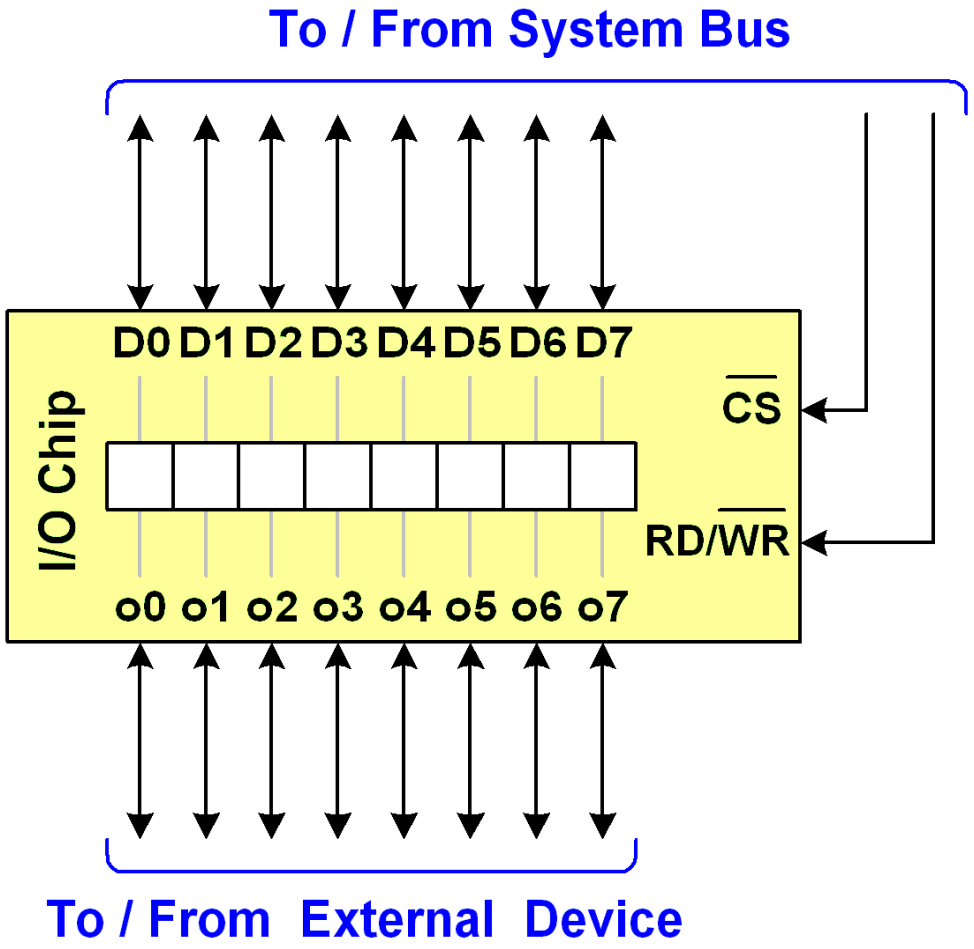
The entire process is known as a "Bus Transaction" or "Bus Cycle". Writing to memory is similar, but the CPU sends the opposite RD/$\overline{\text{WR}}$ control signal and the data flows from the CPU to memory instead of the other way around.

I/O Ports

An I/O port is a circuit that allows the computer system to be connected to an external I/O device such as a keyboard or a network. The external devices send and accept binary signals much the same way that a memory chip would. The I/O port is the way that these I/O signals are passed to and from the address, data and control lines on the system bus.

The I/O port acts very much like a register – it stores a word of information that is being read or written. The difference is that the bits in the register are connected to the external I/O device and can be used to send data to the device or, for input devices, the bits can be loaded from data sent from the I/O device.

The chips that contain I/O ports have connections to the system bus that are very similar to memory chips. But unlike memory chips (which hold thousands or millions of cells), most I/O chips have only one or, at most, a very small number of I/O ports. So I/O chips either don't have any address connections at all or, at most, only a very few of them.

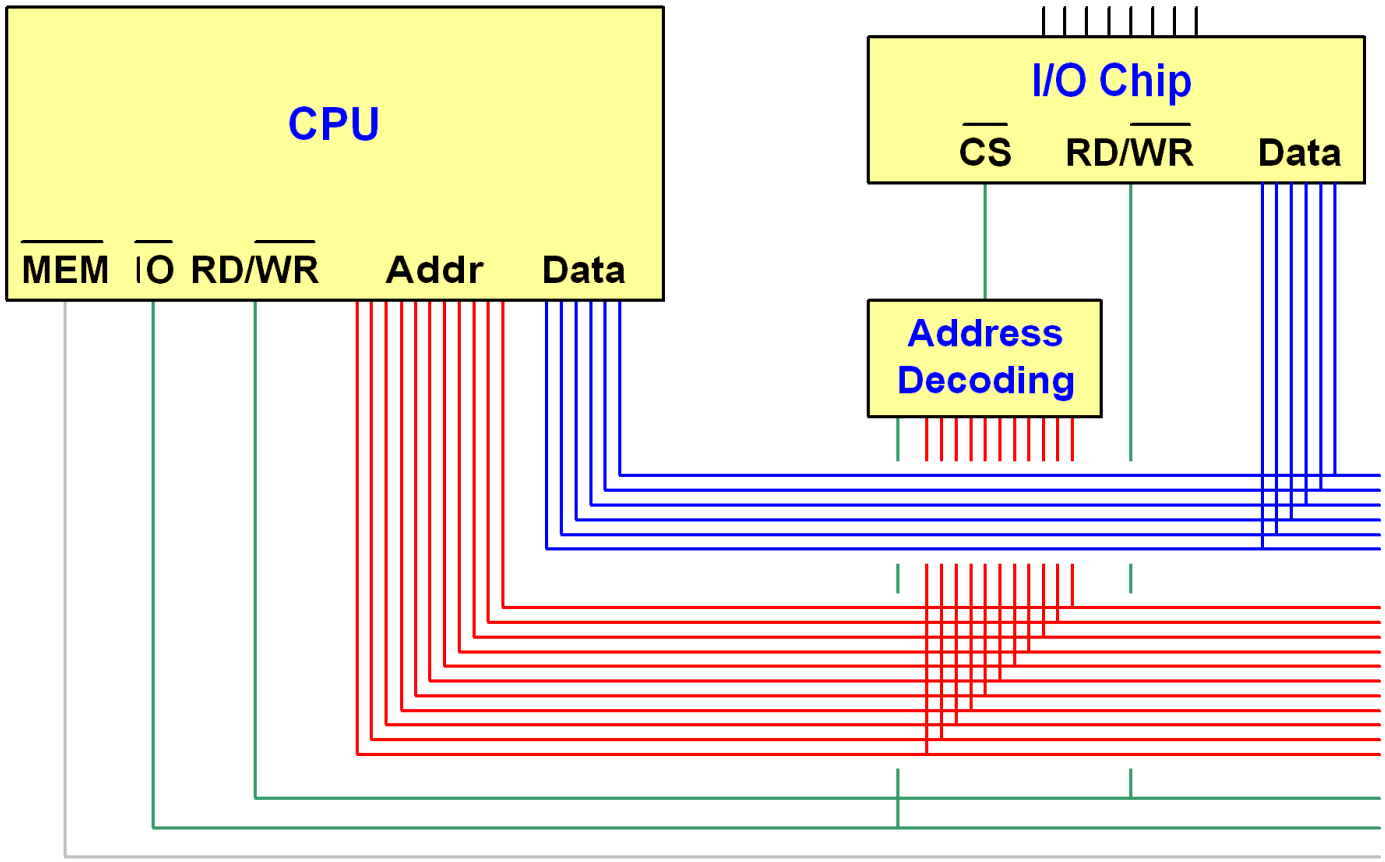


I/O Chip Addressing

I/O ports are accessed by the CPU in much the same manner as memory chips, except that:

- **The CPU asserts it's $\overline{IO}$ line when reading or writing to I/O ports, instead of it's $\overline{MEM}$ line.**

- **Since I/O chips have few or no address lines, external logic must be used to decode the address. This logic needs to activate the I/O chip when the correct address is present on the address bus *and* the $\overline{IO}$ signal is being asserted.**



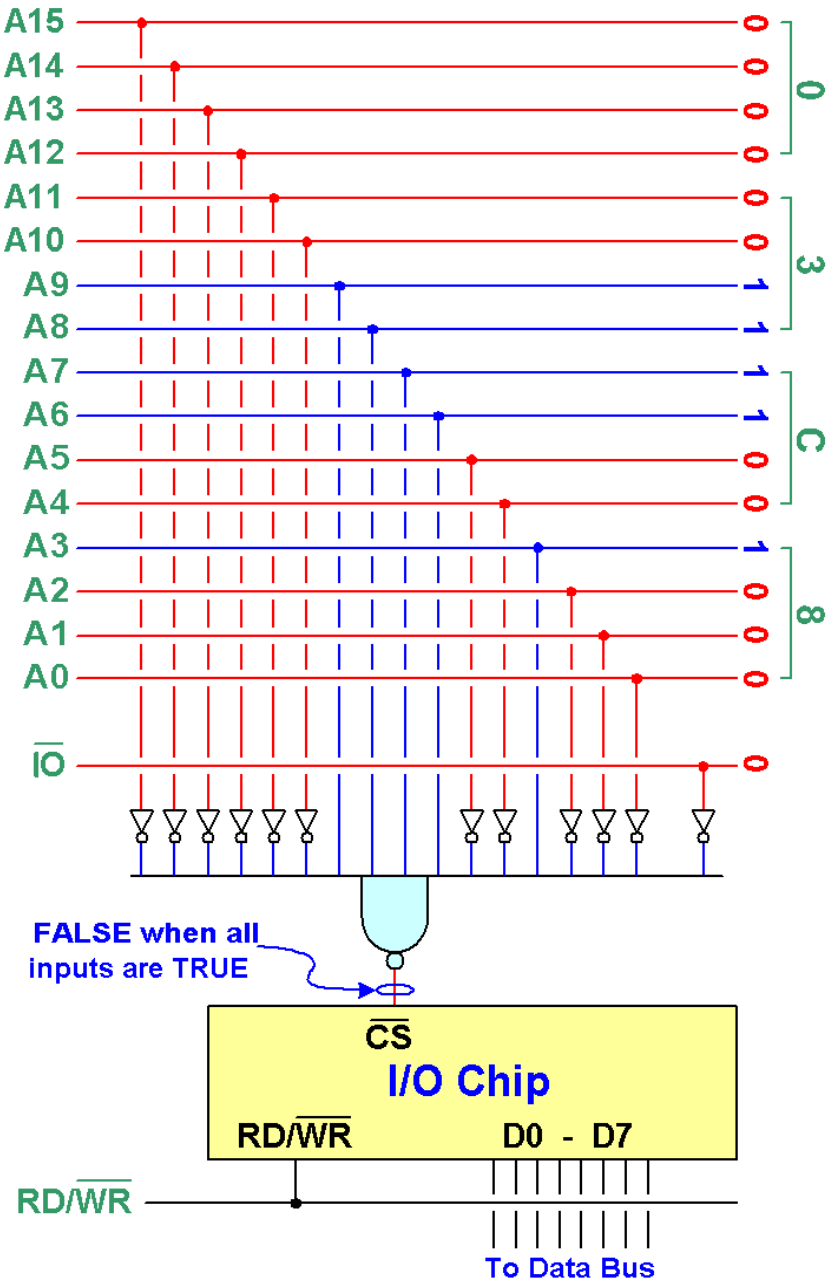
I/O Chip Addressing Example

This diagram shows an example of how an I/O chip addressed as I/O port hex 3C8 is connected to the address and control bus signals (the data connections are omitted for simplicity).

Hex 3C8 is 0000 0011 1100 1000 in binary – this is the combination of address signals that the decoding circuitry must look for. The decoding circuitry must also look for an asserted $\overline{IO}$ signal (that is, a FALSE value).

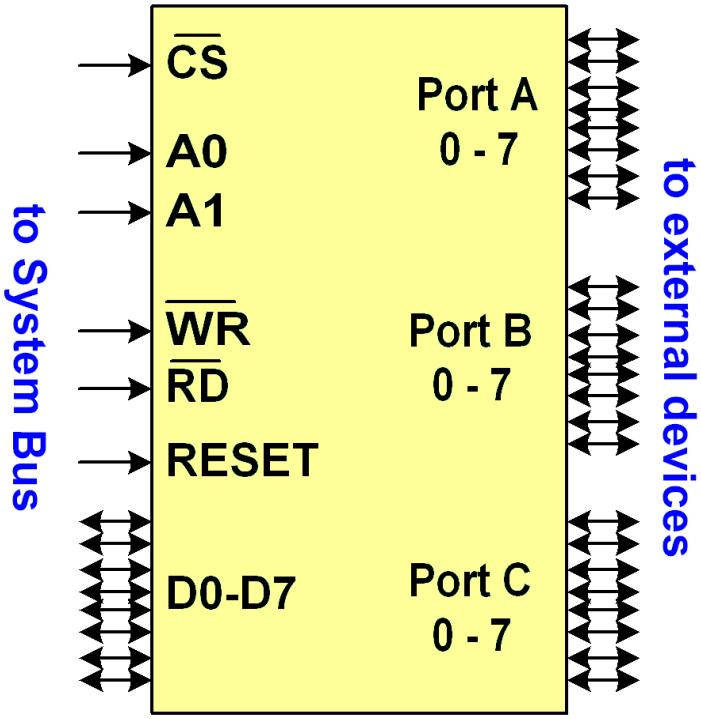
The decoding circuit consists of a NAND gate with it's inputs attached to the $\overline{IO}$ and address signals. The NAND gate produces a FALSE output when all inputs are TRUE. Since the FALSE output activates the $\overline{CS}$ signal on the chip, the I/O chip is therefore activated when all of the NAND gate inputs are TRUE.

Inverters (NOT logic elements) are used on the lines which have to be FALSE in order to activate the I/O chip. This means that the NAND gate will only activate the I/O chip when all the lines that are supposed to be FALSE are in fact FALSE and the lines that are supposed to be TRUE are in fact TRUE.



Multi-Port I/O Chips

It is possible for one I/O chip to contain more than one I/O port. Here's a diagram showing the Intel 8255A three-port parallel I/O chip:



This chip has three ports that can be configured as input or output ports, as well as a fourth internal control register used to do the configuring. So this chip can respond to four different I/O addresses. To do this, there are two address lines (A0 and A1).

This chip also uses separate RD and WR signals (as opposed to a single $\overline{RD}/\overline{WR}$ signal) and has a RESET signal that allows the chip's configuration to be reset to it's default.

8255A Addressing Example

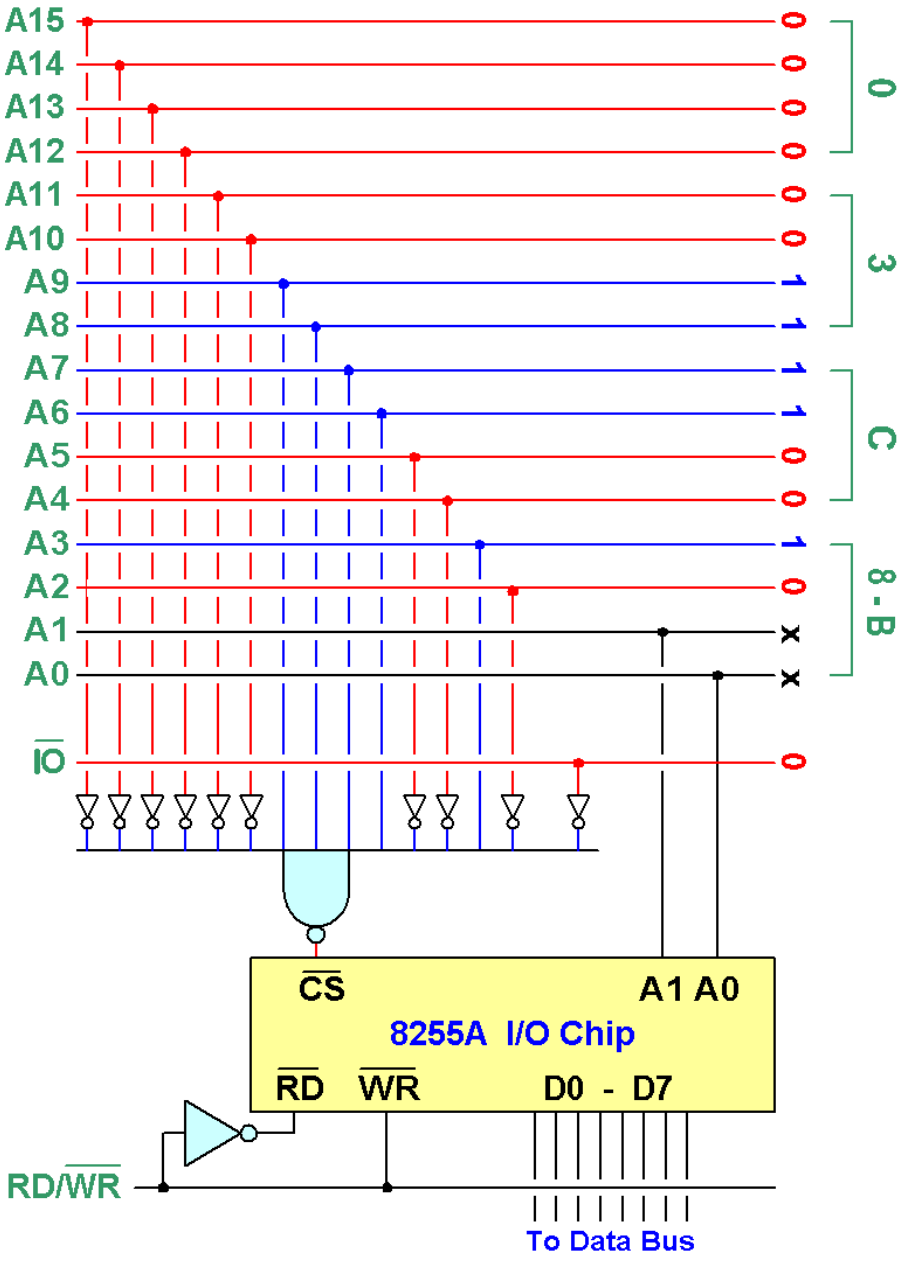
This diagram shows how an 8255A chip addressed as I/O port hex 3C8 is connected to the address and control bus signals (data connections are omitted for simplicity)

This chip's A0 and A1 lines are connected directly to their corresponding lines on the address bus. The remaining 14 address lines are decoded using essentially the same logic as was used in the previous example (but note the difference in the RD/$\overline{WR}$ logic to deal with the separate RD and WR pins on this chip)

With this decoding logic, the I/O chip is activated when any one of the following four addresses on the bus and the $\overline{IO}$ signal is asserted:

