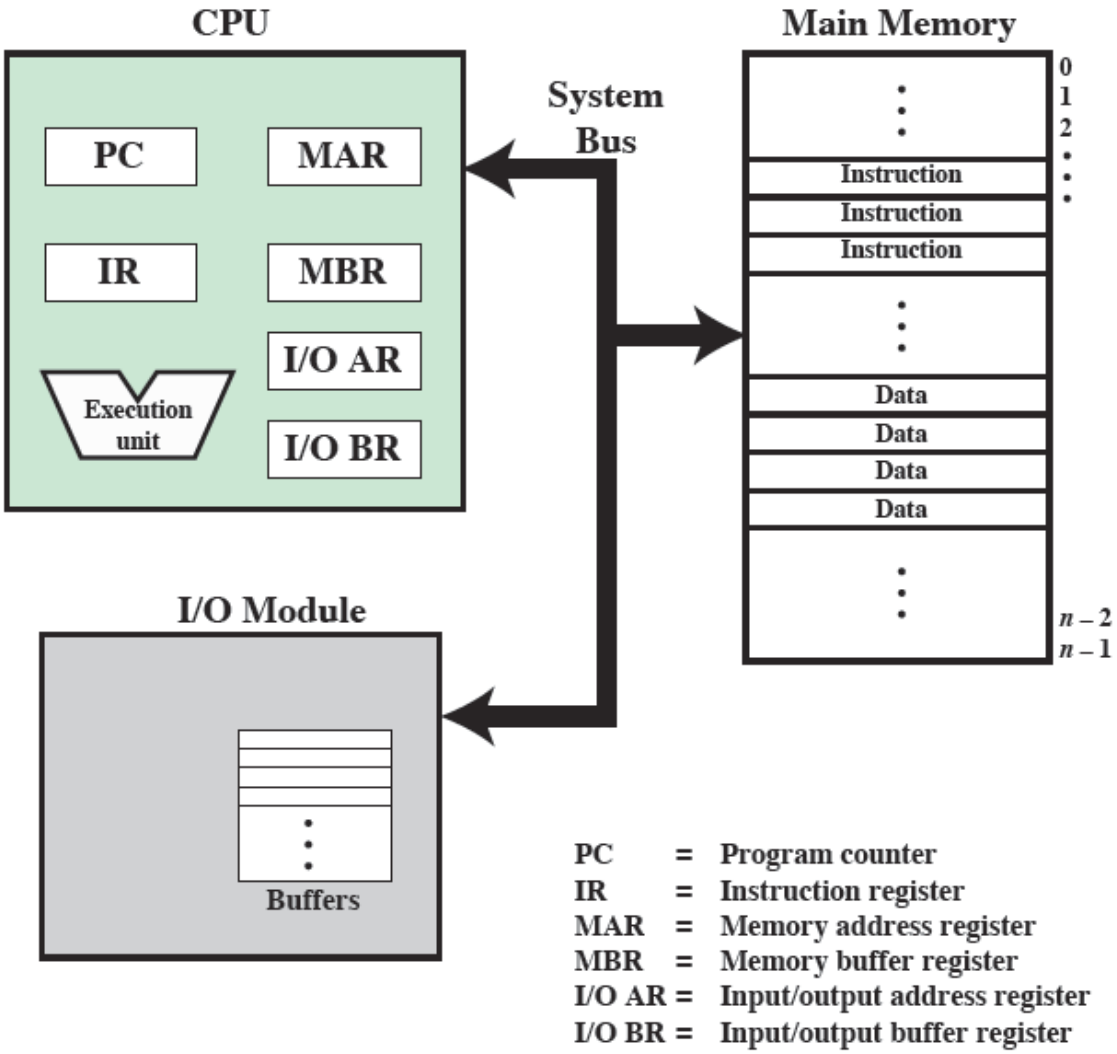
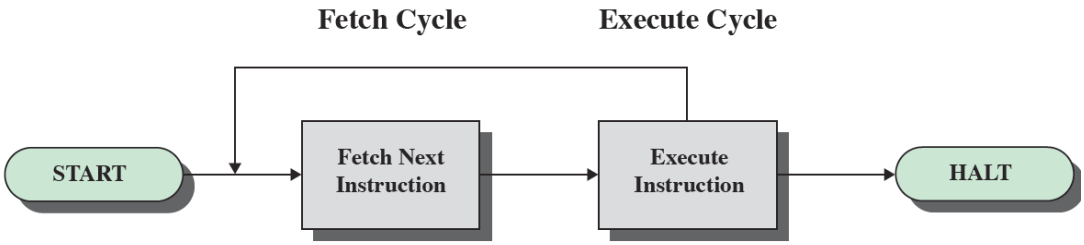
CS 320
Ch. 3

