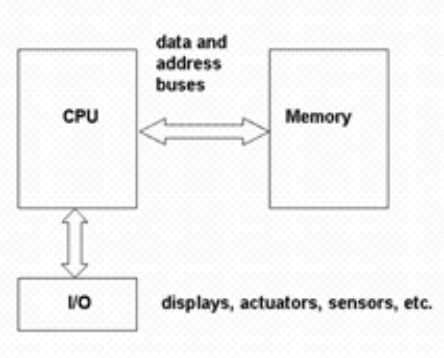
Topic 2: Hardware Considerations [1]

A. Basic architecture
B. Hardware interfacing (from a software or system engineering perspective)
C. CPU
D. Memory
E. I/O
F. Enhancing performance
G. Other special devices
H. Non von Neumann architectures

Basic Architecture

- Three system wide busses: power, address, and data
- System bus refers to the address and data busses collectively
- Real-time systems are:
  - single processor systems
  - multiprocessor systems
    - loosely coupled through messaging
    - schedule tasks across different processors



Von Neumann architecture

Ch2: Hardware Considerations

A. Basic architecture
B. Hardware interfacing (from a software or system engineering perspective)
C. CPU
D. Memory
E. I/O
F. Enhancing performance
G. Other special devices
H. Non von Neumann architectures

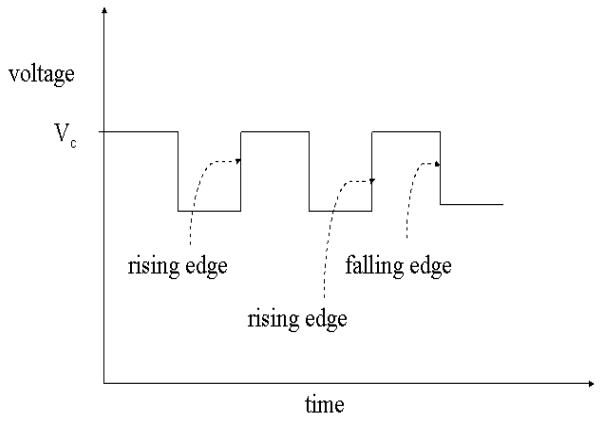
Hardware Interfacing (From a Software or System Engineering Perspective)

- Latching
- Edge versus level triggered
- Tri-state logic
- Wait states
- System interfaces and busses

Latching

- Mechanism for “recording” the appearance of a signal between devices for later processing.
- Interrupt signals are latched into the programmable interrupt controller so that they can be serviced at an appropriate time.
- Once the latch is read, it needs to be reset so that a new signal can be received.
- In the case of an interrupt, if a second interrupt is signaled on the same input a second interrupt may be lost. Therefore, it is important to read and clear the latch as soon as possible.

Edge versus Level Triggered



A fictitious time-varying signal (typically, a clock) showing two rising edges, each of which represents a single event, and a falling edge. *Vc* represents a critical or threshold voltage

Tristate Logic

- When multiple devices are connected to the same bus structure those devices that are not involved are placed into a high-impedance state at their bus interconnections.
- This is called “tri-stating” the device. Tri-state logic is essential in the design of computer systems.
- Signals can be in one of three levels:
  - high, low and tri-stated.
- Signals that are improperly tri-stated will be in an unknown state in which the signal is “floating” (arbitrarily high or low).
- Floating signals can be the source of many insidious problems such as falsely indicated interrupts, improper setting of switches, and so on.

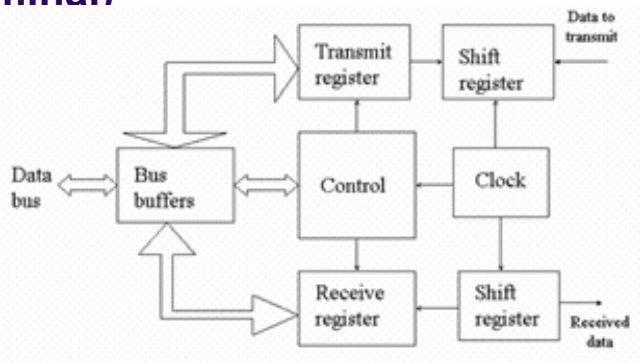
Wait States

- When a microprocessor must interface with a slower peripheral or memory device, a wait state may be needed to be added to the bus cycles.
- Wait states extends the microprocessor read or write cycle by a certain number of processor clock cycles to allow the device or memory to “catch up.”
- For example, EEPROM, RAM, and ROM may have different memory access times. Since RAM memory is typically faster than ROM, wait states would need to be inserted when accessing RAM.
- Wait states degrade overall systems performance, but preserve determinism.

Systems Interfaces and Busses

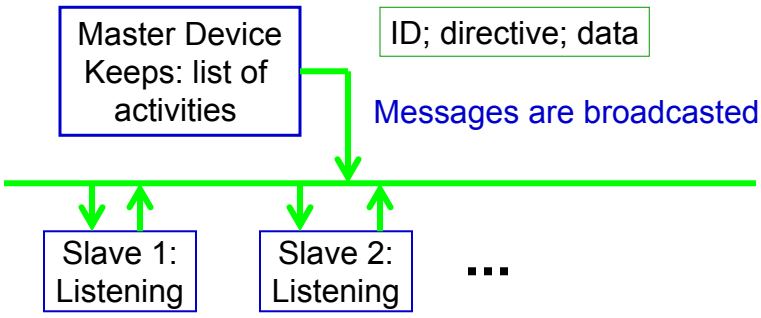
- UART
- MIL-STD-1553B
- SCSI
- IEEE 1394 Firewire

UART (Universal Asynchronous Relay Terminal)

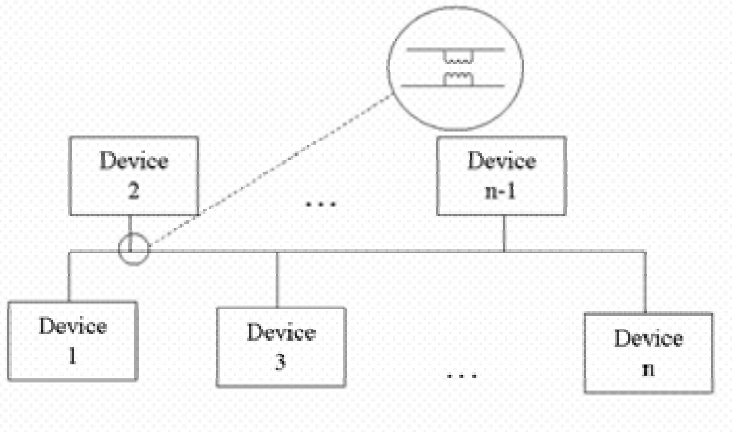


A transmitter/receiver device used to multiplex parallel data to serial. To receive, the parallel data are captured from the bus into a receive register and then shifted into a serial stream of bits. To transmit, the data are loaded into a shift register, then shifted into a parallel transmit-receive buffer for transmission.

