**Chapter 1 - Introduction**

1. What is Microprogramming?
   * Microprogramming is the technique of making machine instructions generate sequences of microinstructions in accordance with a microprogram rather than initiate the desired operations directly.
2. What is the function of the control unit?
   * The control unit controls the operation of the CPU and hence the computer. The control unit contains Things like sequential logic units, registers and decoders, and memory.
3. Basic IAS Structure, MAR/MBR

Diagram, schematic

Description automatically generated

* The IAS Structure Contains Three Main Components:
  + CPU
    - Arithmetic Logic Unit (ALU)
      * Accumulator register
      * Multiply-quotient register
      * Arithmetic-logic circuits
      * Memory Buffer Register
    - Control Unit
      * Program Counter
      * Instruction Buffer Register
      * Memory Address register
      * Instruction register
      * Control circuits
  + Memory
  + I/O Devices
* MBR
  + Memory Buffer Register contains a word to be stored in memory or sent to the I/O unit or is used to receive a word from memory or from the I/O unit.
* MAR
  + Memory Address Register specifies the address in memory of the next read or write.

1. Wafers/Chips
   * A thin wafer of silicon is divided into a matrix of small areas, each a few millimeters square. The identical circuit pattern is fabricated in each area and the wafer is broken up into chips. Each chip consists of many gates and/or memory cells plus several input and output attachment points. The chip is then packaged in a housing that protects it and provides pins for attachment to devices beyond the chip.
2. Moore's Law
   * Moore’s law states that the number of transistors that can be places on a single chip double every year. The pace has slowed to doubling every 1.5 years, but has maintained the same rate since the 70’s.
   * Consequences of Moore’s Law:
     + The cost of a chip has remained virtually unchanged during this period of rapid growth in density.
     + Since the logic and memory elements are places closer together on more densely packed chips, the electrical path length is shortened, thus removing some propagation delay and increasing operating speed.
     + Computers become smaller, allowing them to be more convenient for small computer needs.
     + The closer bound the transistors, the less power requirements you will need.
     + Integrated circuit interconnections are much more reliable than solder connections. With ore circuitry on each chip, there are fewer inter-chip connections.
3. What is an embedded system?
   * An embedded system refers to the use of electronics and software within a product, rather than a general-purpose computer. Embedded systems often interact with the outside world.
4. What is cloud computing?
   * Could Computing is a model for enabling convenient on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction.
5. What is Internet of Things?
   * The IoT is primarily driven by deeply embedded devices that are low-bandwidth, low-repetition data-capture, and low-bandwidth data-usage appliances that communicate with each other and provide data via user interfaces.

**Chapter 2 - Performance Issues**

1. Performance Balance
   1. Performance balance referrers to the imbalance of speed between the processor and other components in the computer. This means that we need to balance the performance: aka an adjustment/tuning of the organization and architecture to compensate for the mismatch among the capabilities of the various components. These are a few examples of changing items in a computer to get a better performance balance.
      1. Change the DRAM interface to make it more efficient by including a cache or other buffering scheme on the DRAM chip.
      2. Reduce the frequency of memory access by incorporating increasingly complex and efficient cache structures between the processor and main memory.
      3. Increase the number of bits that are retrieved at one time by making DRAMs “wider” rather than “deeper” and by using wide bus data paths.
      4. Increase interconnect bandwidth between processors and memory by using higher speed buses and a hierarchy of buses to buffer and structure data flow.
2. Wider vs Deeper memories
   1. A “wider” memory means that more bits can be retrieved at once from the memory. This can be achieved by taking the single memory unit and splitting it into two parts. One part is called the odd bank, and the other is called the even bank. All odd addresses are stored in the odd bank, and all even addresses are stored in the even bank. In a narrow memory, one read cycle yields one byte of data. In a wide memory, one read cycle yields two bytes of data. On a wide memory, usually the even address is presented on the address bus and is changed internally to odd for the odd bank, read directly for the even bank. The number of times you can widen the memory is dependent on the size of the data bus.
   2. A “deeper” memory means that more bits can be held on the main memory at once.
3. I/O Data Rates
   1. Human interface I/O devices are the slowest, then things like scanners and printers are briefly faster, then hard discs, Wi-Fi, graphics display, and ethernet modem (highest speed) The rates of I/o devices are constantly changing, and we need to take this into account to make computer design a constantly evolving art form.
4. Power and RC Delays
   1. Power density increases with density of logic and clock speed, thus increasing heat density. Therefore we have fans to dissipate heat on computers.
   2. RC delay is the fact that the speed at which electrons flow is limited by resistance and capacitance of the metal wires connecting the devices. As components on the chip increase in size, the wire interconnects become thinner, thus increasing resistance. As the wires become closer together, this increases capacitance.
5. Amdahl's Law
   1. Amdahl’s Law calculates the overall speedup that is granted to a computer after some specific technical improvement. Amdahl’s law is given as: where *f* is the fraction of time spend doing the improvement, and is the speedup of the given improvement.
6. (Ignore Little's Law)
7. AM, GM, HM
   1. Arithmetic Mean, Geometric mean, and Harmonic mean are all part of Pythagorean Means. They offer consistent ways to evaluate performance between different computers.
      1. Arithmetic mean is good for things like comparing the execution times of different systems.
      2. Geometric mean is good for comparing normalized metrics.
      3. Harmonic Mean is good for comparing rates. Such as instruction execution rate.