The von Neuman architecture characteristics are: **Data and Instruction in same memory, memory contents addressable by location, execution in sequence.**

The CPU consists of an instruction interpreter, a module of general purpose arithmetic and logic functions, m**emory, and I/O which are interconnected by a System Bus.**



Instructions consist of a fetch cycle and an execute cycle. The CPU registers involved in the Fetch Cycle are the **PC, MAR, Buffer, IR**



All instructions are of four types:

1. CPU ↔ Memory (Load and Store)
2. CPU ↔ I/O (input and output)
3. Data processing (Add, and, etc)
4. Control (branch, call)

Instructions typically have an op code and one or more operands. Opcode tells what the instruction will do and the operands specify the data.

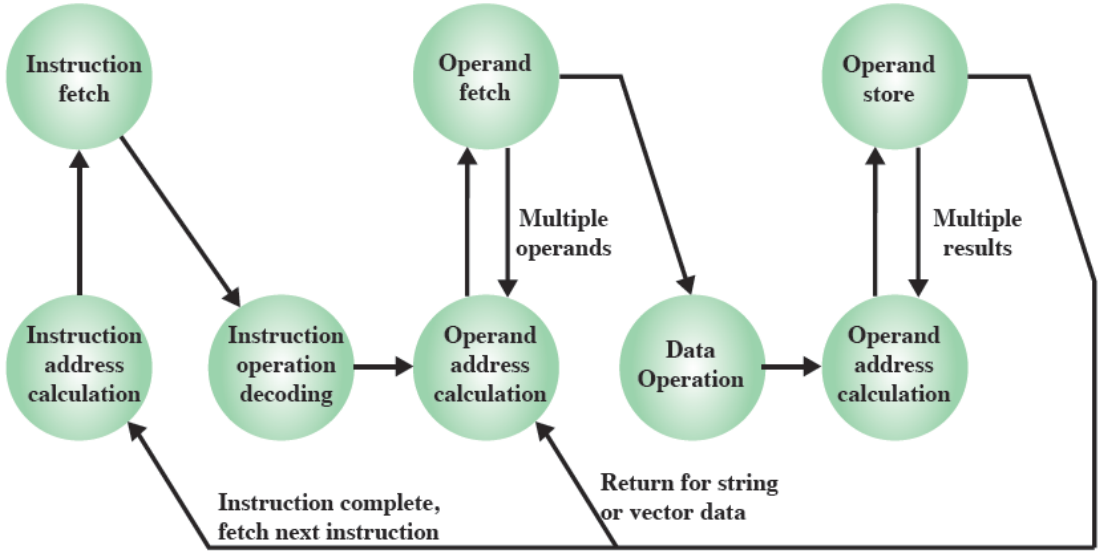
The op code size is determined by the number of different instructions. The operand size is determined by the number of registers and the number of operands. It is often influenced by the size of the address.

With a 4-bit op code you can only specify 16 unique instructions.

With a 12 bit operand the instruction can address 4096 memory locations.

The number of operands an instruction has is determined by the opcode.

The state diagram for an instruction cycle is shown below.



An interrupt is an unscheduled or asynchronous operation triggered by something external to the program.

We need interrupts for efficiency. Without them we would have to constantly poll external sources for their needs.

There are four types of generic interrupts:

1. program
2. timer
3. I/O
4. Hardware failure.

A program interrupt is triggered by the program in software so it is technically not an interrupt.
A timer interrupt is triggered by an internal timer.
An I/O interrupt is triggered by an I/O device external to the CPU.
A hardware interrupts is triggered by a failure (disk not ready, division by zero, …)

This figure shows how interrupts are handled in the instruction sequence.