3C8	3CA
3C9	3CB

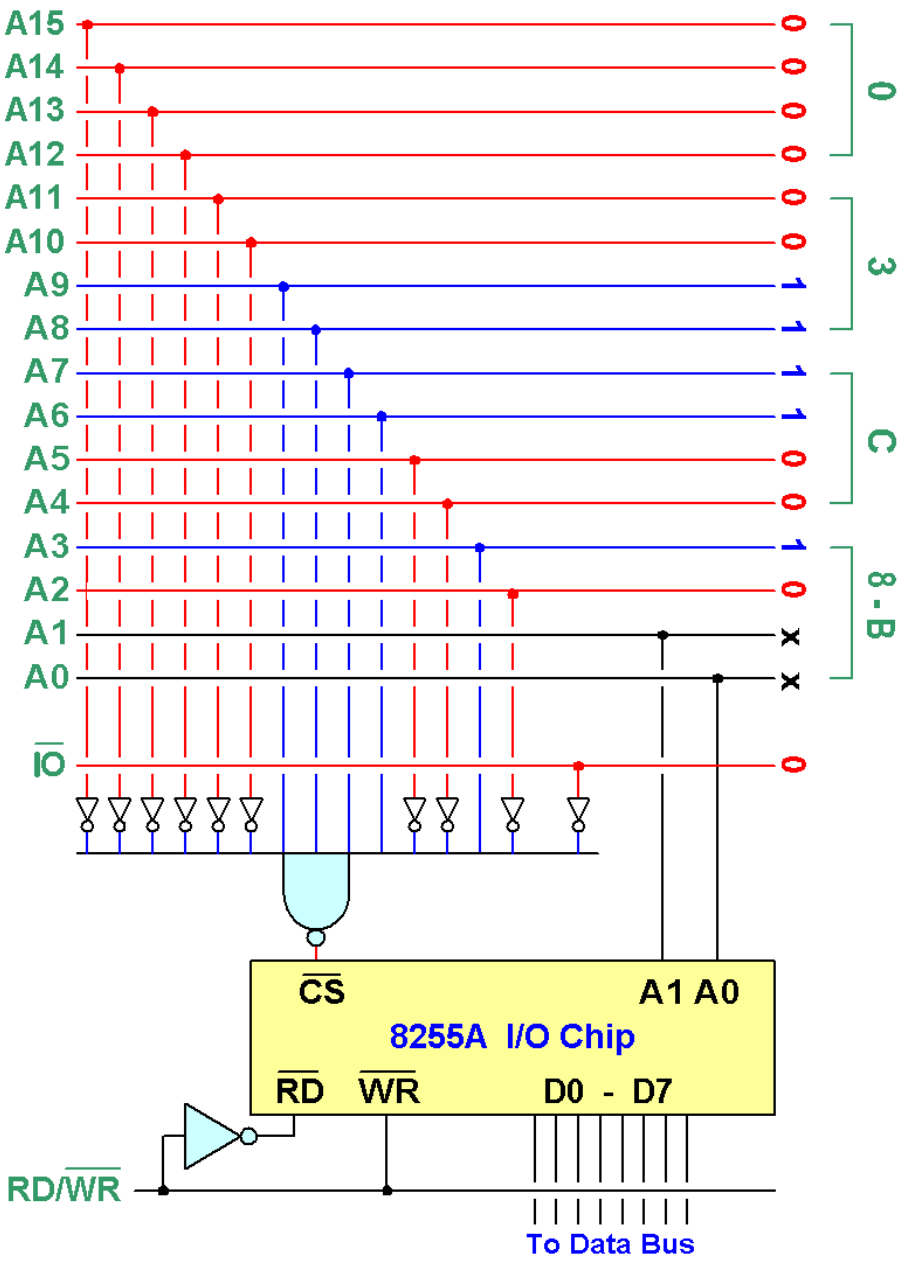
These four addresses have the same high-order 14 address bits, and they differ only in the low-order 2 address bits. When the chip is activated, the two low-order address bits are used to determine which of the four I/O port addresses will be used.



Interfacing I/O Chips

So to interface an I/O chip to a computer, you must:

- **Understand which address bus lines must be FALSE and which must be TRUE in order to activate the I/O chip. This is determined by the I/O port address for the chip.**
- **Connect any address lines from the I/O chip to the equivalent lines on the bus.**
- **Connect the other address lines from the bus to the a NAND gate leading to the I/O port's Chip Select signal. Directly connect the lines that must be TRUE, and use an inverter for the lines that must be FALSE**
- **Also connect the IO control signal from the bus to the NAND gate through an inverter (since this signal must also be FALSE in order to activate the I/O chip).**



Exercise 1 – I/O Port Interfacing

The diagram at right shows a 16-port I/O chip connected to a system that can address a megabyte of memory The I/O controller chip responds to addresses starting at hex FF930.

What is wrong with the diagram?

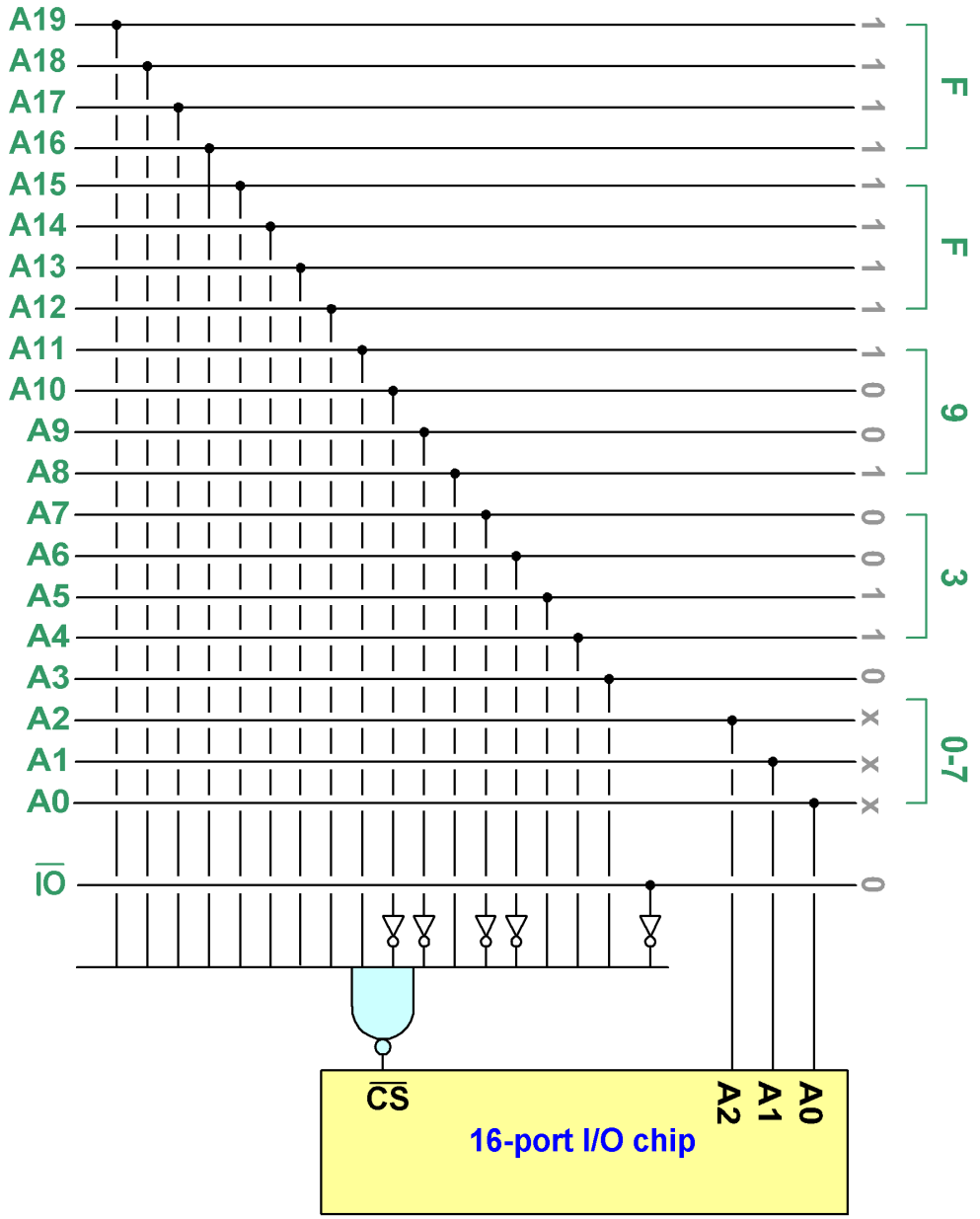
A **The chip has too many address lines**

B **The chip doesn't have enough address lines**

C **The inverters on the NAND gate inputs are connected to the opposite address lines that they should be connected to**

D **The "Not-IO" line should actually be "Not-MEM"**

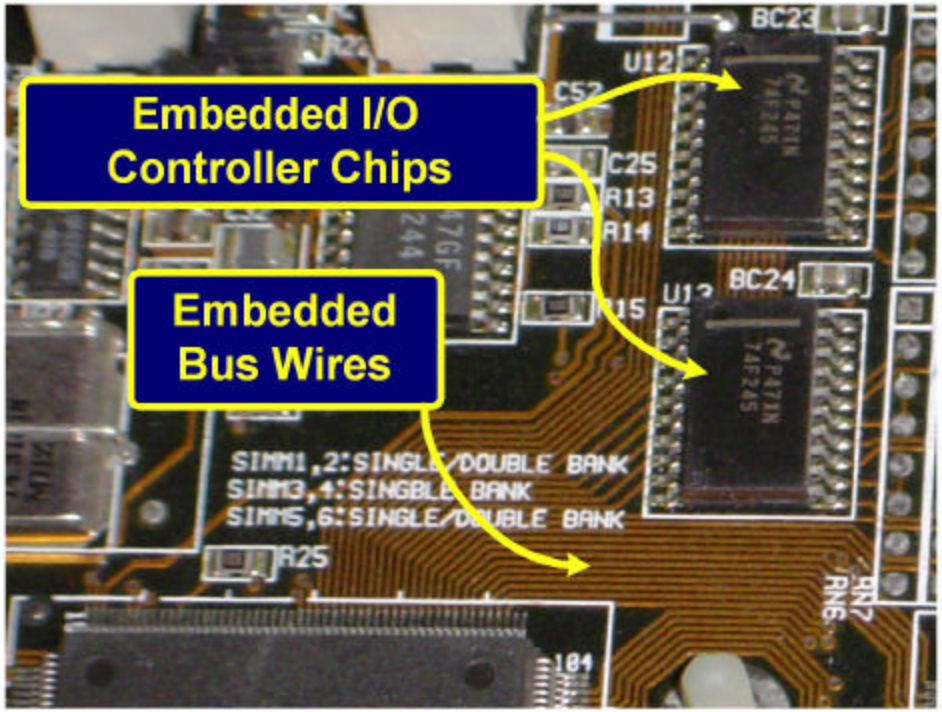
E **The NAND gate should be an AND gate**



How Ports are Physically Connected to the Bus

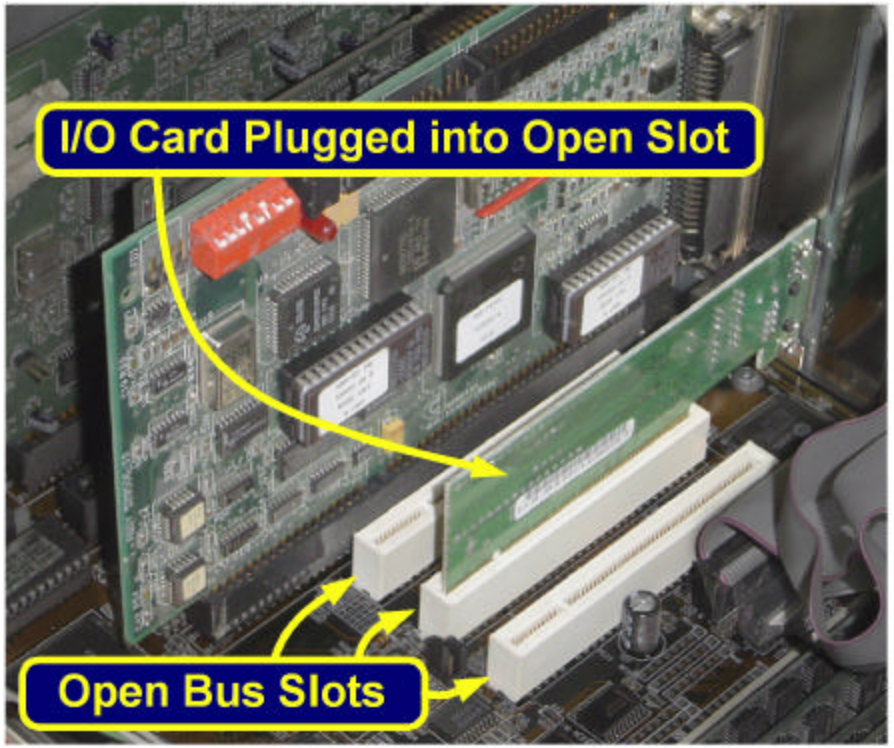
I/O ports and controllers may be physically connected to the system bus in two ways:

Embedded Bus



Embedded devices are “wired in” and cannot be changed.

Open Bus



Open bus slots allow new devices to be added or old devices to be upgraded

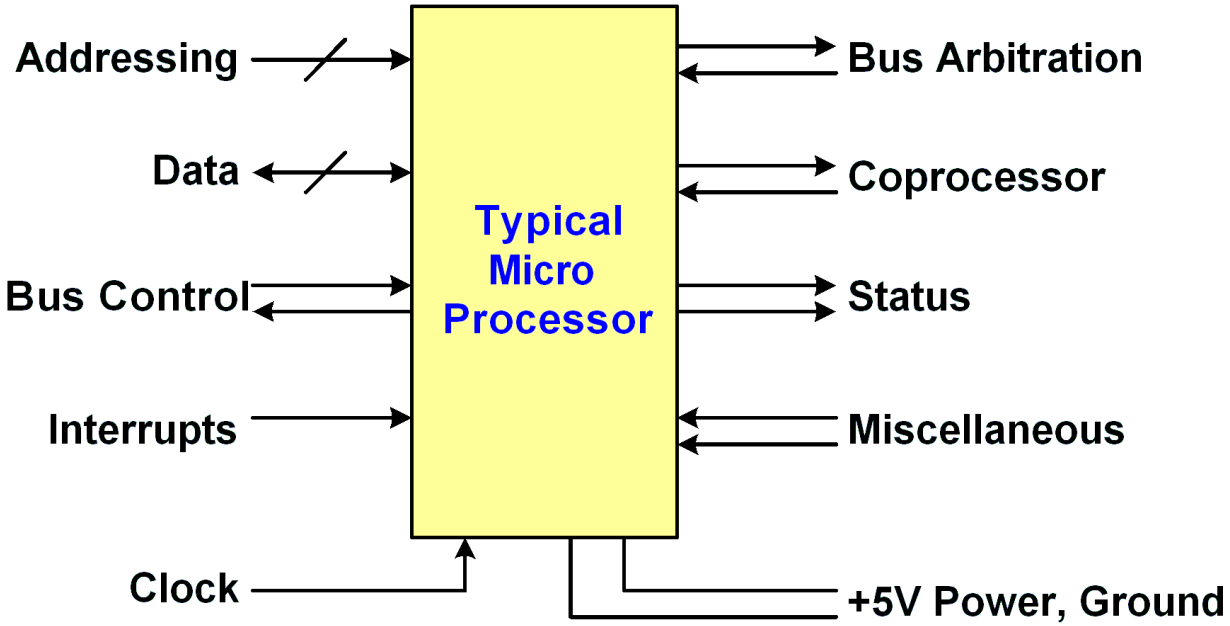
Most systems with open bus slots also include some devices connected to an embedded bus. For example, most desktop computer systems have a USB controller integrated on the motherboard which is connected via an embedded bus. An additional USB controller card could also be added to an open bus slot.

CPU Connections

The CPU chip is the heart of the system and controls the operation of all the other devices. As we've seen, the CPU communicates with other devices over a Bus using Address, Data and Control signals. The number of Address and Data signals tell us something about the CPU:

- **The number of address signals governs the maximum number of memory cells that the CPU can access according to the power of 2 rule. For example, 32 address lines means that the CPU can access 2^{32} = 4G memory cells (this usually means 4G bytes)**
- **The number of data signals indicates how much information the CPU reads or writes with each bus transfer. This is an important factor in the CPU's performance.**

The control bus signals are also important. So far we've seen "MEM", "IO" and "RD/WR" control signals, but a typical CPU has many other control signals too:



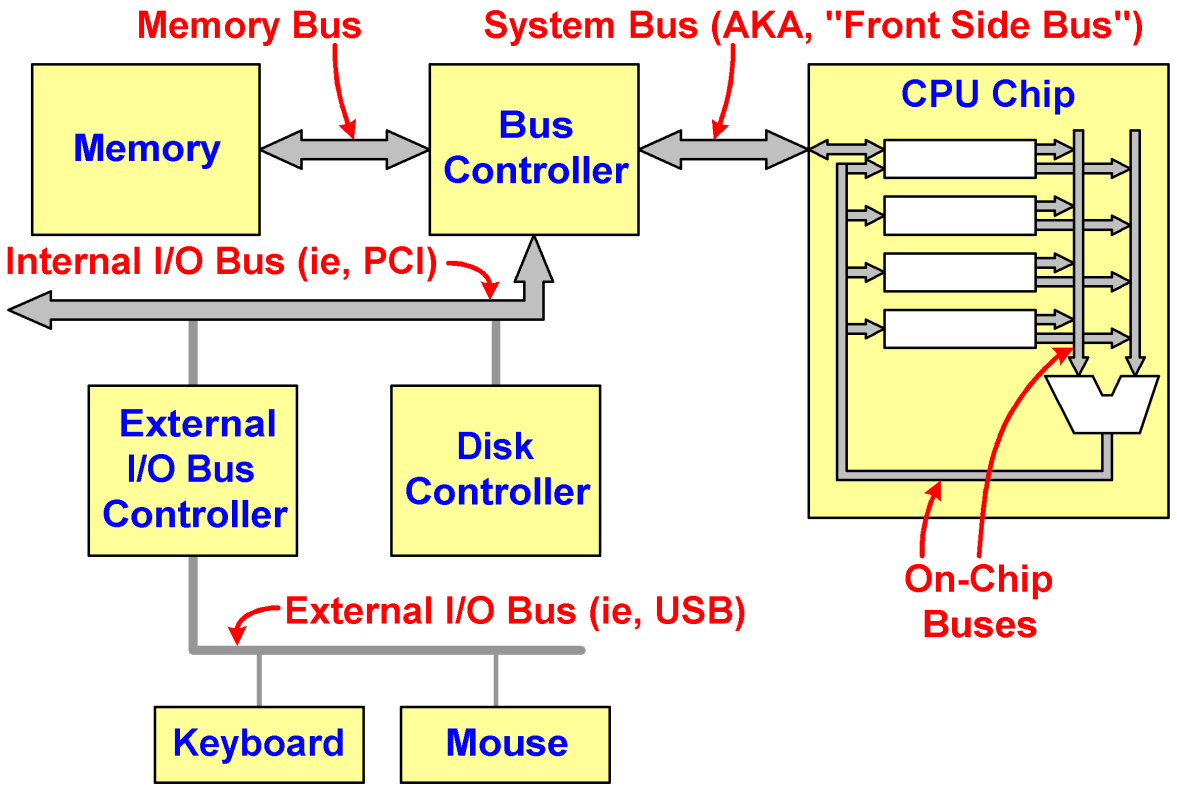
CPU Control Signals

Here are some of the types of control signals used by a typical CPU:

Bus Control	**The CPU uses these signals to control the bus and other devices in the system. These include the MEM, IO, and RD/WR signals.**
Interrupts	**These signals are generated by other devices in the system when they want the CPU's attention. The CPU responds to these signals by stopping the current program and jumping to an "Interrupt Service Routine" to process the request.**
Bus Arbitration	**These signals allow the CPU to share control of the bus with other devices in the system. This is useful for "smart" devices that can transfer data to or from memory without needing the CPU's help.**
Coprocessor	**Older CPUs often had separate external chips to handle tasks such as floating-point arithmetic. Modern CPUs have this type of circuitry built in, but many still use similar signals to coordinate activity when two or more CPUs are connected to the same bus.**
Status	**Some CPUs output status signals to indicate when an error has occurred or when they are halted.**
Miscellaneous	**These can include signals to reset the CPU or for communicating with special I/O chips.**

Computer Buses

This diagram shows some of the types of buses that can exist in a computer system:



Buses have different characteristics and features depending on how they are used. The main system bus that interconnects the CPU with memory is typically the fastest bus in the system.