MIL-STD 1553B



MIL-STD 1553B

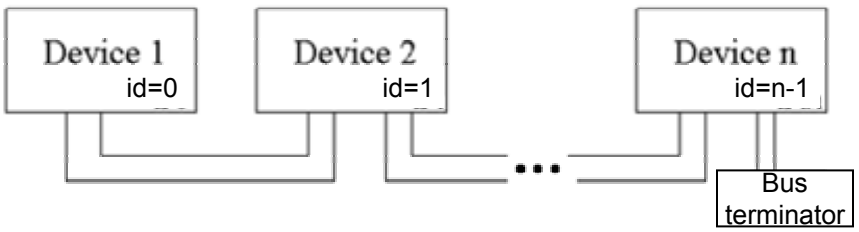


MIL-STD 1553B configuration. The inset shows the inductive coupling connection specified. Such a connection helps the system withstand electrical failure of one of the devices

Small Computer System Interface (SCSI) or "Scuzzy"

- This is a PC-based parallel interface that supports many kinds of devices
- The SCSI 3 standard supports new connection types such as: Fiber Channel and FireWire
- SCSI supports devices connected in a daisy-chained fashion
- ID=0 is set for the boot device and the higher the ID number the higher the priority of the device in bus access arbitration

Small Computer System Interface (SCSI) or "Scuzzy"



SCSI daisy chained connection. Daisy chain connections are used in many kinds of devices in an embedded system (e.g. interrupt controllers) because they allow for an easy "extension" of the system bus by simply attaching to the device at the end.

IEEE 1394 Firewire

- A very fast external bus standard that supports data transfer rates of up to 400Mbps (in 1394a) and 800Mbps (in 1394b).
- Can be used to connect up to 63 external devices.
- Defines 100, 200, and 400 Mbps devices and can support the multiple speeds on a single bus, and is flexible in the sense that:
  - Supports freeform daisy chaining and branching for peer-to-peer implementations.
- It is also hot pluggable (devices can be added and removed while the bus is active).

IEEE 1394 Firewire

- Supports two types of data transfer: asynchronous and isochronous.
  - Asynchronous – for traditional computer memory-mapped, load and store applications.
  - Isochronous – provides guaranteed data transport at a pre-determined rate.
    - Used for multimedia applications where uninterrupted transport of time critical data and just in time delivery reduce the need for costly buffering.
    - Ideal for devices that need to transfer high levels of data in real time, such as cameras, VCRs and televisions.

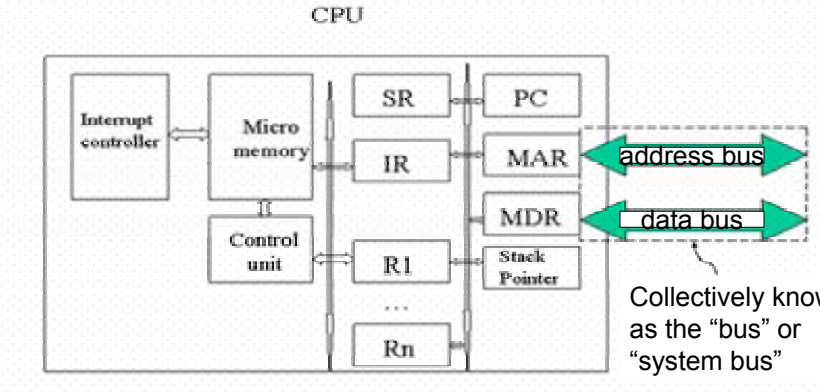
Ch2: Hardware Considerations

A. Basic architecture
B. Hardware interfacing (from a software or system engineering perspective)
C. CPU
D. Memory
E. I/O
F. Enhancing performance
G. Other special devices
H. Non von Neumann architectures

CPU

- Basic structure
- Fetch and execute cycle
- Microcontrollers
- Instruction forms
- Core instructions
- Addressing modes
- RISC versus CISC

Basic Structure

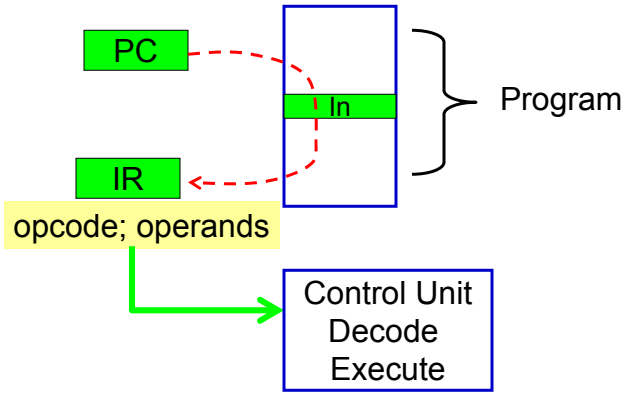


Partial, stylized, internal structure of a typical CPU. The internal paths represent connections to the internal bus structure. The connection to the system bus is shown on the right.

Fetch and Execute Cycle

- Programs are a sequence of macroinstructions or macrocode stored in the main memory in binary form.
- Macroinstructions are sequentially fetched from the main memory location pointed to by the program counter, and placed in the instruction register.
- Each instruction consists of an operation code or opcode field and zero or more operand fields.
- The control unit decodes the instruction.

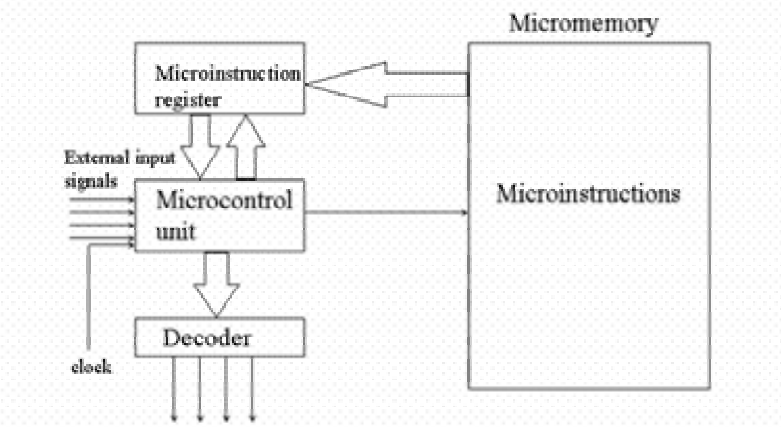
Fetch and Execute Cycle



Fetch and Execute Cycle

- After executing the instruction, the next macroinstruction is retrieved from main memory and executed.
- Certain macroinstructions or external conditions may cause a nonconsecutive macroinstruction to be executed.
- This process is called the fetch-execute cycle (or fetch-decode-execute).
- Even when "idling," the computer is fetching and executing an instruction that causes no effective change to the state of the CPU and is called a no-op (for no-operation).