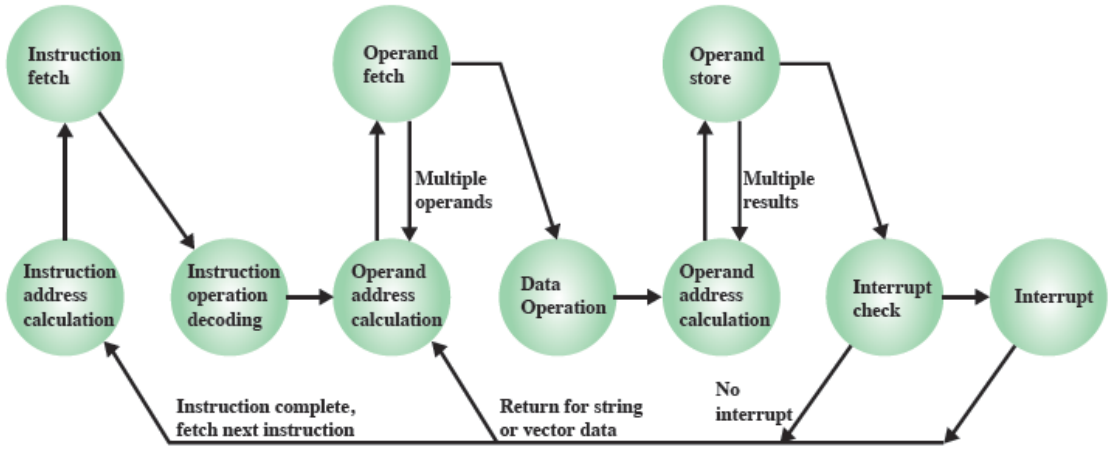


Figure 3.12 Instruction Cycle State Diagram, With Interrupts

There are generally two methods of handling multiple interrupts: 1. disable interrupts, 2. prioritize interrupts.
The disable scheme works by allowing one interrupt to disable all others until it is done.
The prioritize scheme works using hardware which allows a higher priority interrupt to interrupt a running interrupt.

When an interrupt is received the software must do the following:

1) Finish the current instruction
2) Save the current state and the return address
3) branch to the ISR
4) turn off the interrupt
5. Restore the current state
6) go back to the return address

Note that the interrupt must be turned off by the ISR

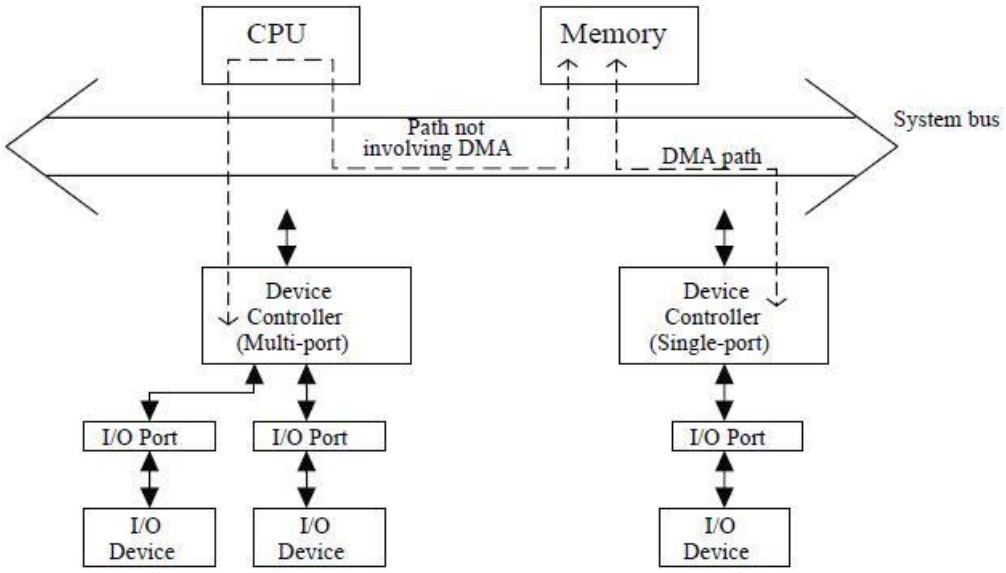
Interrupts can be edge sensitive and level sensitive interrupts.

Interrupts can be triggered by the rising or falling edge of an eternal signal or by the fact that the external signal is high or low.

The edge triggered interrupt must store the edge data (in an internal FF) and it remains on until it is serviced.

Level sensitive interrupts are better for many interrupts that are ORed together. The ISR must notify the external hardware to turn the interrupt off.

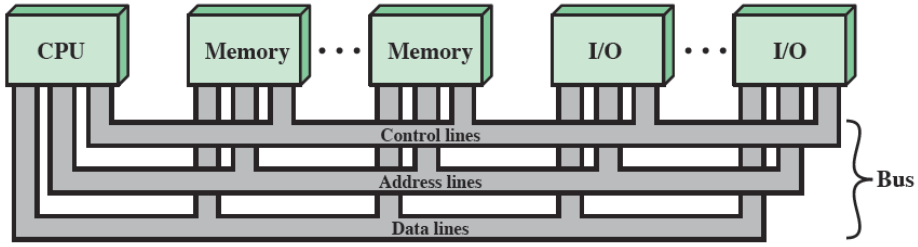
DMA is Direct Memory Access. It allows an I/O device to connect directly to memory without going through the CPU.