I/O buses are usually standardized so that devices from different manufacturers can be used without compatibility problems. The internal I/O bus is the one which gets the most attention in a computer system because it's what the major I/O devices such as video or network cards plug into. It's what we'll be talking about next.

Some Buses

Here are some of the more common buses used during the Personal Computer era:

Name	Timeframe	Bandwidth	Notes
ISA	1980-199~	4-16 MByte/sec	Original 4.7MHz IBM PC 8-bit bus, later extended to 8MHz / 16 bits for the 80286
Nubus	1985-199~	20 MByte/sec	Used in early Apple MacIntosh systems
MCA	1987-199~	33 MByte/sec	IBM's proprietary 32-bit PS/2 bus. Never very popular due to licensing costs.
EISA	1988-199~	33 MByte/sec	32-bit Industry competitor to MCA. Never very popular due to hardware costs.
VESA	1992-1995	160 MByte/sec	Special-purpose video card bus used in 486 systems before PCI
PCI	1992-now	132 – 2000+ MByte/sec	Former industry standard bus, still supported in most new computers
AGP	1997-2007	250 – 2000+ MByte/sec	Special variant of PCI used specifically for graphics cards
PCI Express	2005-now	250 – 8000+ MByte/sec	Serial, switched bus used in most new computer systems

Bus Masters vs. Bus Slaves

Most systems have many devices connected to the bus, but because there is only one bus, only one data transfer can happen at any given time.

The device that requests the data transfer is called the "Bus Master" and the other device is called the "Bus Slave". The data can go in either direction: from master to slave or from slave to master.

Typical Bus Masters include:

- **CPU – The CPU is usually the Bus Master in the computer system (and there may be more than one CPU)**
- **I/O Devices – The CPU can program "Smart" I/O devices with DMA (Direct Memory Access) capability to perform their own transfers to or from memory without requiring any more action by the CPU**

Typical Bus Slaves include:

- **Memory – Memory is always a passive device and never initiates any data transfers**
- **I/O Devices – The CPU can transfer data to or from I/O devices**

In a system that has more than one Bus Master, there needs to be a way to decide who will take control of the bus. This job is done by Bus Arbitration circuitry (to be discussed later).

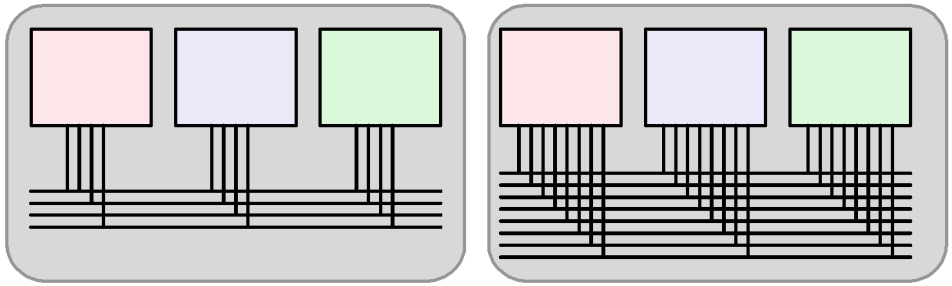
Bus Evolution

Buses have been evolving to meet the needs of faster CPUs and larger memories. Buses evolve in two primary ways:

They get wider

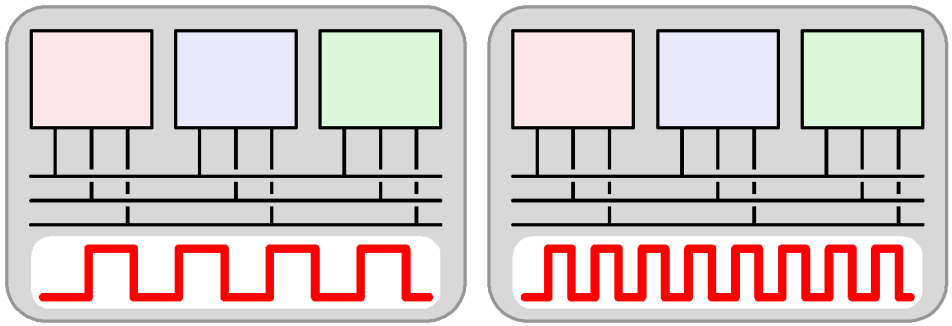
A wider bus has more:

- **data lines (allows more data to be transferred with each bus cycle, improving performance), and/or**
- **more address lines (allows more memory capacity)**



They get faster

More transfers/second increases performance

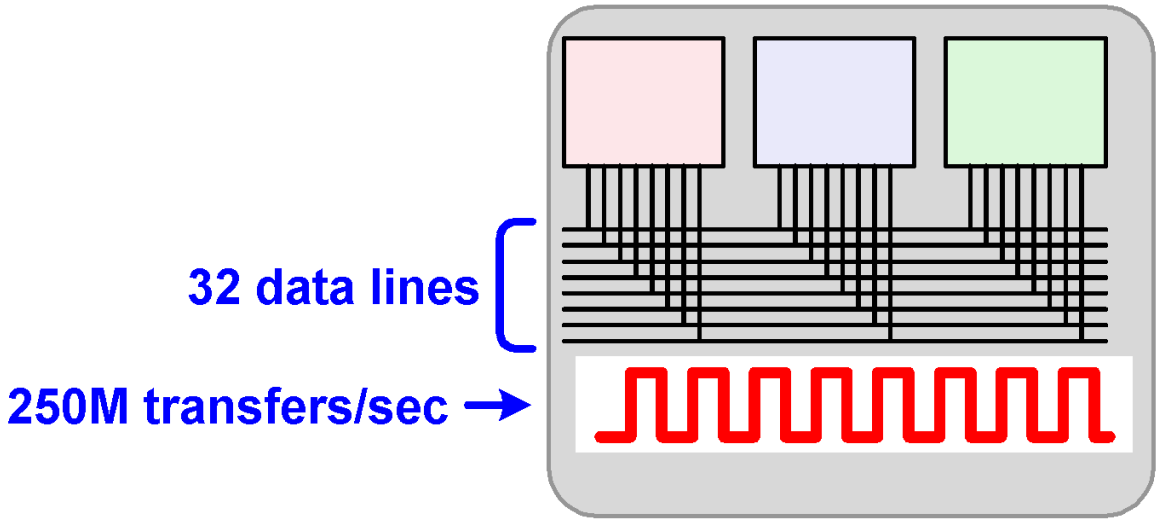


As buses get wider and faster, bus skew (different signals arriving at the other end at different times) becomes more of a problem.

One way to combat having too many wires on the bus is to use a multiplexed bus which uses the same wires for address and data. In this type of bus the address is sent first, and then the data is sent over the same set of wires. This reduces the number of wires at the expense of protocol complexity and performance.

Exercise 2 – Bus Performance

A bus has 32 data lines and can perform 250M transfers per second. What is the total bandwidth (amount of data transferred per second) for the bus?



A 250M Bytes/sec

B 32M Bytes/sec

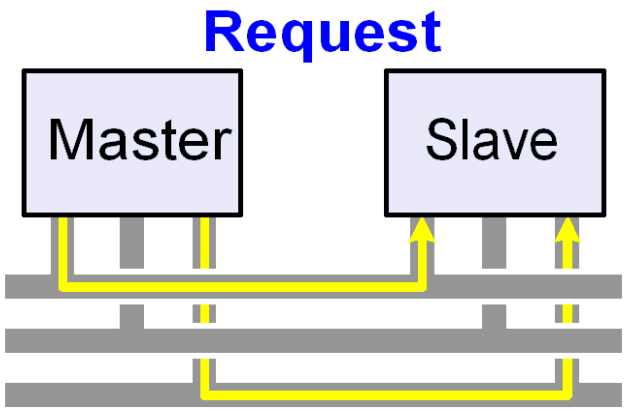
C 1000M Bytes/sec

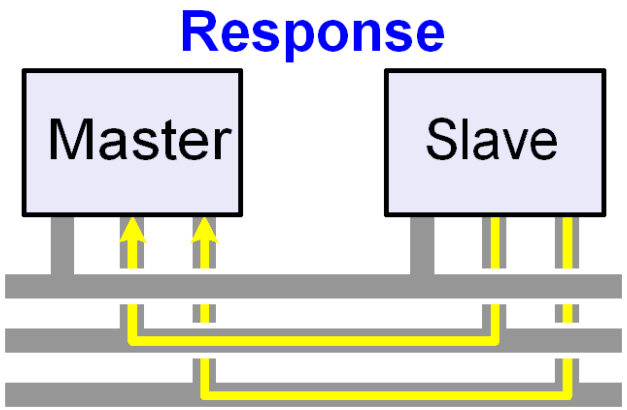
D 8000M Bytes/sec

Bus Operation

To transfer data across the bus, the Bus Master sends the request to the slave by asserting certain control signals and sending the relevant address. The addressed Slave responds with an acknowledgement and the requested data.

The timing of the various signals must be coordinated between the master and the slave so that each party can process them properly. There are two ways to coordinate the various signals on the bus:





Synchronous

In a synchronous bus, the timing of the signals is synchronized to a master clock signal. All devices are expected to send and receive signals based on an established pattern relative to the clock

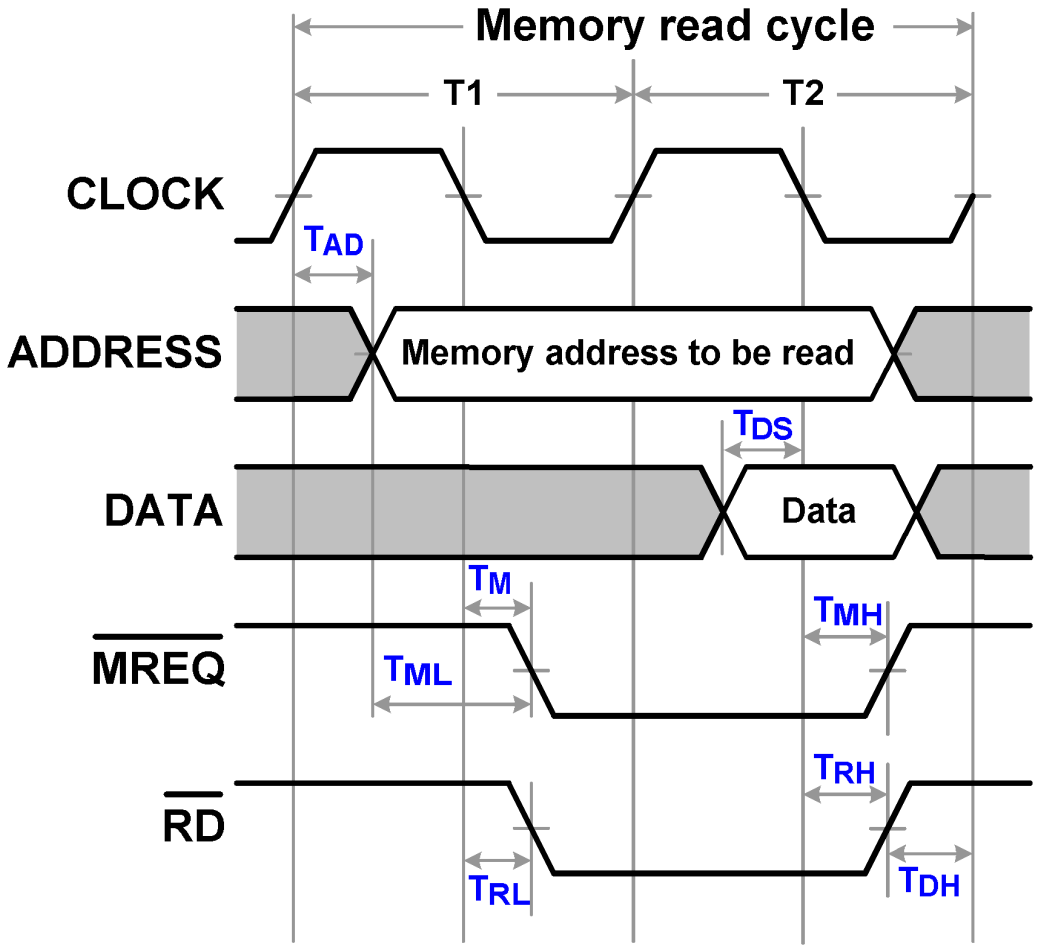
Asynchronous

In an asynchronous bus, the bus signals can be sent at any time without regard to the clock. Separate bus signals are used to synchronize the master and slave by indicating when each has finished processing the request.

Synchronous Bus Example

As an example of how a synchronous bus works, let's look at an example where the clock speed is 100MHz:

This timing diagram shows the bus signals during a memory read. Note the following points about this diagram:

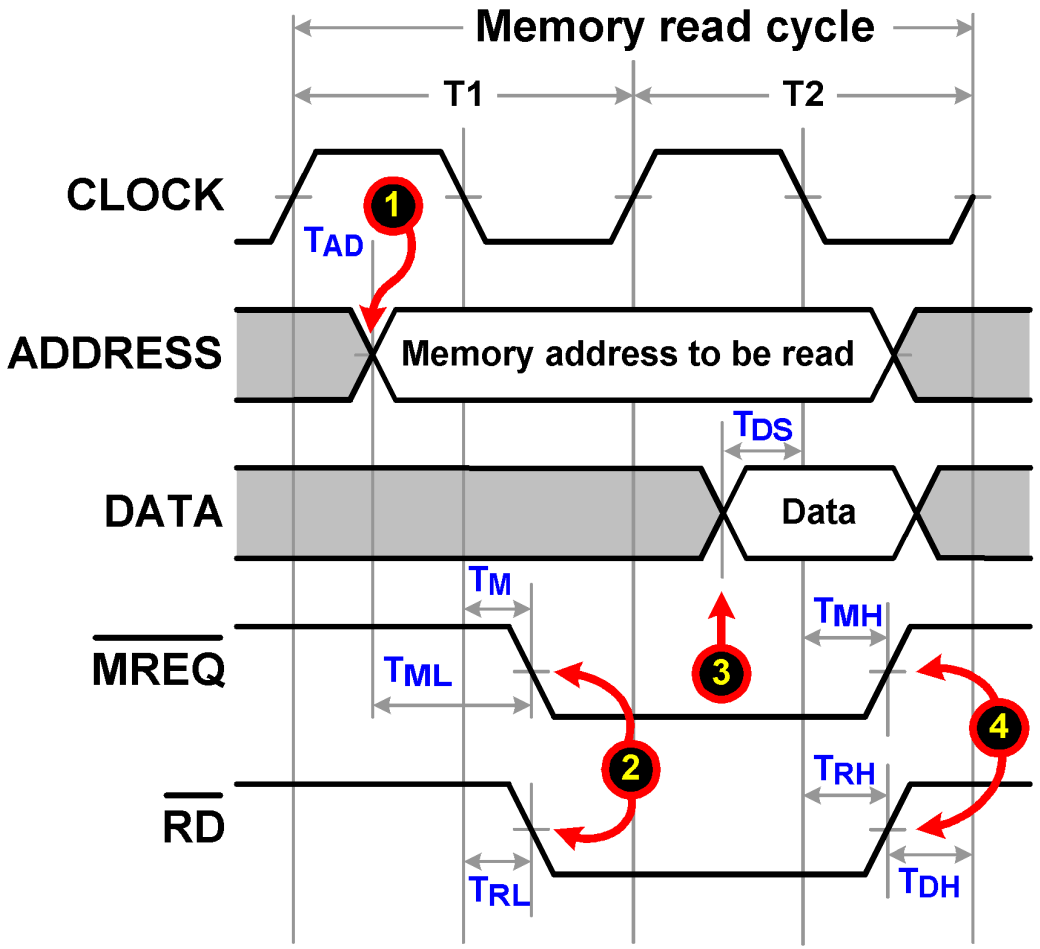


- **At 100MHz, each cycle (T1, T2, etc.) takes 10ns**
- **The rising and falling edge of each signal is shown as a diagonal line because it takes a finite amount of time for the signals to change.**
- **Address and Data signals are shown as "envelopes" because some signals can be TRUE and some can be FALSE. The unshaded areas show where valid signals are present.**
- **The Address, MREQ and RD signals are sent from the Master to the Slave, and the Slave responds by sending the data back to the Master.**

Synchronous Bus Example

Here's the sequence of events that takes place for the bus read cycle:

1. **Master asserts address signals**
2. **The Master waits for about ½ clock cycle so that the address signals can select the correct Slave device, then it asserts the MREQ and RD signals.**
3. **The Slave delivers the requested data over the data lines.**
4. **The Master removes the MREQ and RD signals.**



The timing of the signals is governed by the clock. Each signal must be sent and received at a specific time in order for the transaction to work correctly.

The timing requirements for the read transaction are shown in blue on the diagram. The bus specification states the minimum or maximum times that these are allowed to take. The table on the next page shows what the values are for this example.