Microcontrollers



A microcontroller is a computer system that is programmable via microinstructions. Because the complex and time consuming macroinstruction decoding process does not occur, program execution tends to be very fast.

Instruction Forms

- An instruction set constitutes the language that describes a computer's functionality and its organization.
- Most instructions make reference to either memory locations, pointers to a memory location, or a register.

Computer Organization

Implementation details

Computer Architecture

Hardware details that are visible to the programmer

Instruction Forms

- 0-address form
  - use the stack locations as operands
  - are found in programmable calculators that are programmed using postfix notation
- 1-address form
  - Uses implicit register (accumulator)
- 2-address form
  - Has the form: `op-code operand1, operand2`
- 3-address form
  - Has the form: `op-code operand1, operand2, resultant`

Core Instructions

- There are generally six kinds of instructions. These can be classified as:
  - horizontal-bit operation
    - e.g. `AND, IOR, XOR, NOT`
  - vertical-bit operation
    - e.g. rotate left, rotate right, shift right, and shift left
  - Control
    - e.g. `TRAP, CLI, EPI, DPI, HALT`
  - data movement
    - e.g. `LOAD, STORE, MOVE`
  - mathematical/special processing
  - other (processor specific)
    - e.g. `LOCK, ILLEGAL`

Example of Special Instruction to Support Real-Time Implementations

- Intel IA-32 family provides: LOCK, HLT, BTS
- LOCK => LOCK# signal is asserted =>
  - instruction becomes atomic (uniterruptible)
  - processor has exclusive use of any shared memory while the signal is asserted
- HLT: halt processor => processor is stoped
  - until enabled interrupt or debug exception is received
- BTS (Bit Test and Set): can be used with LOCK prefix to allow the instruction to be executed atomically

Addressing Modes

- Three basic modes
  - immediate data
  - direct memory location
  - indirect memory location
- Others are combinations of these
  - register indirect
  - double indirect

RISC versus CISC

- Complex Instruction Set Computer (CISC)
- CISC based upon the following set of principles:
  - Complex instructions take many different cycles.
  - Any instruction can reference memory.
  - No instructions are pipelined.
  - A microprogram is executed for each native instruction.
  - Instructions are of variable format.
  - There are multiple instructions and addressing modes.
  - There is a single set of registers.
  - Complexity in the micro program and hardware.

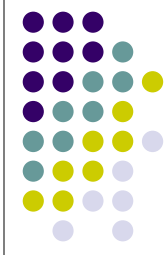
RISC versus CISC

- Reduce Instruction Set Computer (RISC)
- RISC criteria are a complementary set of principles to CISC.
  - Simple instructions taking one clock cycle.
  - LOAD/STORE architecture to reference memory.
  - Highly pipelined design.
  - Instructions executed directly by hardware.
  - Fixed format instructions.
  - Few instructions and addressing modes.
  - Large multiple register sets.
  - Complexity handled by the compiler and software.

RISC versus CISC

- RISC has fewer instructions; hence more complicated instructions are implemented by composing a sequence of simple instructions.
- RISC needs more memory than the equivalent CISC instruction.
- RISCs have several major advantages in real-time systems:
  - The average instruction execution time is shorter than for CISCs.
  - The reduced instruction execution time leads to shorter interrupt latency and thus shorter response times.
  - RISC instruction sets tend to allow compilers to generate faster code because the limited instruction set facilitates a greater number of optimization approaches.

Ch2: Hardware Considerations



A. Basic architecture
B. Hardware interfacing (from a software or system engineering perspective)
C. CPU
D. Memory
E. I/O
F. Enhancing performance
G. Other special devices
H. Non von Neumann architectures

Memory

- Memory access
- Memory technologies
- Memory organization

Memory Access

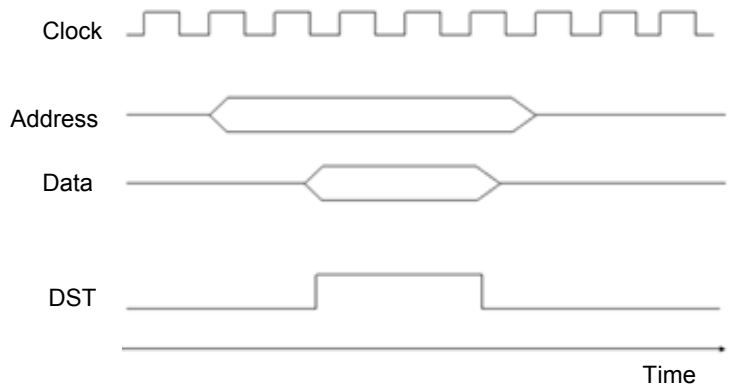


Illustration of the clock synchronized memory transfer process between a device and the CPU. The symbolism "<>" shown in the data and address signals indicates that multiple lines are involved during this period in the transfer.

Memory Technologies

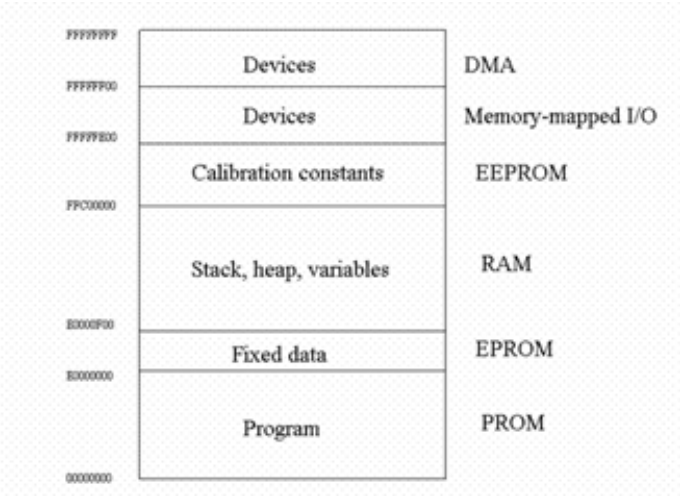
- Primary and secondary memory storage forms a hierarchy involving access time, storage density, cost and other factors.
- The fastest possible memory is desired in real-time systems, but economics dictates that the fastest affordable technology is used as required.
- In order of fastest to slowest, memory should be assigned, considering cost as follows:
  - internal CPU memory
  - registers
  - cache
  - main memory
  - memory on board external devices