* Impact of performance evaluation

1. Benchmarks and their roles
   1. Benchmarks are used to measure the capability of different computer systems. Benchmarks should have these desirable traits:
      1. It is written in a high-level language. Making it portable across different machines.
      2. It is representative of a particular kind of programming domain or paradigm such as systems programming numerical programming, or commercial programming.
      3. It can be measured easily.
      4. It has wide distribution.
   2. SPEC (System Performance Evaluation Corporation) has a benchmark suite to provide a representative test of a computer in a particular application or system programming area.
   3. Benchmarks can play huge roles in helping major businesses, or even individual users, decide on which machine is the best for them and their needs. Also, benchmarks can play a large role for research purposes in testing new architectural improvements.

**Chapter 3 - Buses**

1. Memory Read/Write cycles
   * At the beginning of each instruction cycle, the processor fetches an instruction from memory. In a typical processor, the address to fetch the next instruction from is held in the Program Counter (PC). Unless told otherwise, the program counter is always incremented after each instruction executes. The processor instruction cycle consists of a fetch instruction step, an execution instruction step, and then it loops through these two cycles over and over as shown in the figure below:

Diagram

Description automatically generated

1. Figure 3.5 (10th edition)– Example Program Execution

Diagram

Description automatically generated

1. Classes of Interrupts
   * Soft, hard, timer, I/O
     1. A soft interrupt (Software Interrupt) is invoked by the use of an INT instruction. This event immediately stops execution of the program and passes execution over to the INT handler. The INT handler is usually part of the Operating System.
     2. A hard interrupt (Hardware Interrupt) is caused by some hardware device such as request to start an I/O or a hardware failure.
     3. A Timer interrupt is generated by a timer within the processor. This allows the operating system to perform certain functions on a regular basis.
     4. An I/O interrupt is generated by an I/O controller, to signal normal completion of an operation, request service from the processor, or to signal a variety of error conditions.
2. When is a check for an interrupt made?
   * The check for an interrupt is made after the execute instruction if interrupts are enabled. See the figure below:

Diagram

Description automatically generated

1. How are multiple Interrupts handled?
   * Multiple interrupts can be handled in two different ways: sequential or nested.
     1. In sequential interrupt processing, when an interrupt occurs the processor will disable any future interrupts and handle the interrupt. Once the ISR (Interrupt Service Routine) is finished, interrupts will be re-enabled and any interrupts that occurred while in the ISR will be handled on a sequential basis. (First come first serve) This method is simple, but it fails to take into account relative priority of time sensitive interrupts that may occur.
     2. In Nested Interrupts, each interrupt has its own defined priority number. When the processor goes to handle an interrupt, it will leave interrupts enabled. While in the first interrupt’s handler if another interrupt of higher priority occurs, then the processor will stop the execution of the current interrupt and go to handle the higher priority interrupt. Once the higher interrupt is finished, the execution of the lower priority interrupt will continue.
2. Data Bus, Address Bus, Control Bus
   * A bus in general is a communication pathway that connects two or more devices. A key characteristic of a bus is that it is a shared transmission medium meaning that if multiple devices are connected to the bus and a signal is transmitted by one device, then that same signal will be made available to all devices connected to the same bus. If two devices attempt to transmit at the same time, their signals will overlap, and the resulting signal will be unusable. Therefore, it is important that we ensure that only one device on a shared bus transmits at a time.
   * Data Bus: The data bus provides a path for moving data between system modules. The number of lines is referred to as the width of the data bus. The width of the data bus is a key factor in determining overall system performance.
   * Address Bus: The address bus is used to designate the source or destination of the data on the data bus. The width of the address bus determines the maximum possible memory capacity of the system. The address bus is also used to address I/O ports. The higher order bits on the bus are used to select a particular module on the bus and the lower order bits select a memory location or I/O port within the module.
   * Control Bus: The control bus is used to control the access and the use of the data and address busses. The data transmitted on this bus contains both command and timing information for all modules. The timing signals indicate the validity of data and address information. The command signals specify operations to be performed. Typical control command signals include things like: memory write, memory read, I/O write, I/O read, Bus request, Bus grant…etc.
3. Hierarchical Bus Configuration and elements of bus design
   * Bus Design
     1. Bus Lines
        1. Dedicated: A dedicated bus line is permanently assigned either to one function or two a physical subset of computer components.
        2. Multiplexed: Using the same bus lines for multiple purposes is known as time multiplexing. If data and addresses are sharing the same bus, an address line must exist and for address use of the bus, the address line will be activated, if it is a data instruction then the address line will not be activated.
   * The operation of the bus follows this pattern for sending data to another module:
     1. Obtain the use of the bus
     2. Transfer the data on the bus
   * The operation of the bus follows this pattern for requesting data from another module:
     1. Obtain use of the bus.
     2. Transfer a request to the other module over the appropriate control lines and wait for that module to send the data.
4. Arbitration - centralized and distributed
   * In a centralized scheme, a single hardware device referred to as a bus control or arbiter, is responsible for allocating time on the bus. The device may be a separate module or part of the processor.
   * In a distributed scheme, there is no central controller. Rather, each module contains access control logic, and the modules act together to share the bus.
5. Synchronous and Asynchronous Buses
   * With synchronous timing, the occurrence of events on the bus is determined by a clock. The bus includes a clock line upon which a clock transmits a regular sequence of alternating 1s and 0s of equal duration. A single 1-0 transmission is referred to as a clock cycle or a bus cycle and defines a time slot. All other devices on the bus can read the clock line, and all events start at the beginning of a clock cycle.
   * With asynchronous timing, the occurrence of one event on a bus follows and depends on the occurrence of a previous event, not the timer.
   * Synchronous timing is simpler to implement and test. However, it is less flexible that asynchronous timing. Because all devices on a synchronous bus are tied to a fixed clock rate, the system cannot take advantage of advances in device performance. With asynchronous timing, a mixture of slow and fast devices, using older and new technology can share a bus.
6. Multi-core QPI configuration (concepts here, not details)
   * QPI allows multiple direct connections. Multiple components within the system enjoy direct pairwise connections to other components. This eliminates the need for arbitration found in shared transmission systems. QPI uses a layered protocol architecture as found in things like TCP/IP-based data networks, rather than the simple use of control signals found in shared bus arrangements. Data is not sent as a raw bit stream. Rather, data is sent as a sequence of packets, each of which includes control headers and error control codes.
7. PCI and PCI Express (concepts only, not details)
   * PCI (Peripheral component interconnect) is a popular high-bandwidth processor-independent bus that can function as a mezzanine or peripheral bus. PCI allows for much better performance with high-speed I/O subsystems. PCI is specifically designed to meet economically the I/O requirements of modern systems; it requires very few chips to implement and supports other buses attached to the PCI bus. PCI uses synchronous timing and a centralized arbitration scheme.
   * PCIe is a point-to-point interconnect scheme intended to replace bus-based schemes like PCI. Key requirement is high capacity to support the needs of higher data rate I/O devices, such as Gigabit Ethernet. Another requirement deals with the need to support time-dependent data streams. A chipset connects the processor and memory system to the PCI Express switch comprising one or more PCIe and PCIe switch devices. The switch manages multiple PCIe streams. A PCIe endpoint is an I/O device or controller that implements PCIe, such as a Gigabit Ethernet switch, a graphics or video controller, disk interface, or a communications controller. A PCIe/PCI bridge allows older PCI devices to be connected to PCIe-based systems.

**Chapter 4 - Cache**

* Access/Cycle/Transfer time
  + Access Time:
    - Random-Access Memory:
      * For random-access memory access time is given as the time it takes to perform a read or write operation.
    - Non-Random-Access Memory
      * For non-random-access memory it is the time it takes to position the read-write mechanism at the desired location. (ex. The needle, head, above the data)
  + Cycle Time (Only for Random-Access Memory):
    - Cycle Time is given as the access-time plus any additional time required before the next access can commence.
    - Additional time may be required for transient signals to die out. This is concerned with the system bus, not the processor.
  + Transfer Rate:
    - This is the rate at which data can be transferred into or out of a memory unit.
    - Random-Access Memory:
      * Transfer rate =
    - Non-Random-Access Memory
      * Average time to read or write n bits =
* Sequential/Direct/Random Access
  + Sequential Access:
    - Memory is organized into units of data, called records. Access must be made in a specific linear sequence. Stored addressing information is used to separate records and assist in the retrieval process. A shared read-write mechanism is used, and this mechanism must be moved from its current location to the desired location passing and rejecting each intermediate record. Therefore, the access time for sequential access memory is variable.
  + Direct Access:
    - Direct access involves a shared read-write mechanism. However, individual blocks or records have a unique address based on physical location. Access is accomplished by direct access to reach a general vicinity plus sequential searching, counting, or waiting to reach the final location. Access time is variable.
  + Random Access:
    - Each addressable location in memory has a unique, physically wired-in addressing mechanism. The time to access a given location is independent of the sequence of prior access and is constant. Thus, any location can be selected at random and directly addressed and accessed.
* Direct Mapping, Associative Mapping, Set Mapping
  + Be able to do problems about these
  + Direct Mapping:
    - This is the simplest mapping technique. Each block of main memory can map to only one cache line. The address for a direct mapped cache consists of a tag, line and word as seen below.