Synchronous Bus Example

The timing specifications for our example bus are as follows:

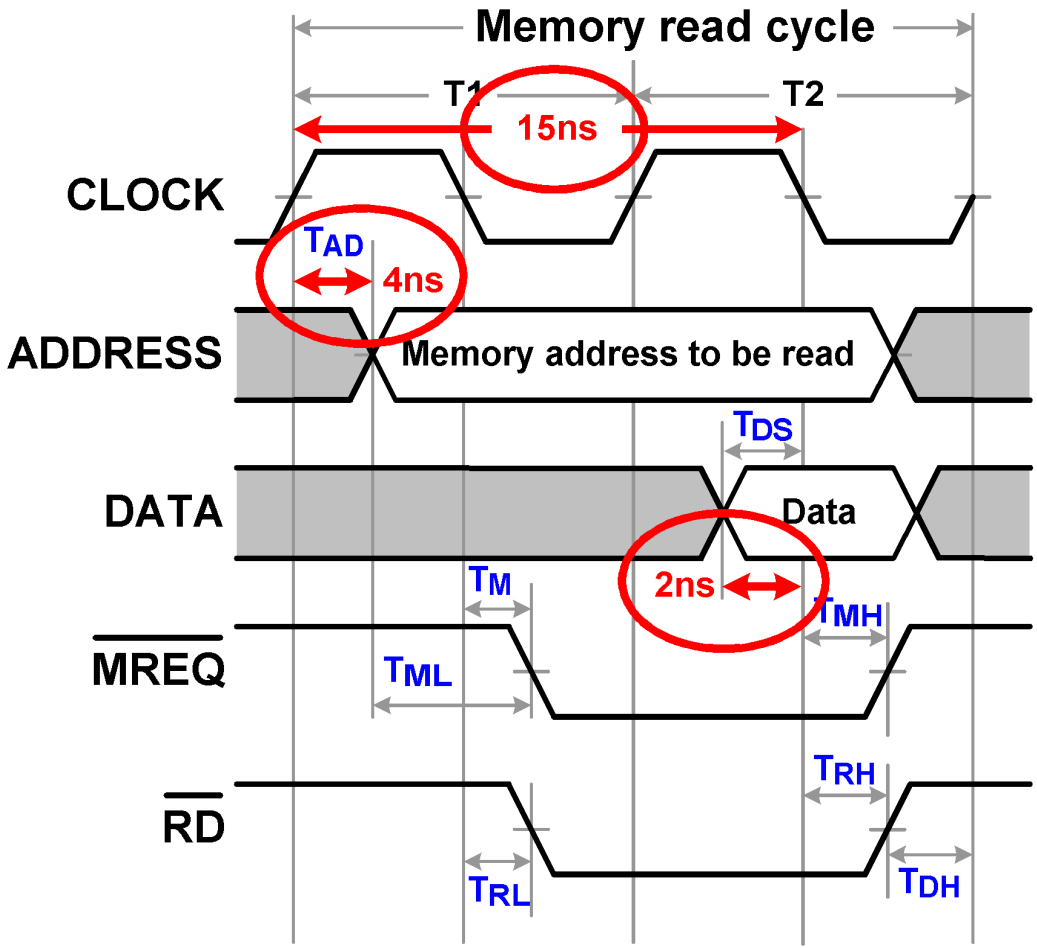
Symbol	Parameter	Min	Max	Units
T_{AD}	Address Output Delay		4	ns
T_{ML}	Address stable prior to MREQ	2		ns
T_{M}	MREQ delay from falling edge of clock in T1		3	ns
T_{RL}	RD delay from falling edge of clock in T1		3	ns
T_{DS}	Data setup time prior to falling edge of clock	2		ns
T_{MH}	MREQ delay from falling edge of clock in T3		3	ns
T_{RH}	RD delay from falling edge of clock in T3		3	ns
T_{DH}	Data hold time from negation of RD	0		ns

These times come from the timing requirements of the various circuits. For example, T_{ML} is at least 2ns because a valid, stable address must reach the circuits that decode which slave device to select before the MREQ signal reaches them – otherwise the decoder may briefly activate the wrong device.

Synchronous Bus Example

In our example bus read transaction, the timing requirements are as follows:

- **The data must be delivered by the middle of the 2nd clock cycle. Since each cycle takes 10ns, this gives an overall time of 15ns from the start of the request to the time the data is delivered.**
- **The address may be sent up to 4ns after the start of the 1st clock cycle.**
- **The data must be delivered no later than 2ns before the middle of the 2nd clock cycle.**



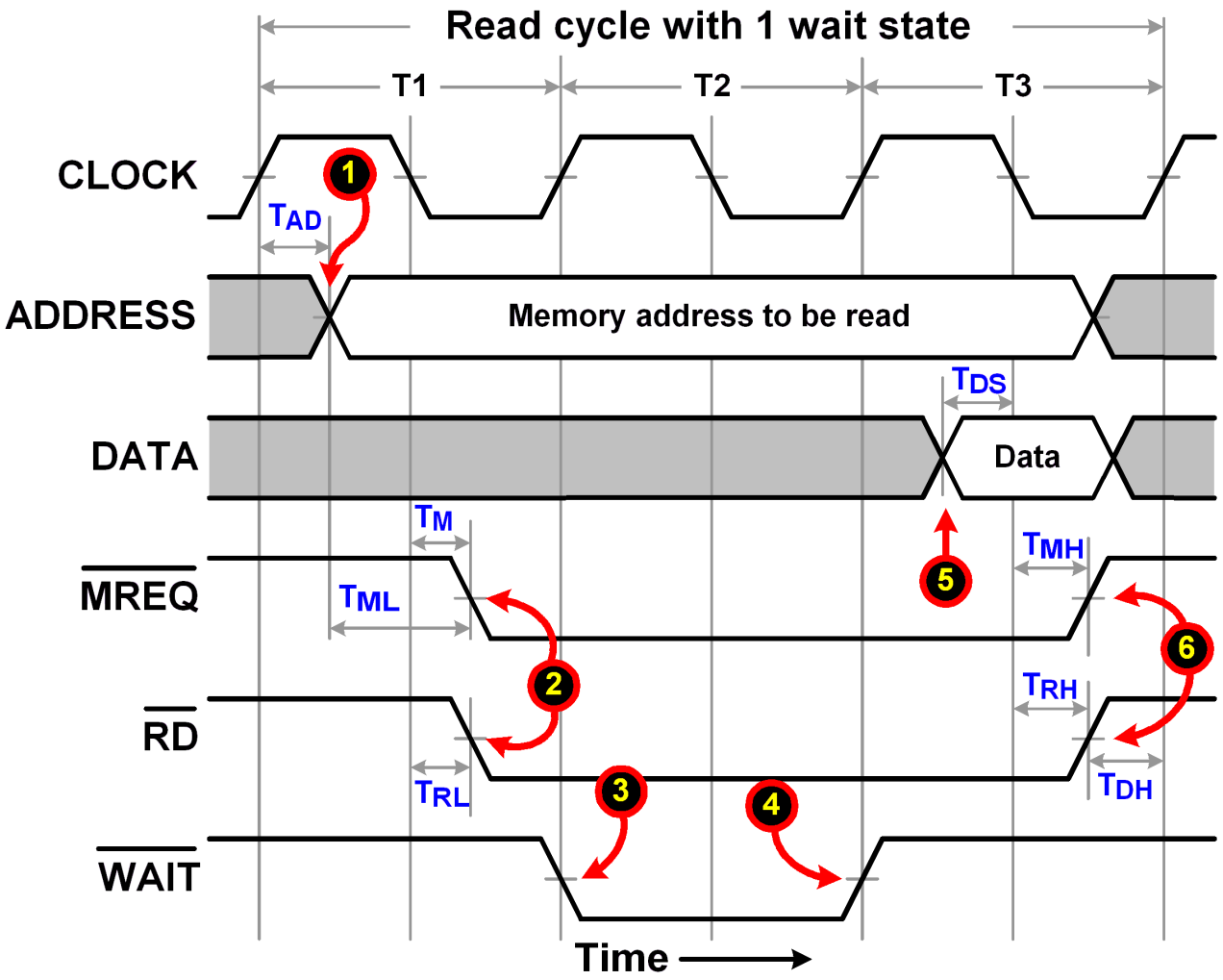
These limitations mean that the Slave device must be able to provide the correct data within 9ns of receiving the correct address (15ns – 4ns – 2ns).

If the Slave can't provide the data quickly enough, it must tell the Master to wait for one or more additional clock cycles before reading the data from the bus. This is done using another control line called "WAIT" as shown on the next page.

Synchronous Bus Example

When the Slave can't respond quickly enough, it extends the read cycle using a WAIT signal.

1. **Master sends the address over the bus.**
2. **The Master asserts the MREQ and RD signals.**
3. **The Slave asserts the WAIT signal, which tells the master not to expect data during this cycle.**
4. **The Slave terminates the WAIT signal at the start of the clock cycle in which the data will be returned.**
5. **The Slave delivers the data.**
6. **The Master removes the MREQ and RD signals.**



In this example the read transaction has been extended by one extra clock cycle and now takes a total of 3 clock cycles to complete. If the Slave is slower still, then it can keep WAIT asserted for several clock cycles.

Synchronous Bus Example

When the Slave assert the WAIT signal to extend the read transaction by one clock cycle, the timing restrictions mean that the Slave now has up to 19ns between the time the address is received and the data is returned:

The time from the start of the first clock cycle to the middle of the third clock cycle is:

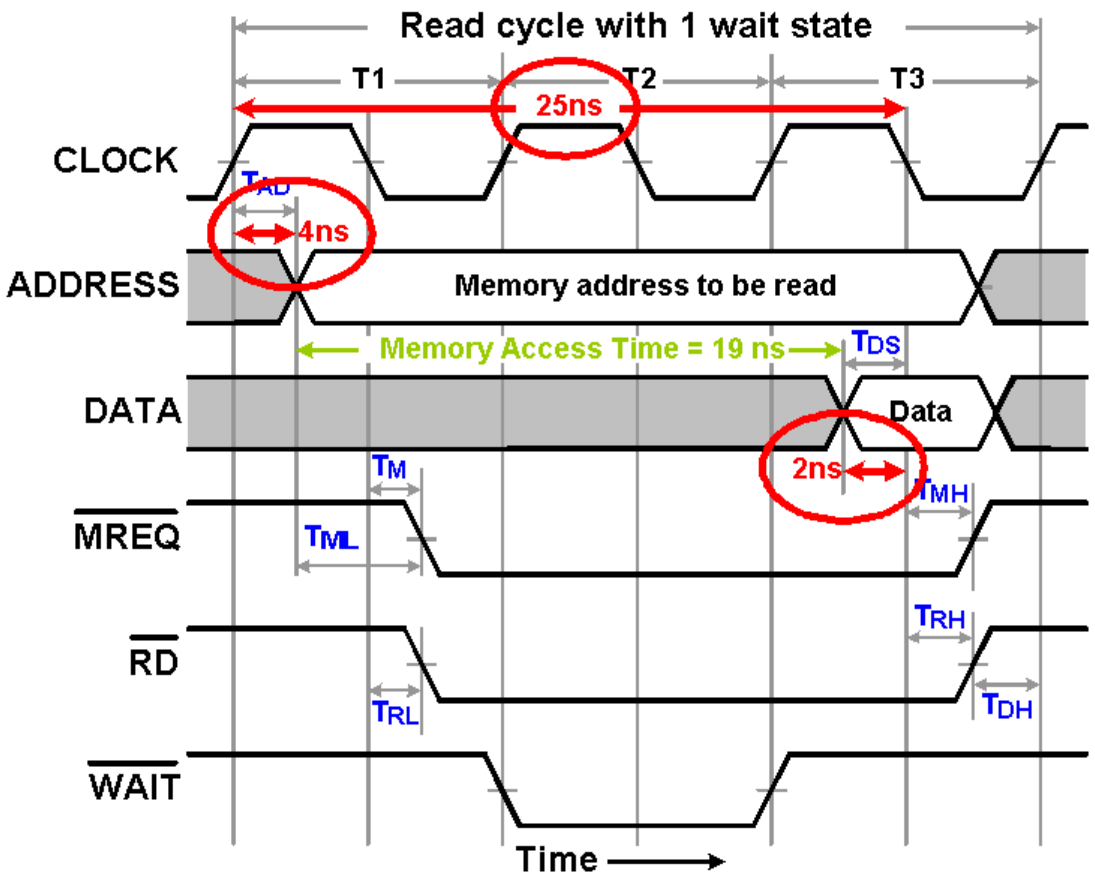
10ns/cycle x 2.5cycles = 25ns

This allows us a memory access time of as much as:

25ns – 4ns – 2ns = 19ns

The example from page 180 of the textbook uses memory with an access time of 15ns. That memory would be able to return the data quickly enough with one wait cycle as shown here.

But if the memory access time was longer than 19ns, then additional wait cycles would have to be added.

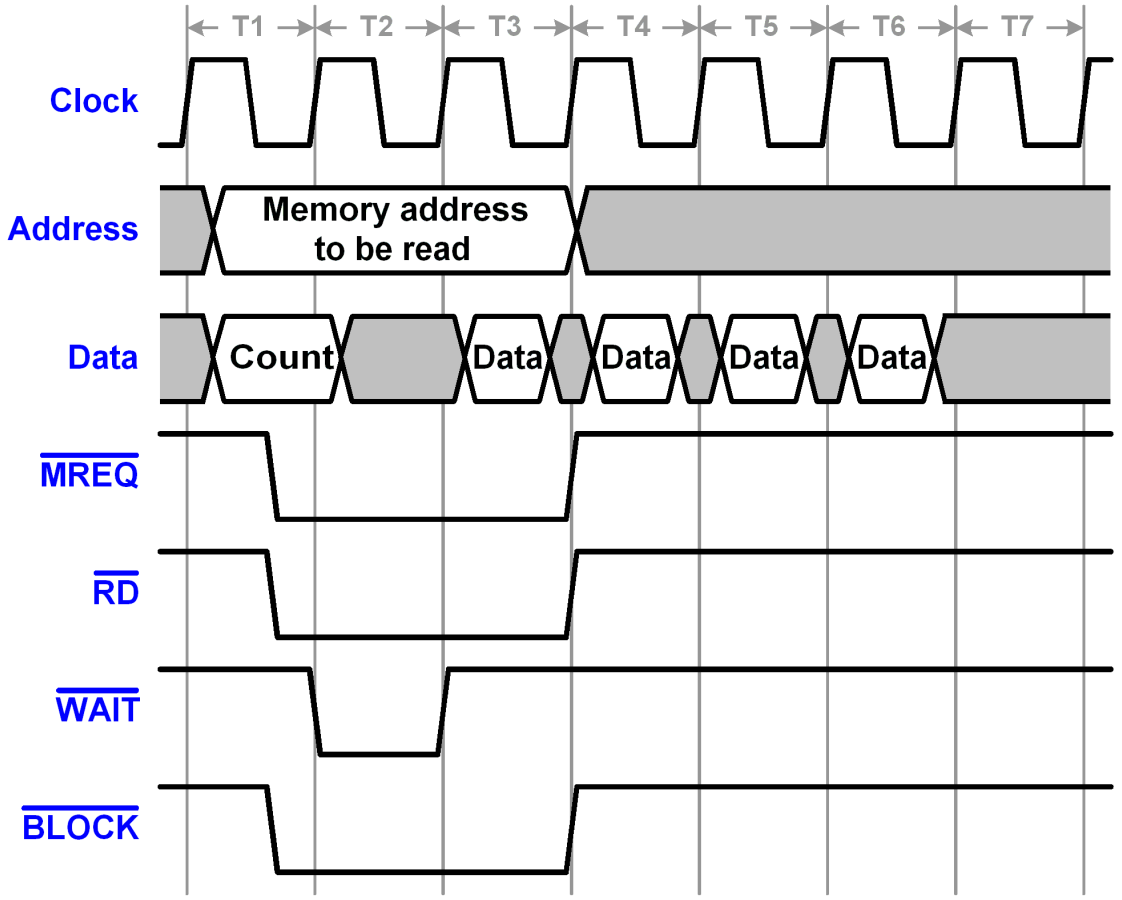


Block Transfers

Many buses support a "Block Transfer" mode (also known as "Burst Mode") to make more efficient use of the bus for transferring more than one item of data at a time. This is often used for transferring cache lines from memory to the CPU.

This is how it works:

- **The CPU sends the starting address over the address lines and a count of the number of transfers requested over the data lines.**
- **The CPU also asserts a special "BLOCK" signal so the memory will recognize this as a request to transfer data from several successive memory locations.**
- **The memory responds by sending data from successive memory locations, one per clock cycle, for as many cycles as was requested by the "count".**



The memory doesn't need to assert WAIT for the subsequent data because it has already decoded the row address and it is faster to get data from successive columns.

Exercise 3 – Block Transfer Performance

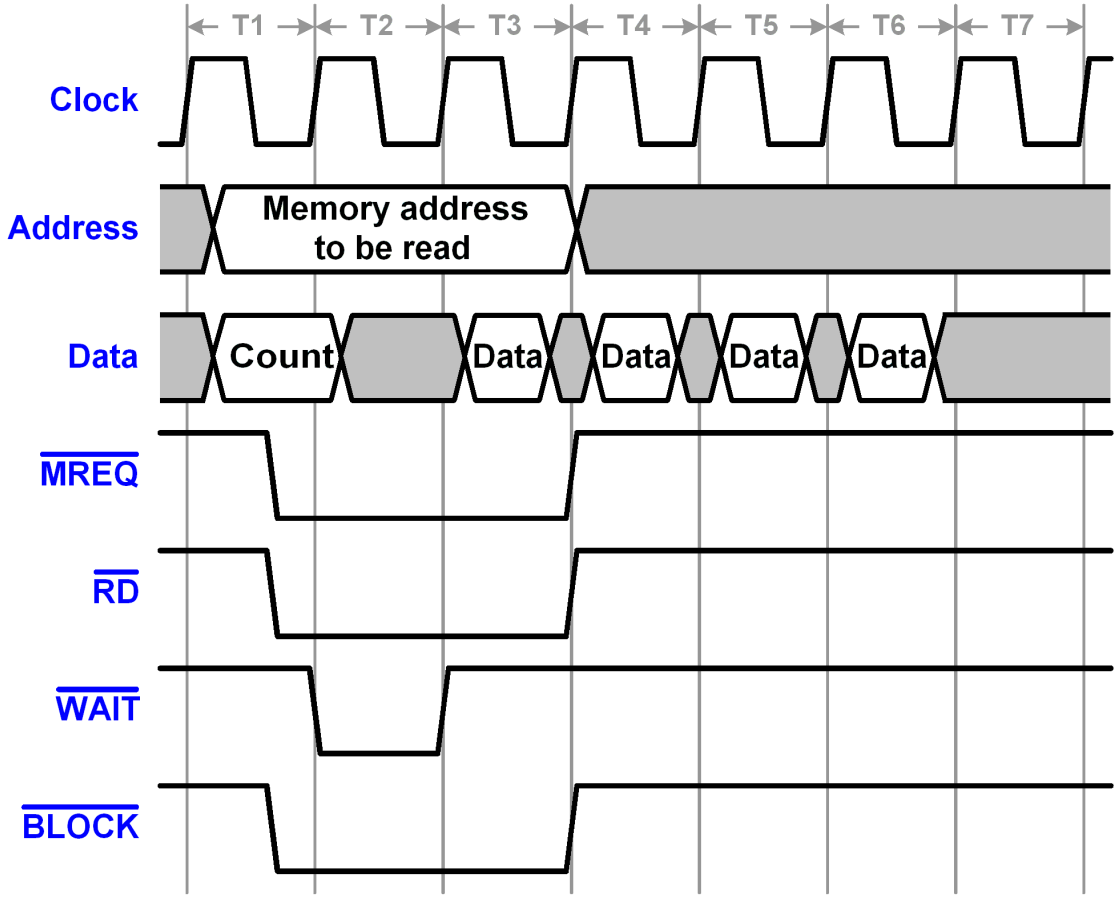
A bus has 32 data lines and uses a 100MHz clock. How long will it take the bus to perform a 500-byte block transfer?