Memory Technologies

Memory type	Typical access time	Density	Typical applications
DRAM	50-100 ns	64 Mb	main memory
SRAM	10 ns	1 Mb	µmemory , cache, fast RAM
UVROM	50 ns	32 Mb	Code and data storage
Fusible link PROM	50 ns	32 Mb	Code and data storage
EEPROM	50-200 ns	1 Mb	Persistent storage of variable data
Flash	20-30 ns (read) 1 µs (write)	64 Mb	Code and data storage
Ferroelectric RAM	40 ns	64 Mb	various
Ferrite core	10 ms	2 Kb or less	None, possibly ultra-hardened non-volatile memory

Selection of the appropriate technology is a systems design issue.

Memory Organization



Typical memory map showing designated regions.

Ch2: Hardware Considerations

A. Basic architecture
B. Hardware interfacing (from a software or system engineering perspective)
C. CPU
D. Memory
E. I/O
F. Enhancing performance
G. Other special devices
H. Non von Neumann architectures

I/O

- Programmed I/O
- Direct memory access (DMA)
- Memory-mapped I/O
- Interrupts

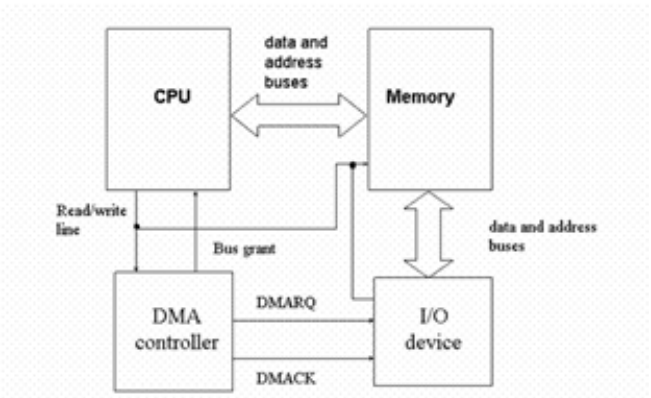
Programmed I/O

- Special data movement instructions are used to transfer data to and from the CPU.
- An IN instruction transfers data from a specified I/O device into a specified CPU register.
- An OUT instruction outputs from a register to some I/O device.
- Normally, the identity of the operative CPU register is embedded in the instruction code.
- Both the IN and OUT instructions require the efforts of the CPU and thus cost time that could impact real-time performance.

Direct Memory Access (DMA)

- Access to the computer's memory is given to other devices in the system without CPU intervention.
- Information is deposited directly into main memory by the external device.
- DMA controller is required unless the DMA circuitry is integrated into the CPU.
- Because CPU participation is not required, data transfer is fast.

Direct Memory Access (DMA)

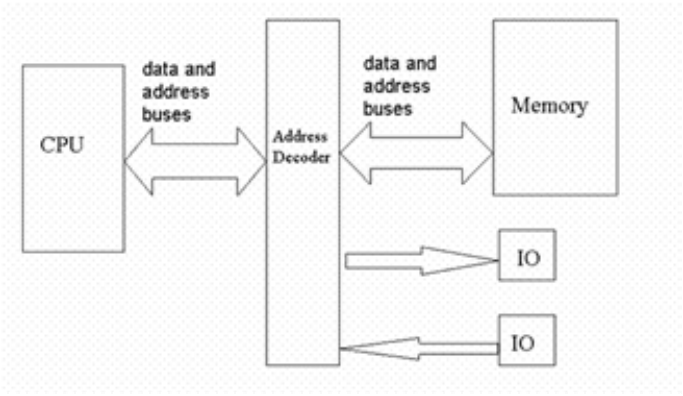


The DMA controller prevents collisions by requiring each device to issue a DMA request signal (DMARQ) that will be acknowledged with a DMA acknowledge signal (DMACK). Until the DMACK signal is given to the requesting device its connection to the main bus remains in a tri-state condition. Any device that is tri-stated cannot affect the data on the memory data lines.

Memory-Mapped I/O

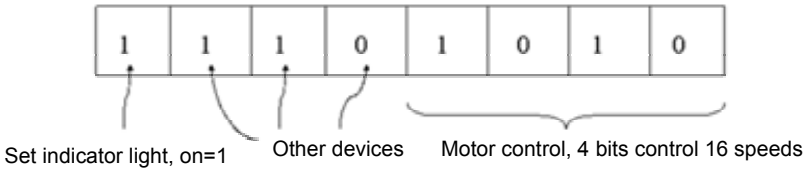
- Memory-mapped I/O provides a data transfer mechanism that is convenient because it does not require the use of special CPU I/O instructions.
- In memory-mapped I/O certain designated locations of memory appear as virtual input/output ports.

Memory-Mapped I/O



Input from an appropriate memory mapped location involves executing a LOAD instruction on a pseudomemory location connected to an input device. Output uses a STORE instruction

Memory-Mapped I/O



- A bit map (packed binary word) describes a view of a set of devices that are accessed by a single (discrete) signal and organized into a word of memory for convenient access either by DMA or memory mapped-addressing.

Interrupts

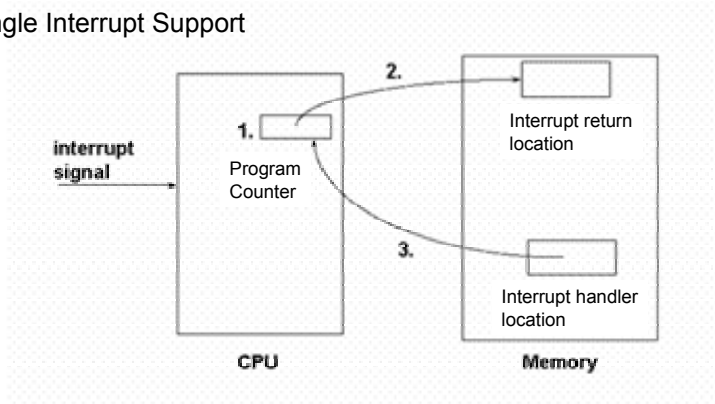
- Instruction support for interrupts
- Internal CPU handling of interrupts
- Programmable interrupt controller (PIC)
- Interfacing devices to the CPU via interrupts
- Interruptible instructions
- Watchdog timers

Instruction Support for Interrupts

- Interrupt: a hardware signal that initiates an event
  - external
  - internal
- Processors provide two instructions
  - enable (turn on) priority interrupt (EPI)
  - disable (turn off) priority interrupt (DPI).
- These are atomic instructions that are used for many purposes, such as buffering, within interrupt handlers, and for parameter passing.

