COS2621 - Computer Organization

Summary 2013

Computer Organization and Architecture - *Designing for Performance*
8th Edition - William Stallings

v1.00 April 2013
Ron Barnard

Chapter 1 - Introduction

Summary

1.1 Organization and Architecture

Computer architecture - refers to those attributes of a system visible to a programmer, ie: those attributes that have a direct impact on the logical execution of a program.

Examples are: instruction set, number of bits used to represent various data types, I/O mechanisms, and techniques for addressing memory.

Computer organization - refers to the operational units and their interconnections that realize the architectural specifications.

Examples are: hardware details transparent to the programmer such as - control signals, interfaces between the computer and peripherals, and the memory technology used.

The distinction between architecture and organization is important. Manufacturers offer a family of computer models with the same architecture, but with differences in organization. Different models in the family have different price and performance characteristics. A particular architecture may span many years, with its organization changing with changing technology.

1.2 Structure and Function

Structure - The way in which the components are interelated.

Function - The operation of each individual component as part of the structure.

Function

There are four basic functions that a computer can perform -

- **Data processing** - Computer must be able to process data;
- **Data storage** - Must be able to store data that is being processed, as well as long-term storage;
- **Data movement** - Must be able to move data between itself and the outside world. When data are received or delivered to a device that is directly connected to the computer, the process is known as *input-output* (I/O), and the device is referred to as a *peripheral*. When data are moved over longer distances, to or from remote devices, the process is known as *data communications*.
- **Control** - Must be control of above 3 functions. Ultimately control is exercised by the programmer, who provides computer with instructions. Within the computer a *control unit* manages the computers' resources, in response to those instructions.

Structure

Four main structural components - (There may be one or more of each)

- **Central Processing Unit** (CPU) - Controls the operation of the computer and performs its data processing functions;

  - **Control unit** - Controls the operation of the CPU and hence the computer;
  - **Arithmetic and logic unit** (ALU) - Performs the computers' data processing functions;
  - **Registers** - Provides storage internal to the CPU;
  - **CPU interconnection** - Mechanism that provides for communication among the control unit, ALU, and registers.

- **Main memory** - Stores data;
- **I/O** - Moves data between the computer and its external environment;
- **System interconnection** - provides for communication among CPU, main memory, and I/O - *system bus*.

0.3 Why study Computer Organization & Architecture ? (Read Appendix B: Assembly Language - Text book)

Computer organization and architecture encompasses a broad range of design issues and concepts. A good overall understanding of these concepts will be useful in other areas of study and work.

Why study Assembly language -

- Assembly language code is often much smaller and faster than code written in a high-level language;
- We could speed up the execution time of a program if those parts of the program that are performed frequently are written in assembly language;
- With assembly language programming we have complete access to hardware. This is not possible with high-level languages;
- A basic understanding of low-level programming is essential for understanding the intricacies of compilers and operating systems;
- It is easier to understand the operation of a computer at the architctural level if we have knowledge of assembly language.

Key Terms

- **arithmetic and logic unit** (ALU) - A part of a computer that performs arithmetic operations, logic operations, and related operations.
- **central processing unit** (CPU) - Controls the operaton of a computer, and performs its data processing functions.
- **computer architecture** - refers to those attributes of a system visible to a programmer, ie: those attributes that have a direct impact on the logical execution of a program.
- **computer organization** - refers to the operational units and their interconnections that realize the architectural specifications.
- **control unit** - Controls the operation of the CPU, and hence the computer.
- **input-output** (I/O) - Refers to the movement of data between a computer and a directly attached peripheral, or other remote devices.
- **main memory** - Program addressable storage from which instructions and other data can be loaded directly into registers for subsequent execution or processing
- **processor** - See *Central Processing Unit* (CPU).
- **registers** - Provide storage internal to the CPU.
- **system bus** - Interconnection between CPU, main memory, and I/O; consisting of a number of conducting wires attached to all the other components.

--ooOoo--

Chapter 2 - Computer Evolution & Performance

Key Points

- The evolution of computers has been characterized by increasing processor speed, decreasing component size, increasing memory size, and increasing I/O capacity and speed.
- One factor responsible for the great increase in processor speed is the shrinking size of microprocessor components; this reduces the distance between components and hence increases speed. However, the true gains in speed in recent years have come from the organization of the processor, including heavy use of pipelining and parallel execution techniques and the use of speculative execution techniques (tentative execution of future instructions that might be needed). All of these techniques are designed to keep the processor busy as much of the time as possible.
- A critical issue in computer system design is balancing the performance of the various elements so that gains in performance in one area are not handicapped by a lag in other areas. In particular, processor speed has increased more rapidly than memory access time. A variety of techniques is used to compensate for this mismatch, including caches, wider data paths from memory to processor, and more intelligent memory chips.