A 1270ns

B 120ns

C 502ns

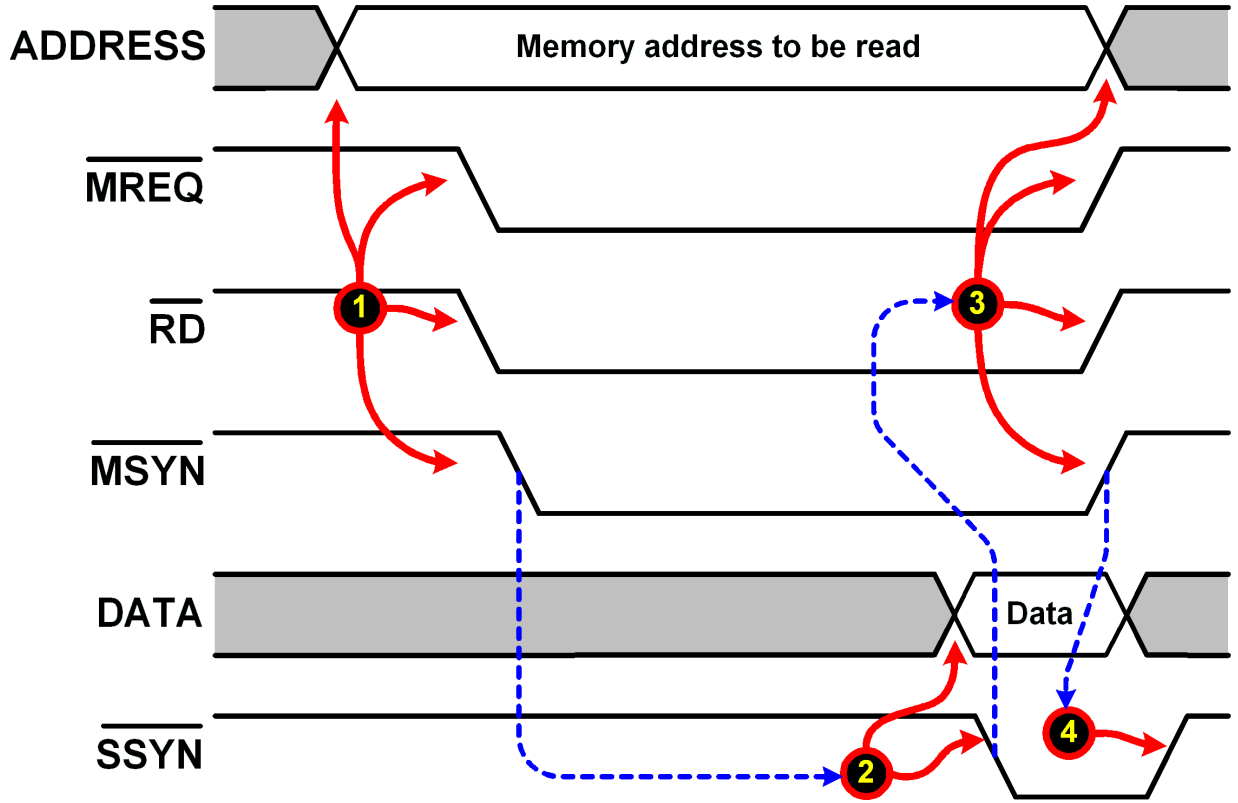
D 5020ns



Asynchronous Bus

An Asynchronous Bus doesn't use clock signals. Instead, MSYN (Master Synchronization) and SSYN (Slave Synchronization) signals coordinate the request and response:

1. **The master asserts the address and then the MREQ, RD and MSYN signals. The MSYN signal tells the slave that the signals it needs are available on the bus.**
2. **The slave accesses the requested data. When the data is available, it puts the data on the bus and asserts the SSYN signal. SSYN tells the master that the response it is waiting for is available on the bus.**



3. **The master sees SSYN and accepts the data from the bus and then removes the address, MREQ, RD and MSYN signals. When MSYN is deasserted the slave knows that the master is finished reading the data from the bus.**
4. **The slave sees that MSYN deasserted and removes the data and SSYN signals from the bus.**

This type of coordinated signal exchange is sometimes called a "handshake".

Synchronous vs. Asynchronous Buses

There are advantages to both Synchronous Buses and Asynchronous Buses:

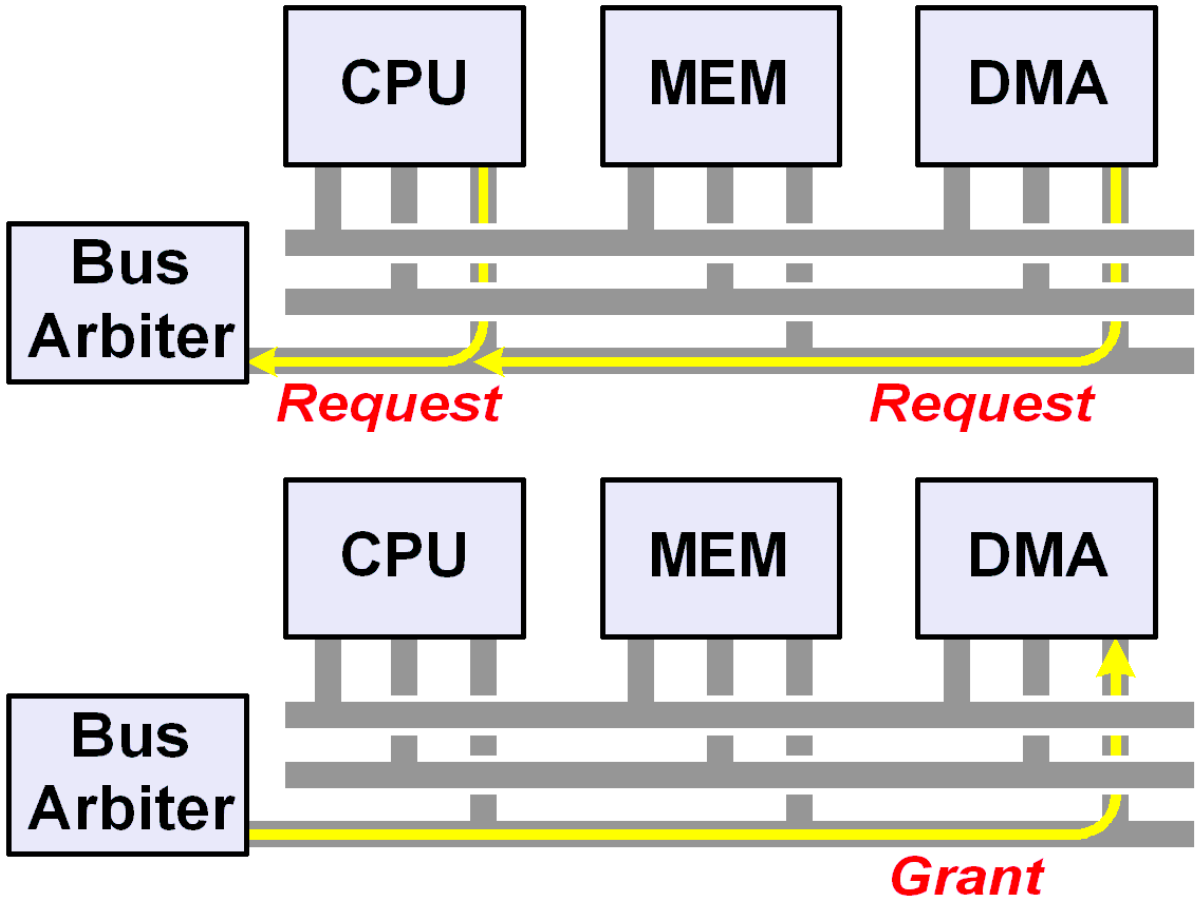
Synchronous Bus Advantages	Asynchronous Bus Advantages
Despite the fact that the timing diagram for a synchronous bus looks more complex, it's actually easier to design the interface circuitry. Everything on the bus happens in conjunction with a clock pulse, and the CPU circuitry already uses the clock pulse to synchronize internal operations.	The bus is not dependent on the access time for any particular circuit. Each portion of the bus transaction happens as a result of a previous step and does not have to take place at a change in the clock signal. If one particular slave is slower than another, then the bus transaction automatically takes longer. If slow memory is replaced by faster memory, the system will automatically speed up because the faster memory responds to the bus request earlier.

In practice most buses are Synchronous because the circuitry is so much easier to design and build.

Bus Arbitration

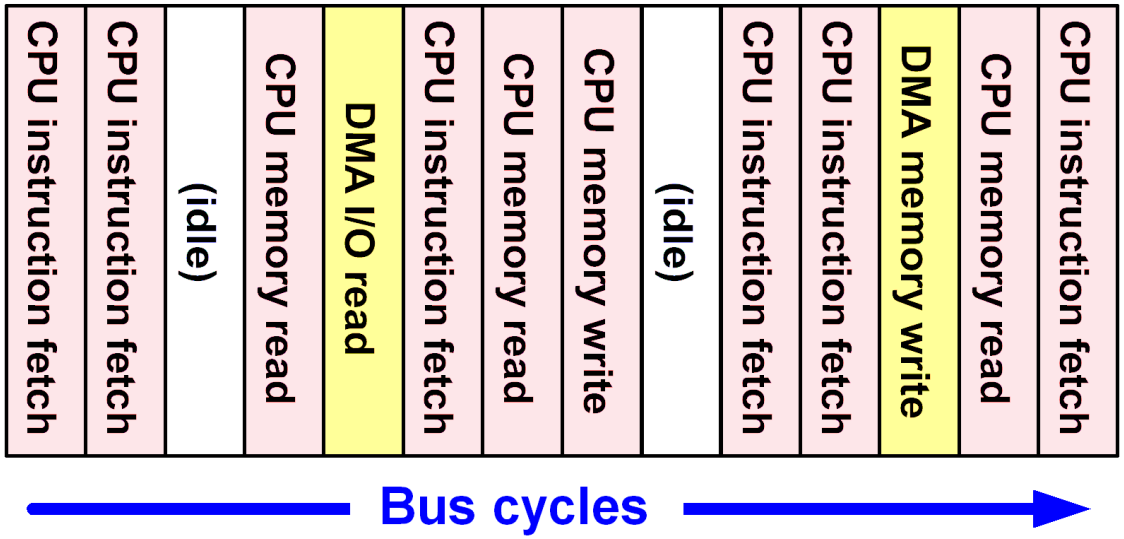
The bus is can only be controlled by one master at a time, because there is only one set of wires on the bus. If two devices both want to become the bus master, there needs to be a way to decide which one takes control. The process of deciding who will become master is called Bus Arbitration. It works like this:

- **When a device wants to take control of the bus and become the master, it sends a special control signal called "Bus Request" to the bus arbiter.**
- **The bus arbiter sees which devices are requesting the bus and decides which one should get the bus.**
- **The bus arbiter sends a special "Bus Grant" control signal to the device it has chosen to let it know that it can proceed.**
- **Devices which don't receive the "Bus Grant" signal must wait until their turn.**



Cycle Stealing

Bus Arbitration allows devices other than the CPU to take control of the system bus when they need to. When the bus is granted to one of these other devices, it's called "cycle stealing", since the CPU is blocked from accessing the bus during those cycles.



For example, if a disk drive transfers 10MB/sec on a system bus with a bandwidth of 100MB/sec, then the disk drive is using up 10% of the bandwidth of the bus.

Note – this wouldn't necessarily slow down the CPU by 10% since the CPU doesn't use all of the other 90% of the bandwidth anyway. It's also often possible to be able to preempt a few bus cycles while the CPU continues executing instructions using cached information.

Interlocked Bus Cycles

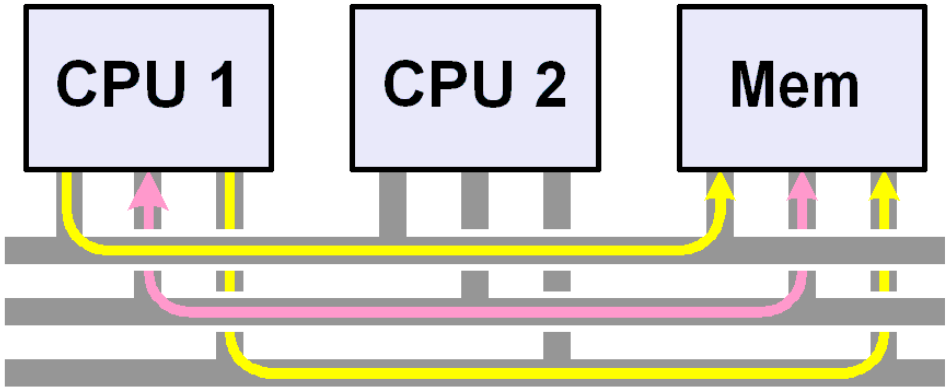
In a multi-CPU system the software often needs to coordinate the actions of the two CPUs. The only way to do this is to use special "lock" variables which are stored in a shared memory that both CPUs can access.

A typical lock variable would be a byte which contains "0" if a particular data structure is free, or "1" if it is in use (locked). A CPU which wanted to "acquire the lock" would:

- **Read the variable. If it is "1" then the data is "locked" and this CPU must wait for the other CPU to release it.**
- **If the variable is "0", the data is available. This CPU writes a "1" to the data to show that it's locked.**

The problem with this scheme is that if both CPUs try to acquire the lock at the same then it's possible that they will both read the "0" value before either one can change it to "1". In that case they would both believe the data was available and would try to access it at the same time.

To avoid this, many buses support an "interlocked read/modify/write" cycle. This allows a CPU to read the data, change it, and write it back to memory with no possibility that the other CPU will gain control of the bus.



Interrupt Handing

Another function the system must handling is that of Interrupt Handling. An interrupt is a signal that is generated by an I/O device, typically to indicate it is ready for input or output.

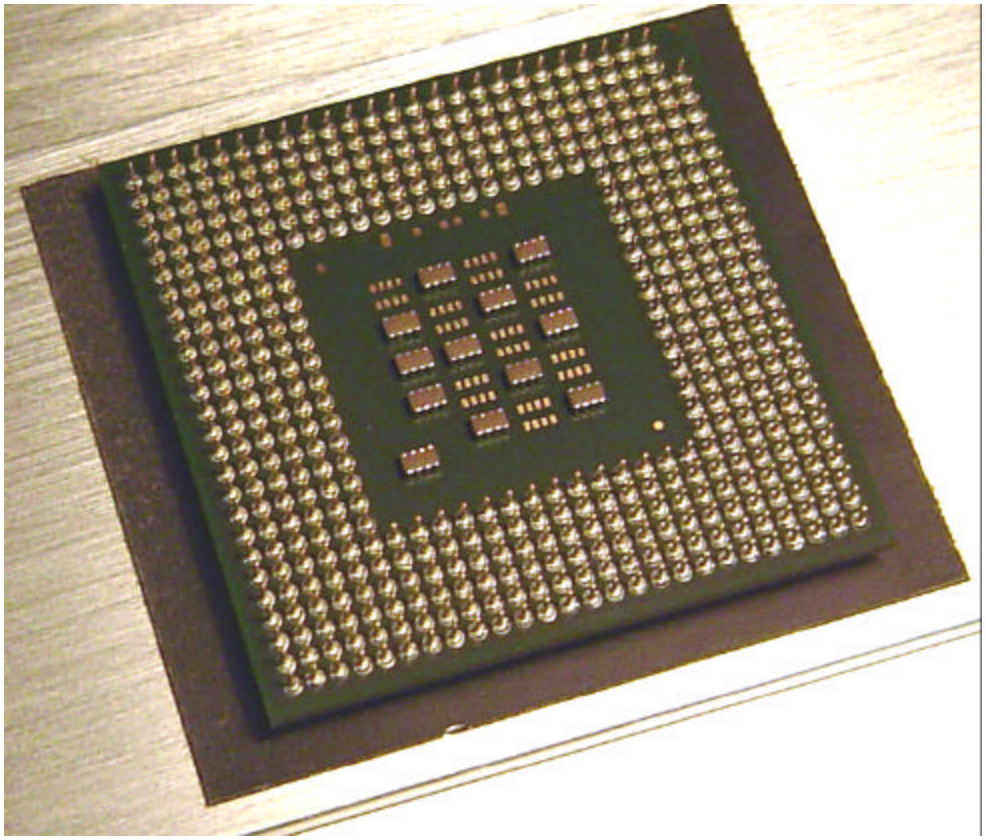
Different devices are connected to different Interrupt Request Lines. When the CPU receives an interrupt, it performs the following actions:

- **Suspends the currently executing program**
- **Uses the number of the Interrupt Request Line to look up the address of an "Interrupt Service Routine" (ISR) in an "Interrupt Vector Table" in memory. The ISR is a subroutine which performs the processing needed to handle the interrupt.**
- **Calls the selected Interrupt Service Routine.**
- **When the ISR is finished, it returns back to the interrupted program and the CPU sends an "Interrupt Acknowledge" signal back to the device to indicate that the interrupt handling has been completed.**

Example CPU Chips – The Pentium-4

The Pentium-4 is one of Intel's series of general-purpose CPUs which the textbook uses as an example of a CISC microprocessor. This CPU makes a good study as a CISC microarchitecture.