Internal CPU Handling of Interrupts

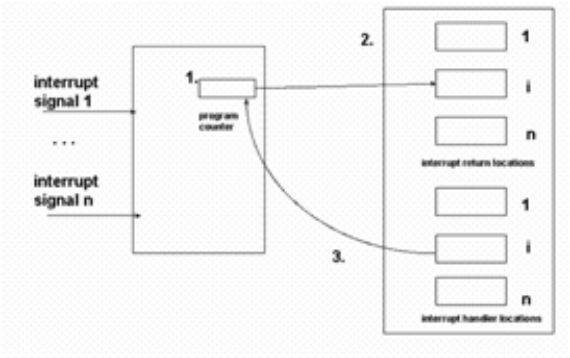
Single Interrupt Support



- Step 1: finish the currently executing macroinstruction.
- Step 2: save the contents of the PC to the interrupt return location
- Step 3: load the address held in the interrupt handler location into the program counter. Resume the fetch and execute sequence.

Internal CPU Handling of Interrupts

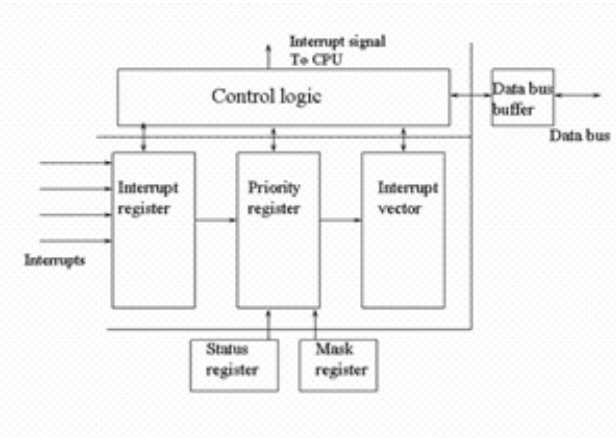
- Step 1: complete the currently executing instruction.
- Step 2: save the contents of PC to interrupt return location *i*.
- Step 3: load the address held in interrupt handler location *i* into the PC. Resume the fetch-execute cycle.



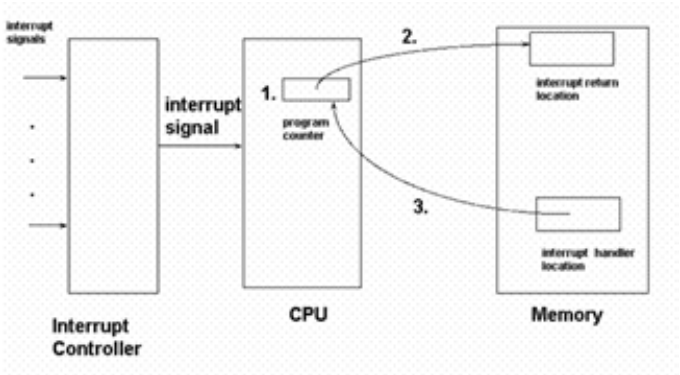
Multiple interrupt support

Programmable Interrupt Controller (PIC)

- A Programmable interrupt controller (PIC).
- **The registers**: interrupt, priority, vector, status, and mask serve the same functions as for the interrupt control circuitry of an on-board CPU.

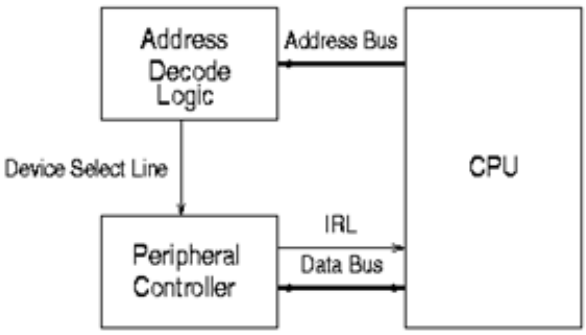


Programmable Interrupt Controller (PIC)



- Handling multiple interrupts with an external interrupt controller.
- Step 1: finish the currently executing instruction.
- Step 2: save the contents of the PC into the interrupt return location.
- Step 3: load the address held in the interrupt handler location into the program counter. Resume the fetch execute cycle. The interrupt handler routine will interrogate the PIC and take the appropriate action.

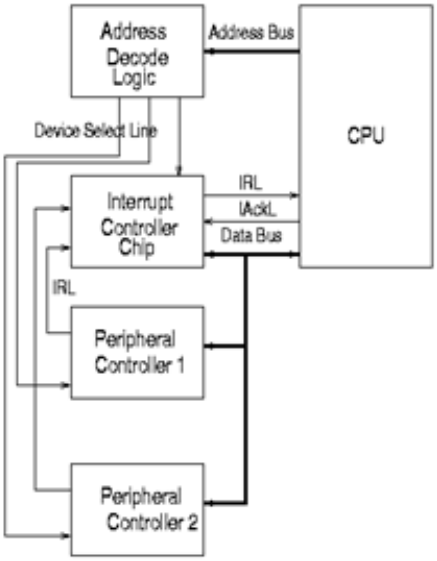
Interfacing Devices to the CPU via Interrupts



A Single peripheral controller. IRL is the interrupt request line. The controller's address on the address but activates the Device Select Line signal

Interfacing Devices to the CPU via Interrupts

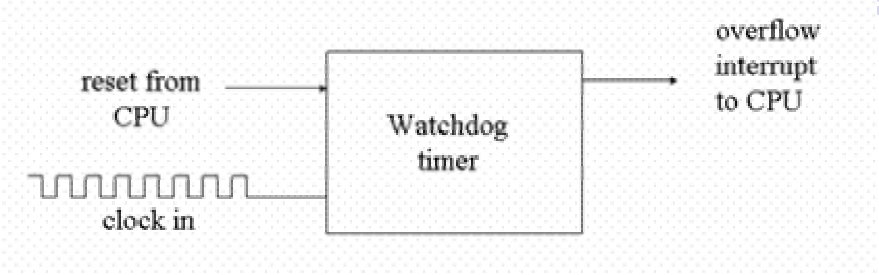
- Several peripheral controllers connected to the CPU via a programmable interrupt controller.
- Notice that the devices share the common data bus, which is facilitated by tri-stating, non-active devices via the device select lines.