A bus is an **interconnection path between two or more devices with 1 sender and multiple receivers.**

A system bus **Connects CPU/Memory/IO**

There are generally 3 types of signals on a bus. **Data, address, control signals (some busses also have power and clock signals)**



Bus performance degrades as more things are added to the bus because propagation delay increases and users must wait on each other

There is a correlation between speed and power. Higher speeds generally imply more power. This is due to the increase in the number of times stray bus capacitance must be charged and discharged.

Multibus systems have a local bus, a system bus, and an expansion bus.

A) A local bus **Directly connects CPU to cache or fast I/O**

B) A system bus **Connects main memory, CPU, and some I/O**

C) An expansion bus **Ties the system bus to I/O devices**

Most busses have some type of bus arbitration. A bus arbiter decides who gets to use the bus.

Bus arbitration can be centralized or distributed:

Central (one arbiter)

Distributed (no arbiter but uses master/slave)

But timing may be synchronous and asynchronous

Synchronous bus is **faster and simpler. Every transfer is governed by the clock.**

Asynchronous bus is more **flexible – there is no clock and devices run at their own speed.**

Data bus width is loosely related to performance. A wider bus allows more data to be accessed at one time.

Address bus width is also loosely related to performance. M**ore capacity allows the operating system access to more memory without invoking virtual memory.**

The author says that computers have relied on the shared bus architectures for decades but we are now moving to a point-to-point interconnect. **Effectively a non-shared bus that connects two internal systems.**

A shared bus structure is **Too constrained to make wide synchronous high speed busses. The Point-to-Point bus has lower latency, higher data rates, and better scalability.**

The QPI bus is a **Quick Path Interconnect – has multiple direct connections, a layered protocol architecture, and packetized data transfers.**

QPI bus applied **Multi-core processors.**

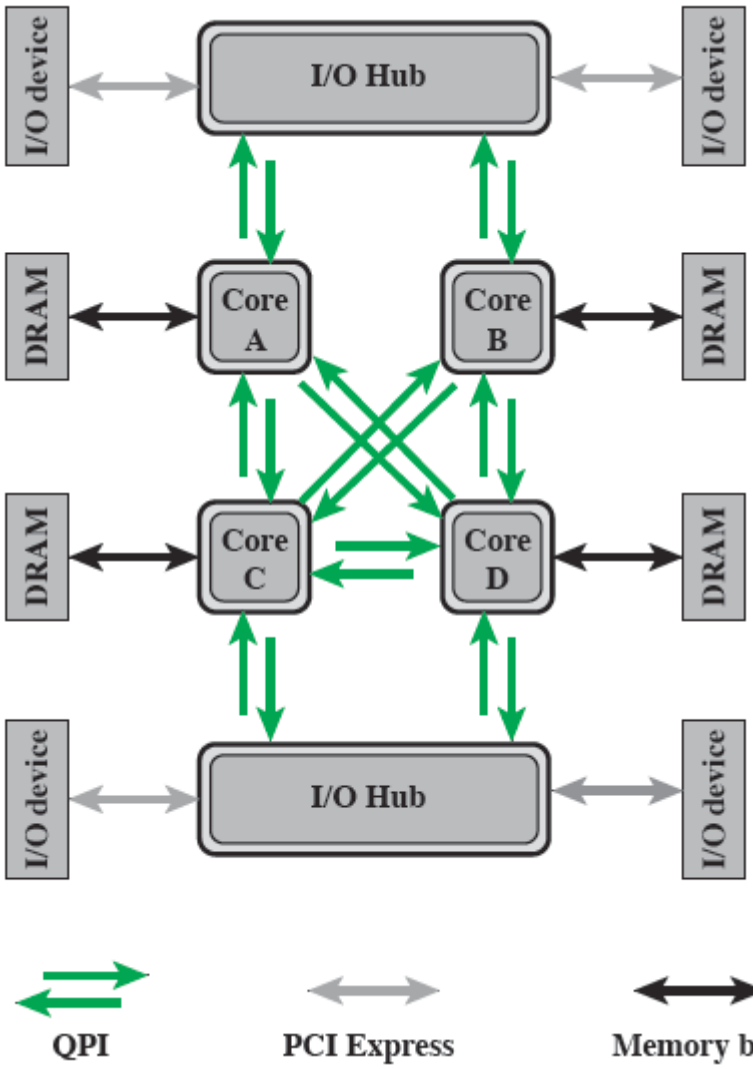


Figure 3.17 Multicore Configuration Using QPI

The PCI bus was invented by **about 1990.**

The PCI (Peripheral Component Interconnect) bus is multiplexed with respect to address and data.

The PCI bus is said to be a *reflected bus*. What does this mean?

PCI bus is synchronous

The PCI bus. **Uses a central bus arbiter where each bus slot has unique req/grant lines.**

The interrupt system supported by the PCI bus is a **4-line level sensitive to be shared.**