- **55 million transistors**
- **Backwards compatibility from 8088 used in original IBM PC**
- **64GB of addressable memory**
- **Supports two CPUs in a dual-processor system**
- **478-pin packaged includes 85 pins for power and 180 for grounding to reduce noise**
- **On-chip L1 and L2 caches (size depends on model)**
- **Package designed to dissipate as much as 82 watts of heat through a metal backplate.**

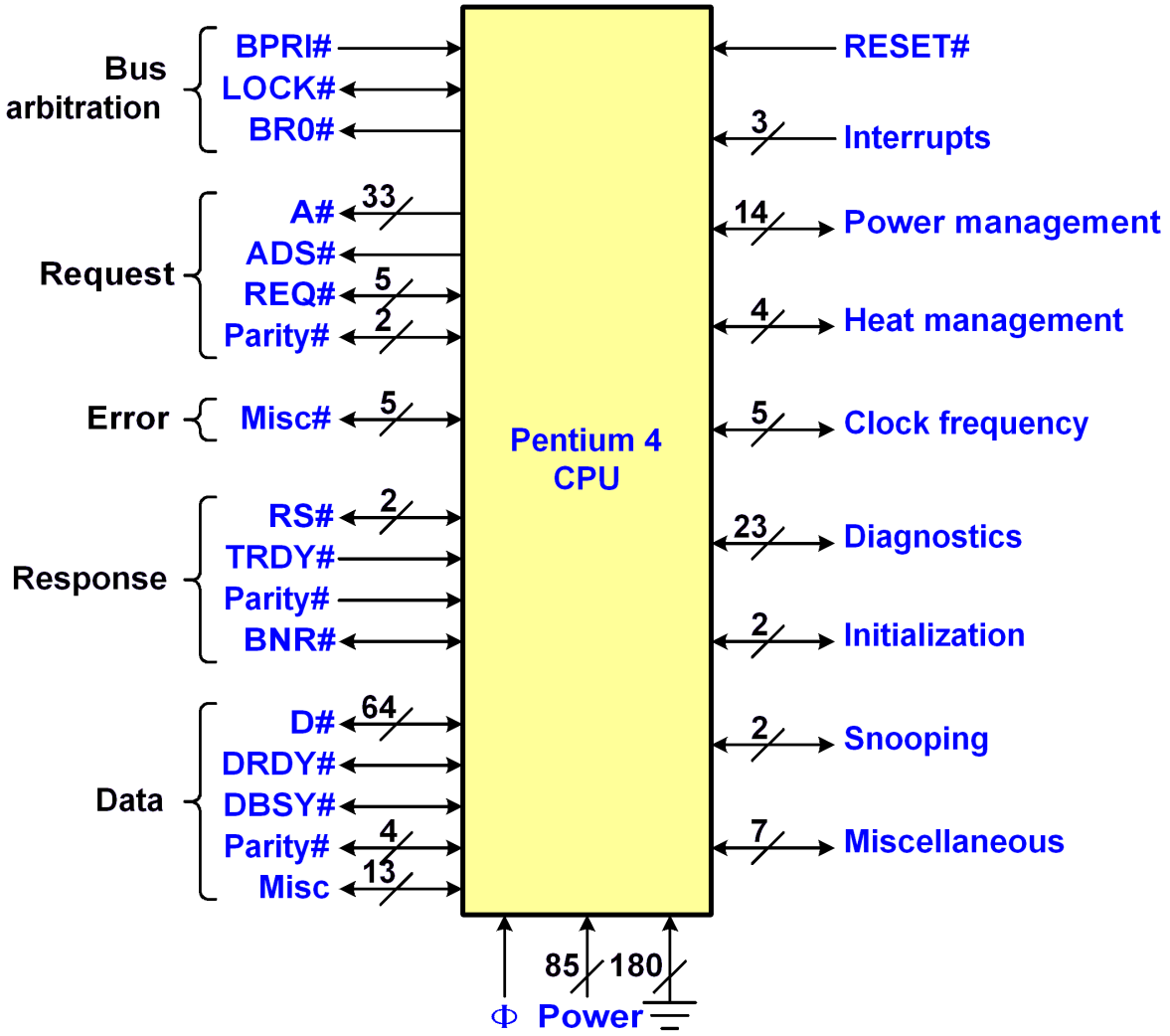


Pentium-4 Logical Pinout

This diagram shows the logical connections for the Pentium-4 CPU

Symbols with a "#" suffix indicate a negative logic signal. Intel uses this notation instead of an overbar because it's easier to use in circuit design software.

Note that these pins are not connected directly to the memory or the I/O bus – they go to a chipset which contains the necessary logic to translate these CPU signals to those required by the other components in the system.



Pentium-4 Bus Pipelining

The Pentium-4 uses a pipelined bus. Each bus transaction has six pipeline stages:

Bus Arbitration

The CPU requests and waits for ownership of the bus

Request

The CPU sends the address and the transaction type to the targeted bus slave

Error Reporting

Allows the slave to indicate an error (for example, a parity error)

Snoop

The CPU which is the bus master snoops a 2nd CPU to check for cache coherency

Response

The bus slave acknowledges the type of transfer that is about to take place

Data

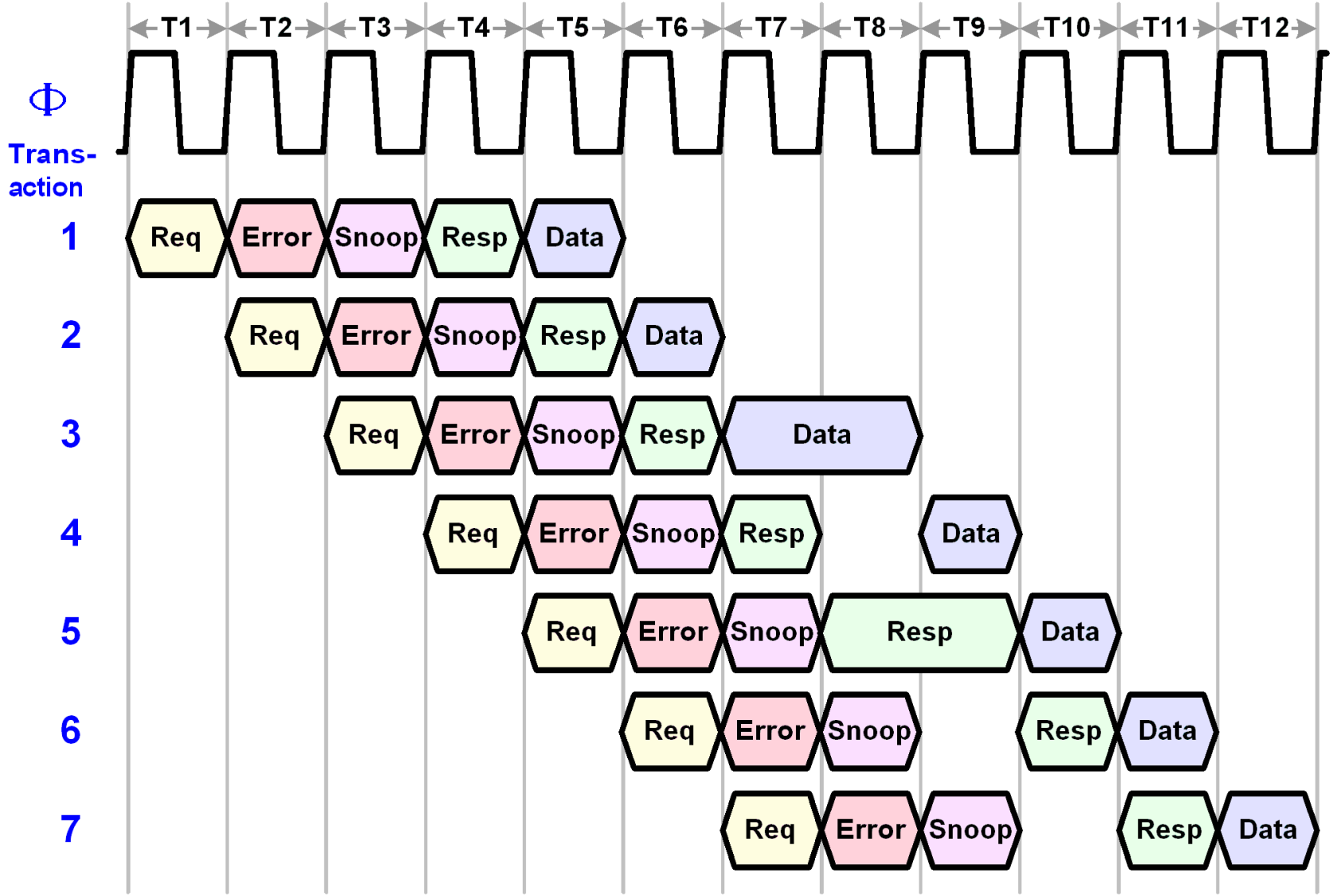
The data is transferred to or from the bus slave

Not all types of bus transactions require all six pipeline stages.

Each stage uses different sets of bus signals, so different stages can be in progress at the same time. This is how the Pentium-4 is able to pipeline it's bus transactions.

Pentium-4 Bus Pipelining

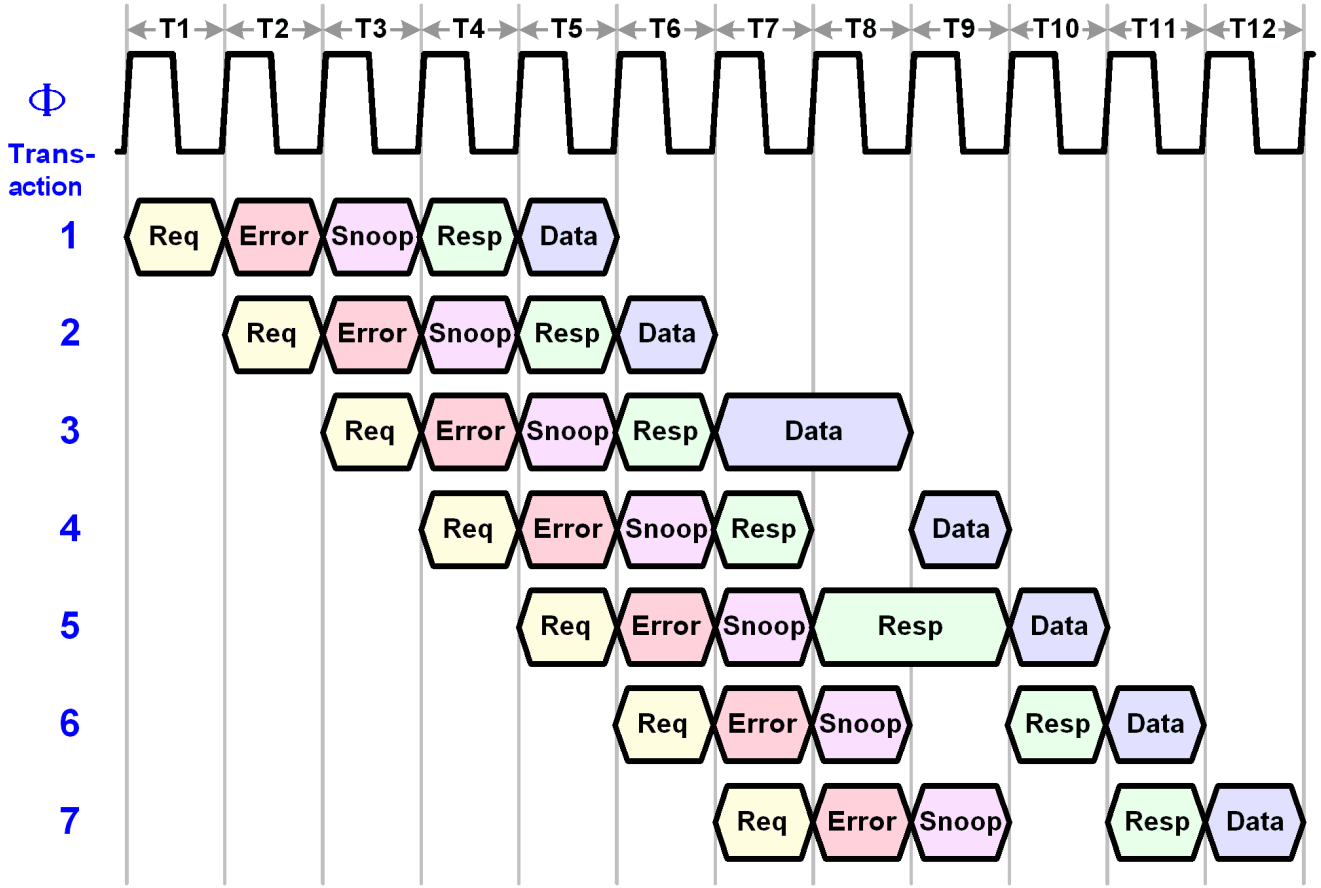
This diagram shows different transactions can be pipelined on the bus:



Pentium-4 Bus Pipelining

This diagram shows different transactions can be pipelined on the bus:

- **The bus arbitration phase is not shown because it's not always needed.**
- **In transaction 3, the memory inserted a wait state before the data was returned.**
- **Transaction 3 is still asserting DBSY# during T8, so transaction 4 has to wait until T9 for it's data transfer stage.**
- **Transaction 5 has an extended response phase, which can also delay the transaction**
- **The transaction 5 delay causes a "bubble" to be formed in the pipeline as each subsequent transaction's response phase has to wait for the previous transactions' to complete.**

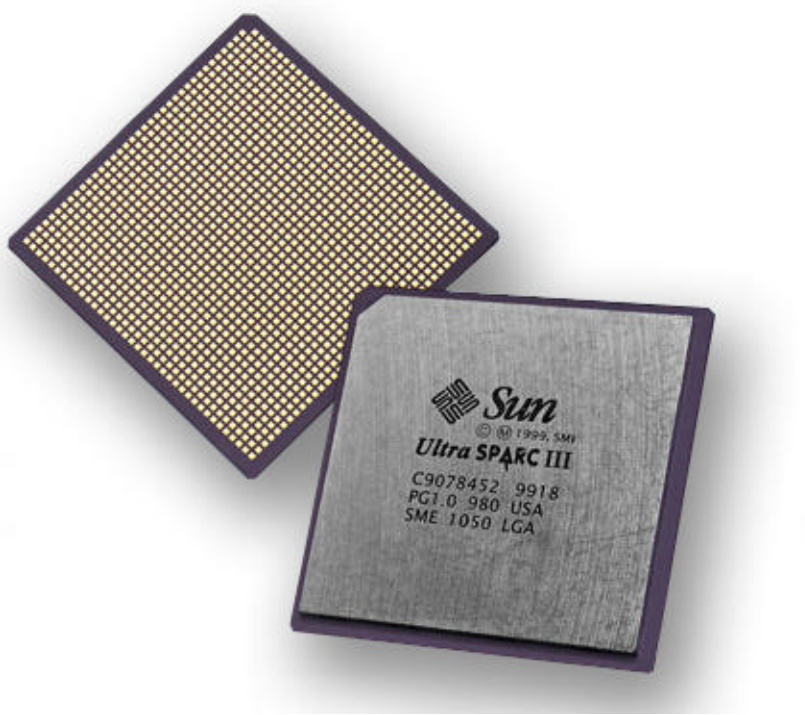


In actual practice, the CPU does not usually initiate new transactions on every bus cycle, so "bubbles" don't stay in the pipeline for very long.

Sample CPU Chips – The UltraSPARC-III

The UltraSPARC-III is the third model in Sun's series of general-purpose RISC CPUs. SPARC stands for Scalable Processor ARChitecture. Sun designed the UltraSPARC series of systems from the ground up, and this CPU makes a good case study of the principles behind RISC architectures.

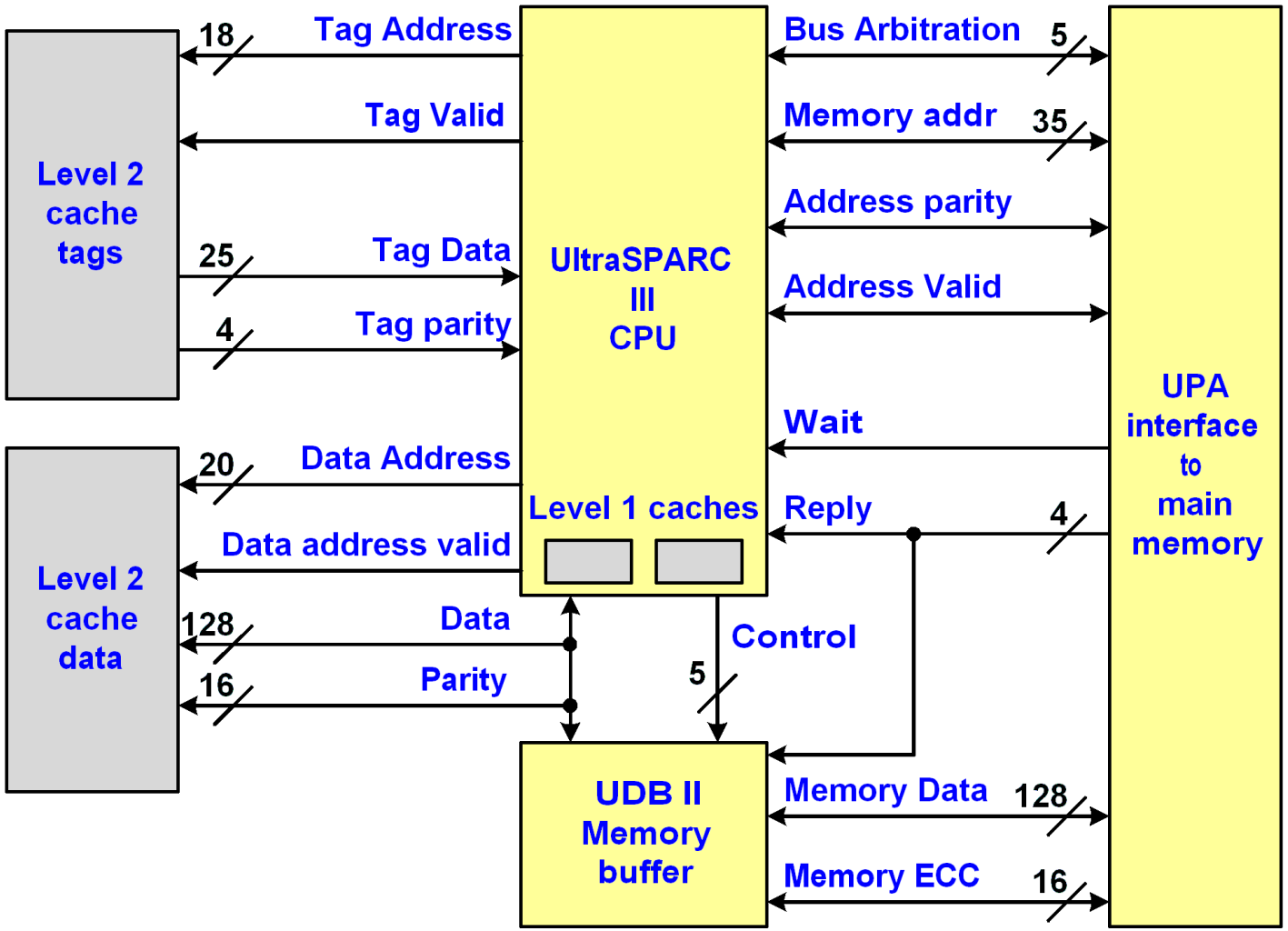
- **The UltraSPARC-III is a 64-bit CPU that maintains backward compatibility with earlier versions of the SPARC series**
- **Chip has 1368 pins and 29 million transistors**
- **Contains an internal split L1 cache (32K for instructions, 64K for data)**
- **L2 cache is external to the CPU chip allowing designers to match it to system requirements**
- **Includes VIS multimedia instructions (similar to Intel's MMX instructions)**
- **Supports up to 106 CPUs in a multiprocessor system**



- **Sun does not build it's own CPU chips. Instead, it designs the chip and then contracts outside semiconductor manufacturers to build them.**

Sample CPU Chips – The UltraSPARC-III

The UltraSPARC-III uses external chips to store cache tags and data (shown in grey at left). (Tags tell the cache which memory addresses the data in cache came from). This gives designers flexibility to build a high performance system with lots of cache or an inexpensive system with little cache.