Interruptible Instructions

- In rare instances certain macroinstruction may need to be interruptible.
- This might be the case where the instruction takes a great deal of time to complete.
  - E.g. a memory to memory instruction that moves large amounts of data.
  - In most cases, such an instruction should be interruptible between blocks to reduce interrupt latency.
  - However, interrupting this particular instruction could cause data integrity problems.

Watchdog Timers (WDT)



- **WDT**: is a counting register that is increased periodically. Overflow generates interrupt to CPU
- **NOTE**: the register must be cleared by appropriate code using memory-mapped I/O before the register overflows and generates an interrupt.

Ch2: Hardware Considerations

A. Basic architecture
B. Hardware interfacing (from a software or system engineering perspective)
C. CPU
D. Memory
E. I/O
F. Enhancing performance
G. Other special devices
H. Non von Neumann architectures

Enhancing Performance

- Locality of reference
- Cache
- Pipelining
- Coprocessors
- Other special devices

Two architectural enhancements that can improve average case performance in real- time systems are **caches and pipelines** => this proves that when the locality of reference is high, performance is improved

Locality of Reference

- Refers to the relative “distance” in memory between consecutive code or data accesses.
- If data or code fetched tends to reside relatively close in memory, then the locality of reference is high.
- When programs execute instructions that are relatively widely scattered locality of reference is low,
- Well-written programs in procedural languages tend to execute sequentially within code modules and within the body of loops, and hence have a high locality of reference.
- Object-oriented code tends to execute in a much more non-linear fashion. But portions of such code can be linearized (e.g. array access).

Cache

- A small block of fast memory where frequently used instructions and data are kept. The cache is much smaller than the main memory.
  - Also contains a table of memory address tags, which are currently in the cache. The table is often in the cache itself.
- Usage:
  - Upon memory access check the address tags to see if instruction/data is in the cache.
  - If present, retrieve data from cache,
  - If not present, cache contents are written back (into the main memory) and new block is read from main memory to cache.
  - The needed information is then delivered from cache to CPU and the address tags adjusted.
- Cache design considerations include: cache size, mapping function, block replacement algorithm, write policy, block size, and number of caches.

Cache

- Performance benefits are a function of cache hit radio.
- This is due to the fact that if needed data or instructions are not found in the cache, then the cache contents need to be written back (if any were altered) and overwritten by a memory block containing the needed information.
  - This overhead can become significant when the hit ratio is low. Therefore a low hit ratio can degrade performance.
- Hence, if the locality of reference is low, a low number of cache hits would be expected, degrading performance.
- Using a cache is also non-deterministic – it is impossible to know *a priori* what the cache contents and hence the overall access time will be.

Cache

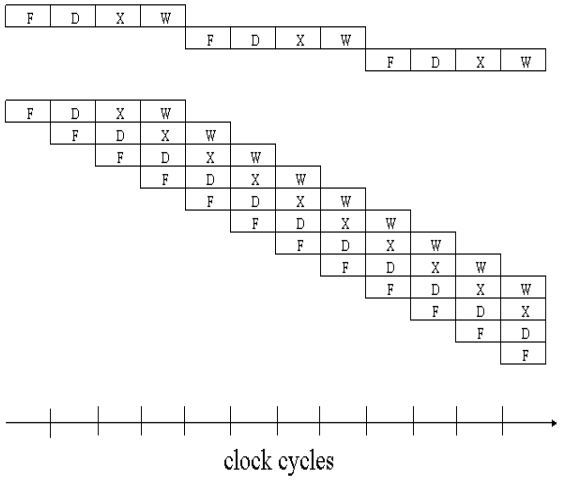
- What performance benefit does the cache give?
- Consider a simple system with a single cache
  - noncached memory reference costs 100ns
  - access from the cache 30ns
  - cache hit ratio is 60%
  - As a result the average access time would be:
  - 0.6*30 ns + 0.4 * 100 ns = 58 ns
- Performance benefits in a cache system are a function of the cache hit ratio

Pipelining

- Pipelining imparts an implicit execution parallelism in the different cycles of processing an instruction.
- With pipelining, more instructions can be processed in different cycles simultaneously, improving processor performance.
- Suppose execution of an instruction consists of the following stages:
  - fetch – get the instruction from memory
  - decode – determine what the instruction is
  - execute – perform the instruction decode
  - write – store the results to memory

Pipelining

- Sequential instruction execution versus pipelined instruction execution.
- Nine complete instruction can be completed in the pipelined approach in the same time it takes to complete three instruction in the sequential (scalar) approach



Pipelining

- ***speedup from pipelining = (pipeline depth)/(1+ pipeline stall cycles per instruction)***
- Note that if there are no stalls, the speedup is equal to the number of pipeline stages

Pipelining

- Superpipelined architectures can be achieved if the fetch-and-execute cycle can be decomposed further.
- A superscalar architecture can be achieved by the use of redundant hardware to replicate one or more stages in the pipeline.
- Superscalar and superpipelined architectures can be combined to obtain a superscalar, superpipelined computer.
- Pipelining can actually degrade performance –
  - if any of the instructions in the pipeline are a branch instruction, the prefetched instructions further in the pipeline are no longer valid and must be flushed.
  - Data dependencies can also degrade performance

Coprocessors

- A second specialized CPU to perform special instructions that are not part of the base instruction set (e.g. signal processing instructions).
  - The main processor loads certain registers with data for the coprocessor; issues an interrupt to the coprocessor, then halts itself. When the coprocessor finishes it awakens the main processor via an interrupt, and then halts itself.
- Coprocessors improve real- time performance by extending the instruction set to support faster, specialized instructions.
- Coprocessors do not improve performance because of any inherent parallelism.
- The coprocessor and its resources are a critical resource and need to be protected. Ex. registers should be saved

Ch2: Hardware Considerations

A. Basic architecture
B. Hardware interfacing (from a software or system engineering perspective)
C. CPU
D. Memory
E. I/O
F. Enhancing performance
G. Other special devices
H. Non von Neumann architectures