Tag = s - r

Line = r

Word = w

* + - Address Length = (s + w) bits
    - Number of addressable units = words or bytes
    - Block Size = Cache Line Size = words or bytes
    - Number of blocks in main memory =
    - Number of lines in the cache =
    - Size of the cache = words or bytes
    - Size of the tag = (s - r) bits
  + Associative Mapping:
    - This allows any block in memory to be mapped to any line in the cache. The address for a fully associative cache consists of a tag and word as seen below.



Tag = s

Word = w

* + - Address Length = (s + w) bits
    - Number of addressable units = words or bytes
    - Block Size = Cache Line size = words or bytes
    - Number of blocks in main memory =
    - Number of lines in the cache = undetermined
    - Size of the tag = s bits
  + Set-Associative Mapping:
    - This combines the benefits of both direct and associative mapping while trying to minimize their downsides. The cache consists of several sets, and each set has several lines (k for a k-way set-associative cache). The blocks of main memory are directly mapped to each set and are associatively mapped to its individual line within the set. The address for set-associative mapping consists of a tag, set, and word as seen below:



Tag = s – d

Set = d

Word = w

* + - Address length = (s + w) bits
    - Number of addressable units = words or bytes
    - Block size = Line size = words or bytes
    - Number of blocks in main memory =
    - Number of lines in a set = k
    - Number of sets = v =
    - Number of lines in the cache = m = k \* v =
    - Size of cache = words or bytes
    - Size of tag = (s – d) bits
* Write Thru/Back
  + If a write operation has been performed on a block in the cache, we must write that block to memory before evicting it from the cache. This calls for the need of a write policy.
  + Write Through
    - This is the simplest of the write policies. Every time a write operation is performed on the cache, it is also written to main memory. This generates substantial memory traffic and may create a bottleneck.
  + Write Back
    - Write updates are only performed in the cache. Therefore, portions of main memory are invalid and accesses by I/O modules can only be allowed through the cache. This also makes for complex circuitry. When a write is performed, we must set a dirty bit in the cache line to indicate that the cache data is out of sync with the memory. Therefore, when a line in the cache needs to be replaced, if it has the dirty bit set, we know we must write back to main memory before erasing that line.
* Unified and Split Cache Decision
  + Split Cache
    - It has become common to split the cache into two separate caches: one for instructions and one for data. These two caches both exist at the same level. When the processor attempts to fetch an instruction from main memory, it first consults the instruction L1 cache, and when the processor attempts to fetch data from main memory, it first consults the data L1 cache.
    - The key advantage of a split cache is that it eliminates contention for the cache between the instruction fetch/decode unit and the execution unit. This is important when the design relies on instruction pipelining.
  + Unified Cache Advantages
    - Unified caches have a higher hit ratio than split caches because it balances the load between instruction and data fetches automatically.
    - Only one cache needs to be designed and implemented.
  + The overall trend is toward split caches at the L1 and unified caches for the other levels. This helps especially with superscalar machines and the prefetching of future instructions.
* Replacement Algorithms
  + LRU Least Recently Used
    - This is the simplest replacement algorithm. This algorithm replaces the block int eh set that has been in the cache the longest with no reference to it. This can be implemented using a use-bit for a 2-way set associative cache. For a fully associative cache, this can be easily implemented with a separate list of indexes to all the lines in the cache. When a line is referenced, it moved to the front of the list. Because of its simplicity, LRU is the most popular replacement algorithm.
  + FIFO First in First Out
    - Replace the block in the set that has been in the cache the longest. This is easily implemented as a round-robin or circular buffer technique.
  + LFU Least Frequently Used.
    - Replace the block in the set that has experienced the fewest references. This can be implemented by associating a counter with each line.
  + Random
    - If all else fails, you can always replace a random line in the cache. Simulation studies have shown that random replacement provides only slightly worse performance to LFU.
* Victim Cache
  + The victim cache was originally proposed as an approach to reduce the conflict misses of direct mapped caches without affecting its fast access time. Victim caches are small, usually only 4 to 16 cache lines. This is a fully associative cache residing between direct mapped L1 cache and the next level of memory.
* Hit Ratios
* Cost Analysis for Cache and Memory
  + Cost of two-level caches example from lecture:

A white paper with writing on it

Description automatically generated with low confidence