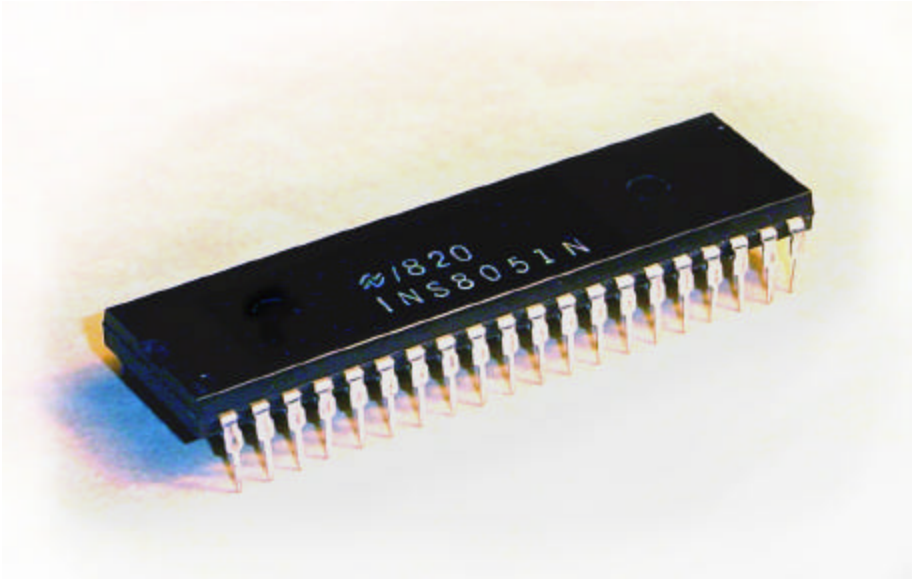


The UPA (Ultra Port Architecture) interface provides the connection to main memory.

The UDB-II (UltraSPARC Data Buffer II) chip allows memory requests to be buffered when the UPA interface is busy – this allows the CPU to go on to other tasks.

Sample CPU Chips – The 8051

The 8051 is one member in family of CPU chips that were designed for as an embedded system. Embedded computers are part of an electronic device which is not designed as a computer system. Examples of devices with embedded computers include VCRs, ovens, traffic lights, cars, sewing machines, etc.



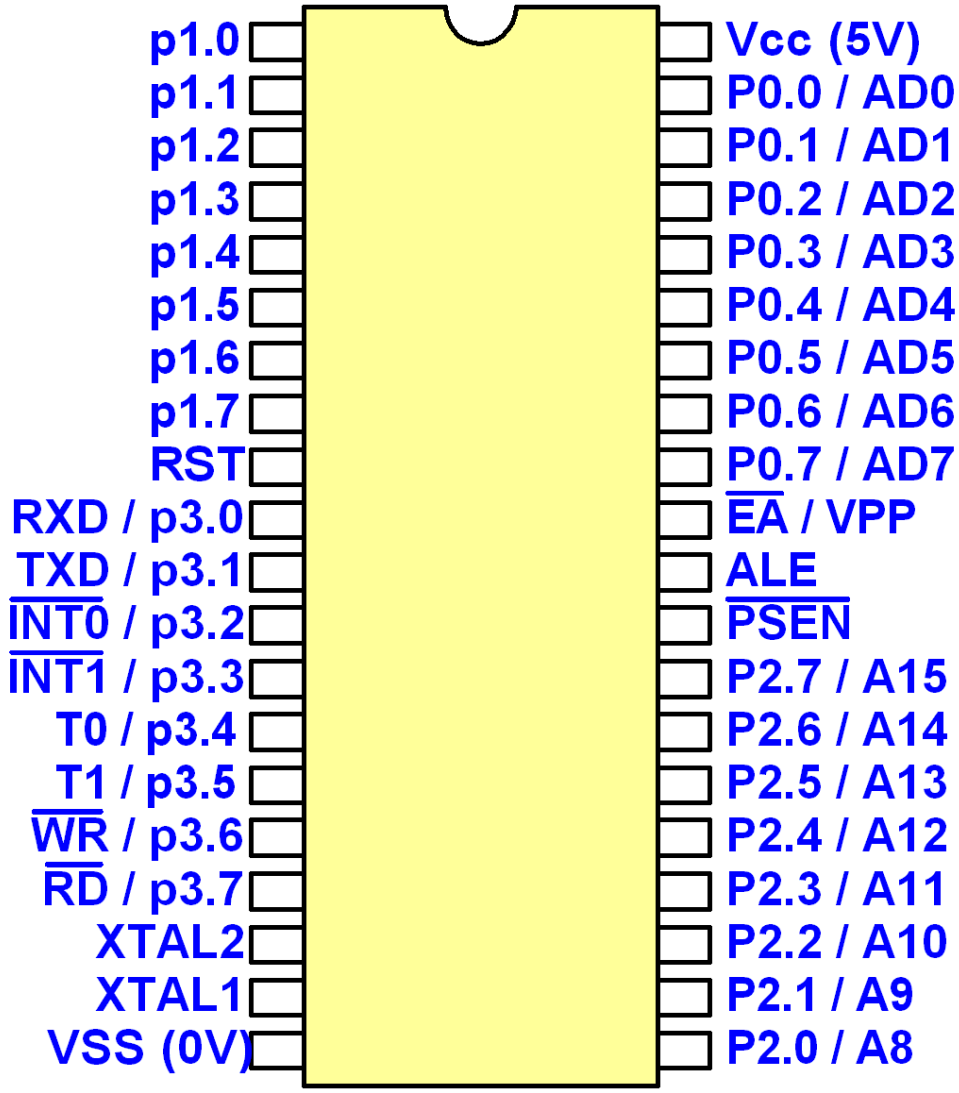
- **The chip family is widely available from several companies.**
- **Speeds range from around 12MHz to 100MHz depending on manufacturer and model – much slower than a typical general purpose CPU chip**
- **Cost is very low, as little as 10-15¢ per chip when bought in large quantities**
- **Chip includes built-in ROM and RAM memory which eliminates the need to connect external memory chips.**
- **ROM style varies according to model – models designed for development have programmable ROM, while models designed for production use typically use mask-programmed ROM.**
- **Chip also includes three I/O ports. Simple applications require no extra I/O chips to interface with lights, buttons, etc.**

Sample CPU Chips – The 8051

The 8051 comes in a 40-pin package with connections as shown in the diagram of it's physical pinout at right.

Many of the pins are multiplexed – the same pin is used for two different functions. For example, the P2.0 / A8 pin at lower right doubles as both the "A8" address line and I/O port 2 bit 0. The data lines are also multiplexed with the low-order 8 address lines.

Although the chip includes internal memory and several I/O port lines, it also provides traditional address and data lines so that additional external memory or I/O port chips can be added.



Sample CPU Chips – The 8051

The logical pinout of the 8051 is shown here. It includes the following signals:

ALE – Address Latch Enable is asserted when a valid address is being output on the address pins so that external circuitry can latch it. Used to distinguish between addresses and I/O or other data on the multiplexed pins.

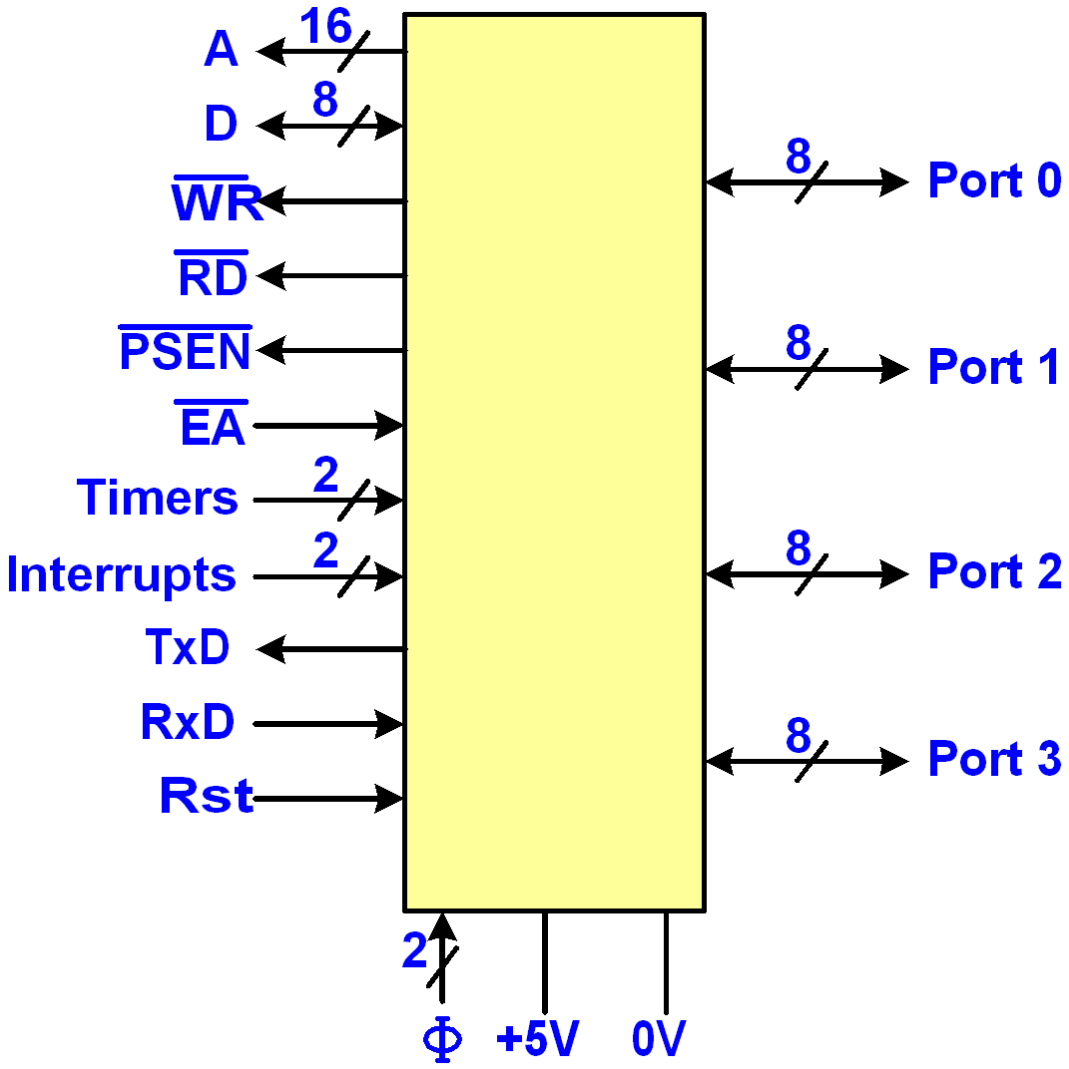
PSEN – Program Store Enable is asserted when the 8051 wants to read from external program memory.

EA – External Access signal is sent as LOW to allow external memory to replace the internal 4K of RAM.

Timers / Interrupts – allow external events to trigger program actions.

TxD / RxD – Serial transmit and receive lines that can connect to a modem

RST – resets the CPU



Sample Buses – the PCI Bus

The PCI ("Peripheral Component Interconnect") bus was developed by Intel in the early 1990s as a way to ensure that computer systems could take advantage of Intel's newest CPUs.

Intel encouraged the use of the bus by putting it into the public domain and making special chips that vendors could use to easily build PCI-compatible devices. Because of this PCI has become overwhelmingly popular and is now found in the vast majority of computer systems.

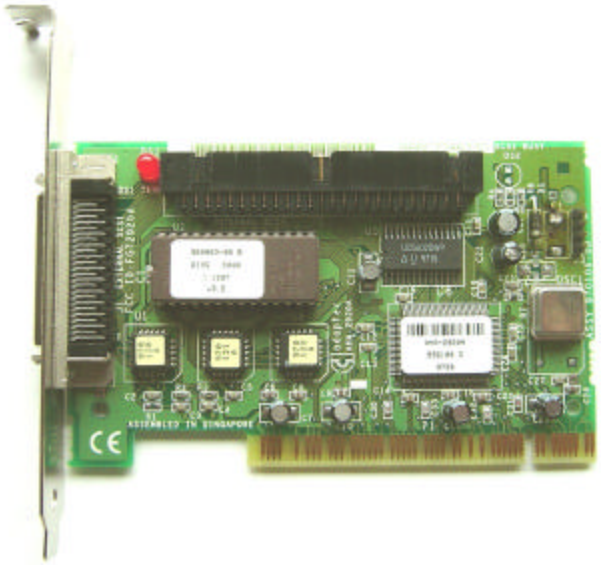
The original PCI bus was a synchronous bus that carried 32 or 64 address and data lines and operated at 33MHz or 66MHz. That gave the bus a throughput of from 132 to 528 Mbytes/sec (4 bytes / transfer x 33MHz)

Later versions of the bus were introduced, and were known as "PCI-Extended" (PCI-X). They used lower voltages (3.3 or 1.5V), faster clock speeds (266MHz or 533MHz) to give it an overall transfer rate of as much as 2GBytes/sec.

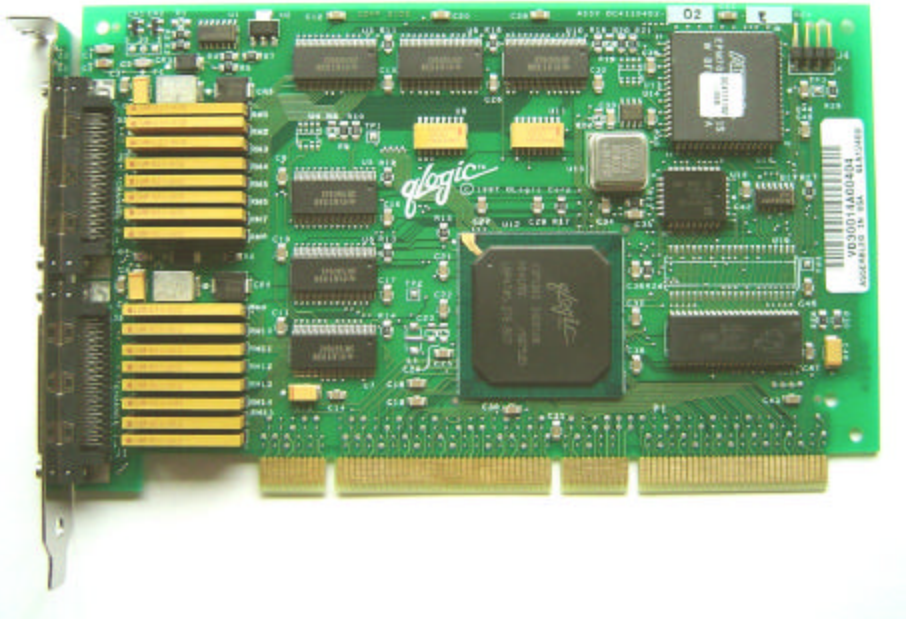
NOTE – Do not confuse "PCI Extended" (PCI-X) with "PCI Express" (PCIe)

Sample Buses – the PCI Bus

Here are some examples of PCI plug-in cards:



A 32-bit PCI Card



A 64-bit PCI Card

Note the following:

- **The 64-bit card has extra connectors on the left side – these carry the additional data and control signals for the wider bus.**
- **There are "notches" in the row of connectors, and the 64-bit card has an extra "notch" in the 32-bit group of connectors near the left end. These notches are used to ensure that 3.3V-only cards can't be plugged into a 5V bus, or vice versa.**

The AGP Bus

High-performance video cards with large memories need more bandwidth than a standard PCI bus can supply. A special version of PCI called "AGP" ("Advanced Graphics Port") was used especially for video cards. Most desktop systems used to include an AGP slot

AGP has several differences from a standard PCI bus:

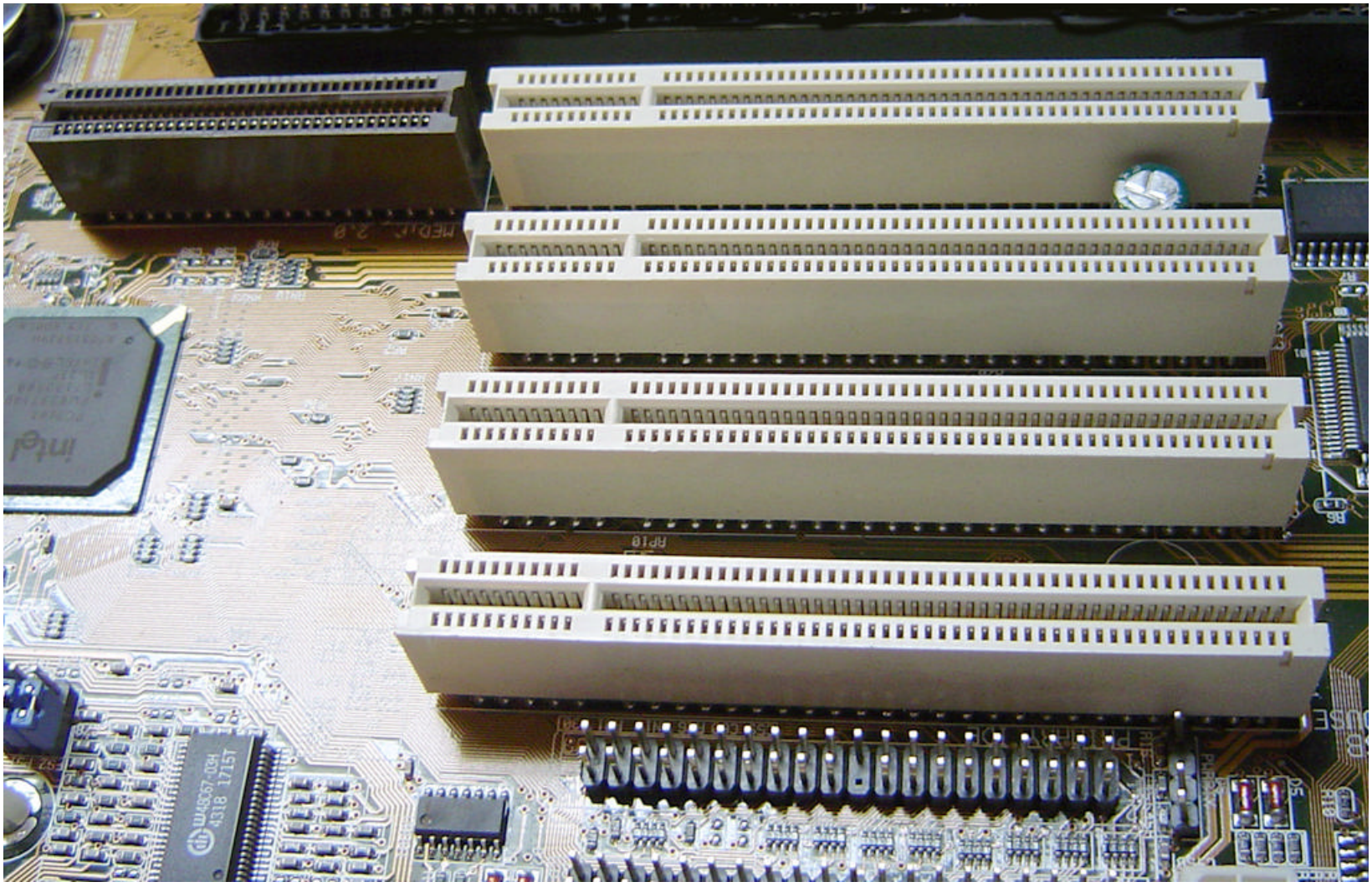
- **Address and data have their own separate signal paths and are not multiplexed**
- **Sideband signaling allows commands to be transmitted in parallel with data**
- **Bus transactions are pipelined**
- **Single target, single master operation**
- **Supports only memory read/write, no other I/O operations**
- **High/low priority queues**
- **Uses "double clocking" (synchronizes on both leading and trailing edge of the clock)**

The basic AGP data transfer rate is about 260Mbytes/sec. Faster versions include "2X", "4X" and "8X" which operate at multiples of the basic speed (for example, the bandwidth of a "AGP 8X" video card is about 2GBytes / sec.