Other Special Devices

- ASICs
- PAL/PLA
- FPGAs
- Tranducers
- A/D converters
- D/A converters

ASICs

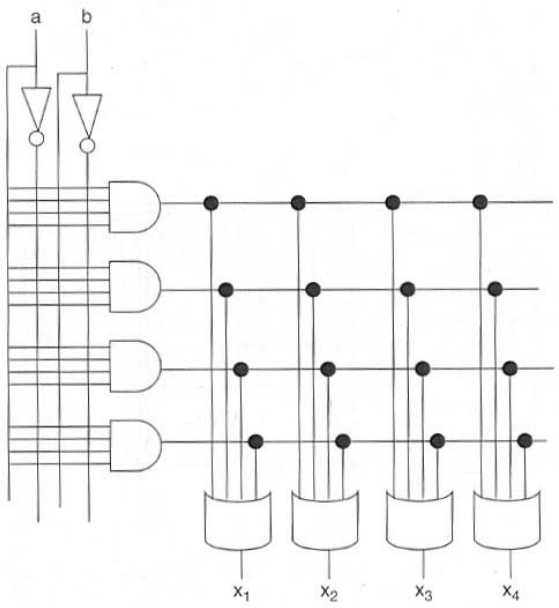
- Applications specific integrated circuit – a special purpose integrated circuit designed for one application only.
- In essence, these devices are systems-on-a-chip that can include a microprocessor, memory, I/O devices and other specialized circuitry.
- ASICs are used in many embedded applications including image processing, avionics systems, medical systems.
- Real-time design issues are the same for them as they are for most other systems.

PAL/PLA

- One-time programmable logic devices are used for special purpose functionality in embedded systems.
- Programmable array logic (PAL) – a programmable AND array followed by a fixed number input OR element. Each OR element has a certain number of dedicated product terms.
- Programmable logic array (PLA) same as PAL but the AND array is followed by a programmable width OR array. This allows the product terms to be shared between macrocells, increasing device density.
- PLA is much more flexible and yields more efficient logic, but is more expensive. PAL is faster (because it uses fewer fuses) and is less expensive.

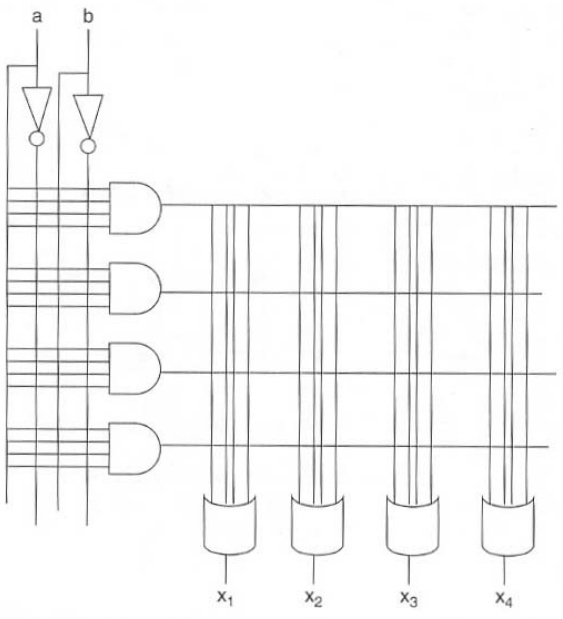
PAL

Only the external junctions are programmable. The internal ones are marked with a dot and are fixed



PLA

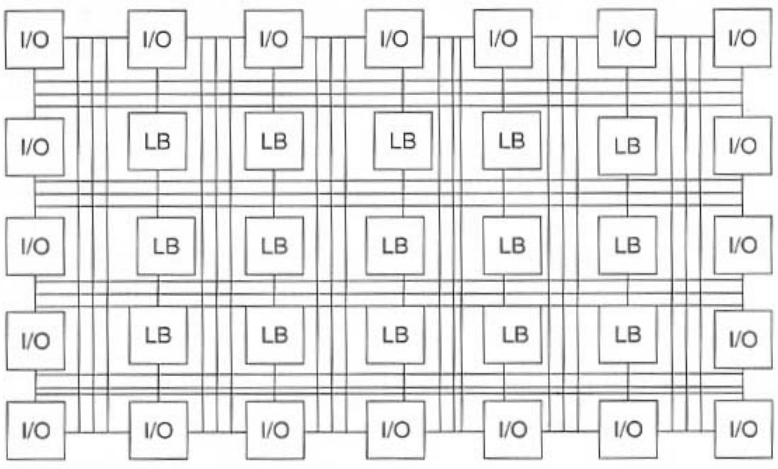
All junctions are programmable. They can be selectively fused



FPGAs

- Field programmable gate array (FPGA) – allows construction of a system on a chip with an integrated processor, memory, and I/O.
- Differs from the ASIC in that it is reprogrammable, even while embedded in the system.
- A reconfigurable architecture allows for the programmed interconnection and functionality of a conventional processor
- Algorithms and functionality are moved from residing in the software side into the hardware side.
- Widely used in embedded, mission-critical systems where fault-tolerance and adaptive functionality is essential.

FPGAs



FPGA, showing internal configurable logic blocks and periphery I/O elements

Transducers

- Transducers are generally any device that converts energy from one form to another.
- In embedded system the input is an analog signal, which must be converted to digital form by another device
- Typical transducers in embedded systems:
  - Temperature sensors
  - Accelerometers
  - Gyroscopes
  - Position resolvers

Temperature Sensors

- Temperature is an important control parameter of most embedded real-time systems.
- Most commonly used electrical temperature sensors are thermocouples, thermistors, and resistance thermometers.
- Thermocouples take advantage of the junction effect – the voltage difference generated at a junction due to the difference in the energy distribution of, of two dissimilar metals.
- Resistance thermometers rely on the increase in resistance of a metal wire as temperature increases.
- Thermistors are resistive elements made of semiconductor materials that have changing resistance properties with temperature.

Accelerometers

- Use a simple transducing function to convert the compression or stretching of a spring or the deformation of a membrane into an electrical output.
  - One mechanism takes advantage of the fact that the capacitance associated with a capacitor is a function of the gap width, which changes according to the spring or membrane deformation.
  - Another kind is a strain gage, which takes advantage of the fact that as a wire is stretched or compressed, its resistance changes. Accelerometers can be constructed using a strain gage.
- Piezoelectric effect can also be used – the phenomenon that if a crystal is compressed and the lattice structure is disrupted electrons are discharged.
  - Hence, the compression of the device due to acceleration can be measured. Piezoelectric accelerometers are widely used where miniaturization is desirable.
- Vibrating beams can also be used as both gyroscopes and accelerometers and have a high scale of miniaturization.

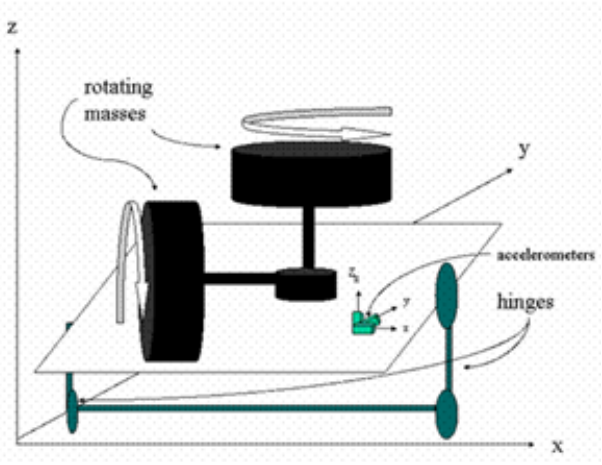
Gyroscopes

Used to sense position, provide stability

- Used in the inertial navigation of aircraft, spacecraft, robots, and automotive applications.
  - Based on the fact that a vertically oriented rotating mass will remain fixed with respect to two spatial directions.
- Mechanical gyroscopes are used to maintain a platform in a fixed position with respect to space.
- Ring laser gyros don't hold platform steady – they just sense rotation.
  - These are constructed from two concentric fiber optic loops.
  - A laser pulse is sent in opposite directions through each of the two loops. If the vehicle rotates in the direction of the loops, then one beam will travel faster than the other. Difference can be measured and the amount of rotation determined.
  - Three ring laser gyros needed to measure yaw, pitch, and roll angles.

Gyroscopes

- Stylized representation of two gyroscopes used to hold a hinged platform with three orthogonal accelerometers, fixed with respect to an inertial reference frame (not to scale).



Position Resolvers

- Sensors that provide angular measurements pertaining to the orientation or attitude of the vehicle.
- Accelerometers that are mounted orthogonally can provide enough information from which orientation can be determined via geometry.
- Other techniques take advantage of the piezoelectric effect or magnetic induction to determine position.
- Ring laser gyros can also be used for position resolution.

A/D Converters

- Analog to digital conversion converts continuous (analog) signals from various transducers and devices into discrete (digital) ones.
- Similar circuitry can be used to convert temperature, sound, pressure, and other inputs from transducers using a variety of sampling schemes to perform the conversion.
- The output of A/D circuitry is a discrete version of the time varying signal being monitored.
- The key factor in the service of A/D circuitry for time varying signals is the sampling rate (the Nyquist rate). This consideration is an inherent part of the design process for the scheduling of tasks.
  - digital samples must be taken at twice the rate of the highest frequency component of the analog signal

D/A Converters

- Digital-to-analog conversion performs the inverse function of A/D circuitry.
- Converts a discrete quantity to a continuous one.
- D/A devices are used to allow the computer to output analog voltages based on the digital version stored internally.
- Communication with D/A circuitry uses one of the three input/output methods discussed.

Ch2: Hardware Considerations

A. Basic architecture
B. Hardware interfacing (from a software or system engineering perspective)
C. CPU
D. Memory
E. I/O
F. Enhancing performance
G. Other special devices
H. Non von Neumann architectures

Non-von-Neumann Architectures

- Von-Neumann architectures are architectures that allow only one instruction or one data to occupy the bus at one time
- The generally accepted taxonomy of parallel systems was proposed by Flynn
  - The classification is based on the notion of two streams of information flow to a processor; instruction (I) and data (D)
  - These two streams can be either single (S) or multiple (M)
- As a result: **SISD; SIMD; MISD; and MIMD**

Non-von-Neumann Architectures

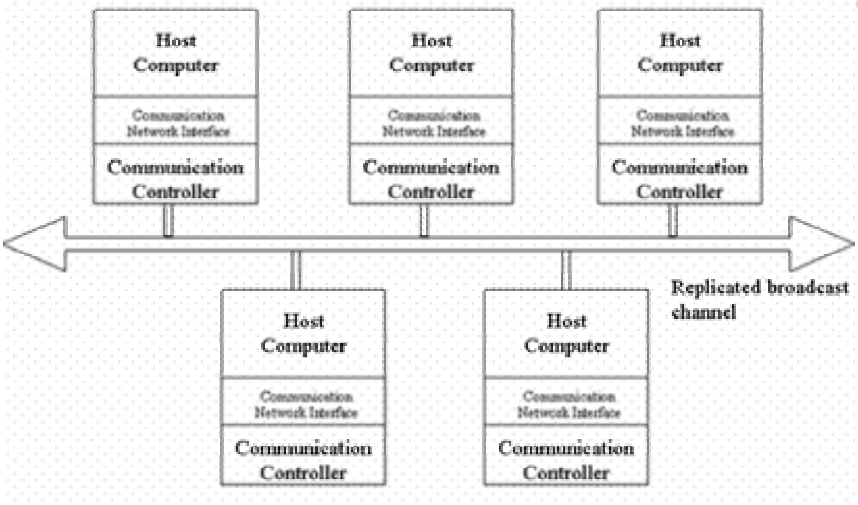
	single data stream	multiple data stream
single instruction strean	von Neumann processors	systolic processors
	RISC	wavefront processors
multiple instruction stream	pipelined architectures	dataflow processors
	VLIW processors	transputers grid computers hypercube processors

Flynn's classification for computer architectures. Examples of typical architectures

Non-von-Neumann Architectures

- Besides SISD and pipelined MISD architectures the others tend to be found only in limited real-time application in industry
- A special case of the MIMD computer is the Transport Triggered Architecture
  - a distributed heterogeneous architecture in which a number of independent von-Neumann CPUs communicate over a network and employ a time-driven processing model rather than an event-driven one.
  - An example is time triggered architecture (TTA)

Non von Neumann Architectures



Time triggered architecture with five nodes.

Non von Neumann Architectures
Time Triggered Architecture (TTA)

- TTA can be used for implementing real-time systems
- TTA models a distributed real-time system as a set of nodes interconnected by a real-time communication system
  - each node consists of a communication controller and a host computer, which are provided with a global synchronization clock with a 1 us tick duration
  - nodes use TDMA (time division multiple access) – a time slot is allocated to each node

Non von Neumann Architectures
Time Triggered Architecture (TTA)

- In TTA architecture is possible to predict the latency of all messages an the bus
  - this guarantees hard real-time message delivery
- Furthermore the latency jitter is minimal since the messages are sent at a predetermined point in time

Ch2: Hardware Considerations

A. Basic architecture
B. Hardware interfacing (from a software or system engineering perspective)
C. CPU
D. Memory
E. I/O
F. Enhancing performance
G. Other special devices
H. Non von Neumann architectures
I. Assignments

Assignments [1] (Laplante Ch2)

- Exercises
  - 2.3; 2.4; 2.5; 2.6; 2.9; 2.10; 2.12; 2.13; 2.14; 2.16;