In modern systems AGP has been replaced by PCI Express (discussed below)

Sample Buses – PCI and AGP

This picture shows a computer system with three 32-bit PCI bus slots and a fourth "AGP" slot (we'll describe what an "AGP" slot is later). The AGP slot is the one at the top.



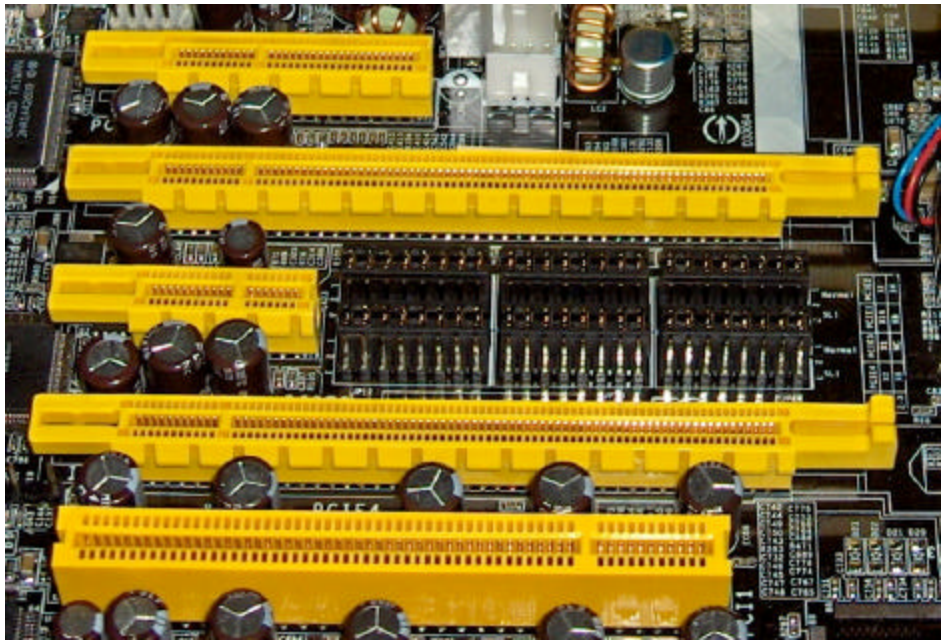
PCI Express

The older PCI buses are being replaced by PCI Express, (PCIe) a new serial-oriented bus introduced by Intel in 2004. This bus is a totally new design and is incompatible with older PCI cards:

- **It transfers data over individual serial "lanes" one bit after another.**
- **The transfer rate is 2.5 GBytes/sec or 250Mbytes/sec.**
- **PCIe slots and cards can have multiple "lanes". A "16X" card has 16 lanes and can transfer data at 16 x 250 = 4000Mbytes/sec.**
- **It is a switched bus – this means that each master and slave get their own individual dedicated connection and can transfer data at the same time as other masters and slaves.**

The picture at right shows a motherboard with the following connectors:

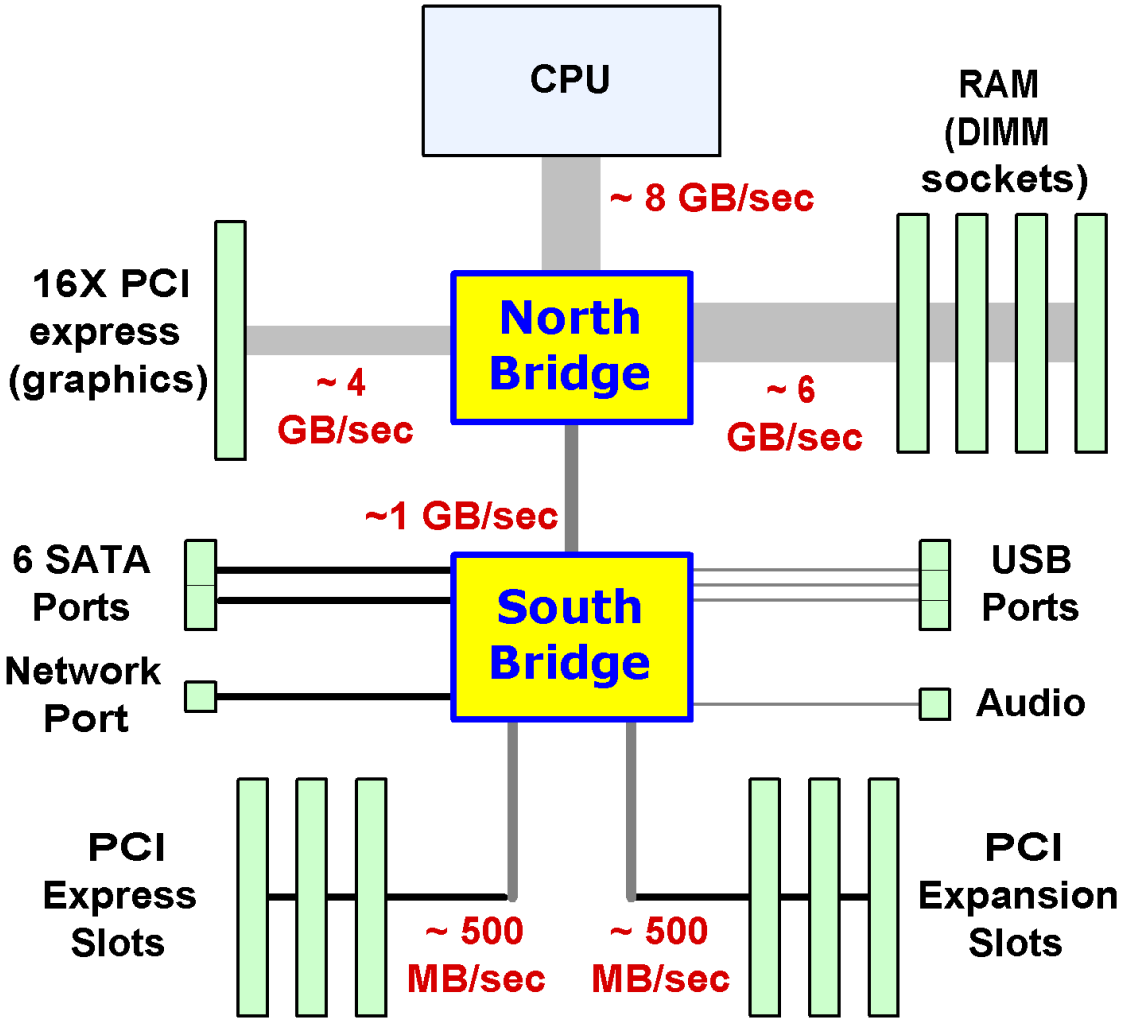
- **4X PCIe**
- **16X PCIe**
- **1X PCIe**
- **16X PCIe**
- **PCI**



Chipsets

Modern computers use support chips to isolate the buses from the CPU so that they're not dependent on the specific signals used by a particular CPU. This means that the buses don't need to be changed with each new CPU generation. Instead, a "chipset" is used to interface a specific CPU into a system with cache, memory, standard I/O ports, and a PCI bus.

The most common chipsets consist of two main chips, the "North" and "South" Bridges. The "North Bridge" is a high-performance chip that connects the highest-speed components , and the "South Bridge" connects the slower-speed devices. The diagram at right gives an idea of the speeds of the connections.



New CPUs with different pinouts require the use of a different chipset to interface them with the rest of the system.

Although the textbook shows a system with an ISA bus, current systems no longer use it.

Chipsets

Not a lot is heard about the characteristics of chipsets - but they're almost as important as the CPU itself in building a system with good performance and capabilities.

Chipsets are usually designed by the CPU manufacturer, and usually there are different versions of chipsets with different features. Some of the features to look for when evaluating a chipset include:

- **CPU**
  What type/speed of CPU does it support and does it support a dual- or quad-CPU system?
- **Memory support**
  How much memory does the chipset support? What type and speed of memory? Does it support parity and/or ECC memory? Memory interleaving?
- **Performance**
  What is the bandwidth between the CPU, memory, AGP and other system components?
- **Standard I/O Ports**
  What type of standard I/O ports does it support and how many of each (USB, ATA or SATA, NIC, etc.)? Does it include built-in audio and video capability? Can these be disabled to allow the use of custom PCI audio or video cards?
- **PCI and PCIe Bus Slots**
  What type of buses does it support and how many slots?
- **Other features**
  Does the chipset support remote management, wake-on LAN, hyperthreading, low-power modes, etc.?

The Universal Serial Bus

External devices such as mice, keyboards, scanners, etc. need an easy connection that doesn't require opening up the computer to plug in a card. The Universal Serial Bus (USB) is an external I/O bus which fills that need.

USB was designed with the following criteria in mind:

- **No hardware configuration (jumpers or switches) should be needed to add devices**
- **New devices must be able to be added without having to open up the computer system**
- **Only one kind of cable should be required no matter what type of device is used**
- **I/O devices should be able to get their power from the same cable as is used for data**
- **It should be possible to add lots of devices (up to 127)**
- **Real-time devices (sound, telephone) should be supported**
- **New devices should be able to be installed while the computer is running**
- **It should not be necessary to reboot the computer after installing a device**
- **The circuitry to support the bus shouldn't add very much cost to a device**

As mentioned earlier, USB comes in two speeds:

USB V1 (or V1.1) slow to medium-speed devices – up to 1.5 MByte/sec

USB V2 for slow to high-speed devices – up to 60MByte/sec

Note – it's common to see USB speeds quoted as "12Mbit/sec" or "480Mbit/sec" – but it's more useful to know the speed in MByte/sec.

USB Connectors and Cables

USB Ports on a computer system. These can be identified by their rectangular shape and (usually) by the USB icon.

The host connector end of a standard USB cable (sometimes called the "A" end of the cable)

The device connector end of a standard USB cable (sometimes called the "B" end of the cable).

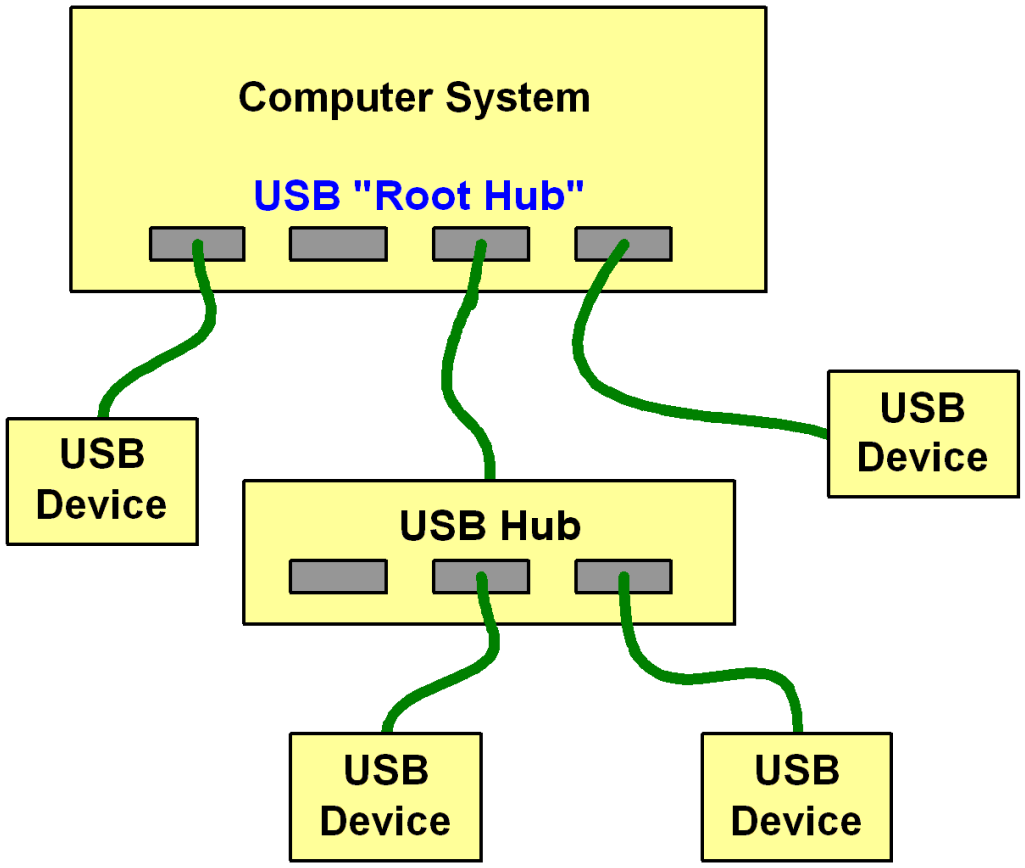
A special "mini" device connector ("B" end) for portable devices which don't have space for the full-sized connector.

USB is a serial protocol, so the connectors and cables carry only four wires. Two of the wires carry power and two of the wires carry data. USB V2 has a 5m (~15 foot) length restriction, and is more sensitive to the quality of the cable than USB V1 is.

USB Topology

USB connections are like a "client-server" system – one end (the USB port in the computer) is always the controlling end and the other end is the controlled device. It's not possible to connect the USB ports in two computers together, or to connect two devices together to exchange information.

The USB ports in the computers are known as the "Root Hub". USB devices can plug directly into these ports, or several devices can be connected to the same USB port by using an external "USB Hub". USB devices connect together in a tree topology, where the "Root Hub" is the base of the tree, the external hubs are branches, and the devices are the leaves.

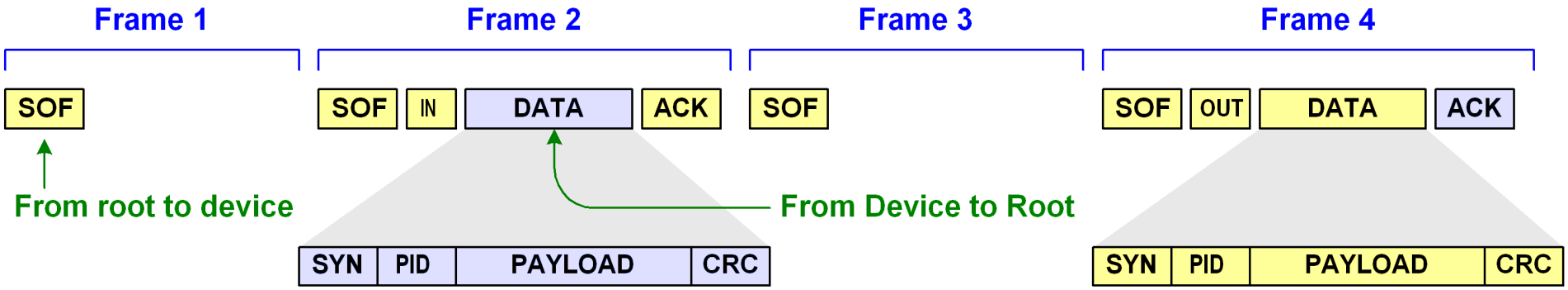


The USB connection can supply power to simple devices like mice and keyboards so they don't have to be plugged into a wall outlet. But when you plug several devices into a hub you may try to draw more power from the computer's USB port than it's designed to handle. It's a good idea to always use "powered hubs" which themselves plug into a wall outlet so that they can provide enough power to drive all of the devices that are plugged into them.

USB Operation

USB cabling provides a logical "bit pipe" connection between the root hub and each device. The connection can be logically divided into as many as 16 sub connections to support different streams of data (for example, a video and an audio stream from a USB Webcam).

Information travels over the pipe in "frames" which are transmitted every millisecond. (Each frame takes the same amount of time although this isn't shown in the diagram below):



SOF – A special pattern transmitted to identify the Start Of Frame. It's transmitted every millisecond whether there is data to be transmitted or not.

IN / OUT – Commands to address a specific USB device and request a data transfer

DATA – A packet of data which includes synchronization bits (SYN), a Packet ID (PID), the actual data being transferred (PAYLOAD), and a checksum (CRC) used to detect transmission errors.

ACK – Sent by the receiver to indicate whether the data was received successfully or not.

Note that these are only sample exchanges, there are other type of exchanges as well.

USB “On the Go”

A newer version of the USB spec is “USB On the Go”. This version is designed to let USB devices interoperate without having to be connected to a computer. For example, it would allow pictures from a USB digital camera to be transferred directly to a USB hard drive without having to use a computer.

Devices built with USB On the Go can recognize when they’re connected to another device that is not a root hub, and they reconfigure themselves to act as a root hub to allow data to be passed between them.

To be useful, devices that support “On the Go” need to have a user interface which can be used to control the transfer. For example, options need to be included in the menu of a digital camera to allow the user to show which picture files exist in the camera and on a connected USB hard drive and to choose files to transfer one way or another.

USB “On the Go” can also include:

- **Low power enhancements for use with battery-powered devices**
- **A new smaller connector for use with small mobile products**

Wireless USB

The another flavour of the USB spec is "Wireless USB", which is designed to let USB devices interoperate via a radio connection without requiring a physical connection.

Wireless USB allows PCs and other devices to communicate at up to 60MB/sec (same as USB V2) at a range of up to 3 meters. The specification includes power conservation modes for battery-powered devices and encryption to address privacy issues.

Wireless USB is similar to an another wireless standard called Bluetooth that does much the same thing, but at somewhat slower speeds. Bluetooth is widely used with consumer electronics devices such as cellular phones and music players.

As with all wireless technologies, security will be a concern. The need for robust security is at odds with a technology that is simple to set up and use.

FireWire (AKA "i.Link" and "IEEE 1394")

Firewire is another external I/O bus that is similar in many respects to USB. It's a serial bus that connects using a simple (six-wire) cable to send data at medium to high speeds. IEEE 1394 was developed around the same time as USB but never became as popular. It's now used mainly as a means of connecting Digital Video (DV) cameras to allow taped video to be transferred to a computer. Even this is being replaced by USB as tape is made obsolete by flash memory and hard drives for the storage medium in camcorders.

One of the reasons that Firewire never became very popular was that it's developers required royalty fees to use the technology. This caused Intel to develop USB as a way to avoid these fees.

There is also confusion over it's name. Apple trademarked the name "FireWire", but the official specification is called "IEEE 1394" and Sony calls it "i.Link".

Sony and Apple computers usually have built-in FireWire connectors, but for most other computers you often need to buy and install a PCI card to add the ports to your system.

There are several upwardly-compatible versions available:

- **Firewire 400 – up to about 50MByte/sec**
- **Firewire 800 – about 100MByte/sec**
- **Firewire 3200 – about 400MByte/sec (very recent and not yet in common use)**

Key Concepts

- **Memory and I/O port connection and address decoding**
- **CPU connections: data vs. address**
- **Types of CPU control signals**
- **Types of computer buses**
- **Bus masters vs. bus slaves**
- **Synchronous vs. Asynchronous buses**
- **Bus arbitration**
- **Block transfers and interlocked bus cycles**
- **PCI bus characteristics**
- **AGP bus characteristics**
- **Chipset characteristics**
- **USB Bus characteristics**

What's Next

- **Look at Review Questions for this week**
- **Study for the Mid-Term exam next week, which includes:**
  - **All the lectures so far**
  - **All the WebCT modules so far**

(Note – marked quizzes and assignments will be handed back before the start of the exam)