Summary

2.1 A Brief History of Computers (Mainly read only, except items below)

The von Neumann Machine

Stored-program concept - The programming process could be facilitated if the program could be represented in a form suitable for storing in memory alongside the data. Then, a computer could get its instructions by reading them from memory, and a program could be set or altered by setting the values of a portion of memory.

General structure of IAS (Princeton Institute for Advanced Studies) computer -

- A main memory, which stores both data and instructions;
- An arithmetic and logic unit (ALU) capable of operating on binary data;
- A control unit, which interprets the instructions in memory and causes them to be executed;
- Input and output (I/O) equipment operated by the control unit.

Moores' Law

Moore observed that the number of transistors that could be put on a single chip was doubling every year and correctly predicted that this pace would continue into the near future. To the surprise of many, including Moore, the pace continued year after year and decade after decade. The pace slowed to a doubling every 18 months in the 1970's, but has sustained that rate ever since.

Consequences of Moores' Law -

- The cost of a chip has remained virtually unchanged during this period of rapid growth in density. This means that the cost of computer logic and memory circuitry has fallen at a dramatic rate;
- Because logic and memory elements are placed closer together on more densely packed chips, the electrical path length is shortened, increasing operating speed;
- The computer becomes smaller, making it more convenient to place in a variety of environments;
- There is a reduction in cooling and power requirements;
- The interconnections on the integrated circuit are much more reliable than solder connections. With more circuitry on each chip, there are fewer interchip connections.

Characteristics of a family of computers

- **Similar or identical instruction set** - In many cases, the exact same set of machine instructions is supported on all members of the family. So, a program that executes on one machine will also execute on any other. In some cases, the lower end of the family has an instruction set that is a subset of that of the top end of the family. This means that programs can move up, but not down.
- **Similar or identical operating system** - The same basic operating system is available for all family members. In some cases, additional features are added to higher-end members.
- **Increasing speed** - The rate of instruction execution increases in going from lower to higher family members.
- **Increasing number of I/O ports** - The number of I/O ports increases in going from lower to higher famly members.
- **Increasing memory size** - The size of main memory increases in going from lower to higher family members.
- **Increasing cost** - At a given point in time, the cost of a system increases in going from lower to higher family members.

2.2 Designing for Performance

Hardware and software are generally logically equivalent. They are frequently capable of performing the same function, and we need to decide which functions to implement in hardware and which in software. Cost is usually an important consideration.

Hardware offers speed, but not flexibility, whereas software offers flexibility but less speed. This is known as 'trade-off', and is a design decision, there is no special reason why a specific operation should be implemented in hardware or software.

2.3 The Evolution of the Intel x86 Architecture

Pentium family	Differences
Pentium	Intel introduced superscalar techniques, which allow multiple instructions to execute in parallel.
Pentium Pro	Continued superscalar organization, with register renaming, branch prediction, data flow analysis, and speculative execution.
Pentium II	Incorporated Intel MMX technology, which is designed to process video, audio, and graphics data efficiently.
Pentium III	Incorporates additional floating-point instructions to support 3D graphics software.
Pentium 4	Includes additional floating-point and other enhancements for multimedia.
Core	First Intel x86 microprocessor with a dual core, ie: the implementation of two processors on a single chip.
Core 2	Extends the architecture to 64 bits. The Core 2 Quad provides four processors on a single chip.

2.4 Embedded Systems and the ARM (Read only)

Key Terms

- **accumulator** (AC) - The name of the CPU register in a single-address instruction format. The accumulator, or AC, is implicitly one of the operands for the instruction.
- **Amdahls' law** - Proposed by Gene Amdahl, it deals with the potential speedup of a program using multiple processors compared to a single processor. A speedup in one aspect of the technology or design does not result in a corresponding improvement in performance.
- **arithmetic & logic unit** (ALU) - A part of a computer that performs arithmetic operations, logic operations, and related operations.
- **benchmark** - A program, defined in a high-level language, that provides a representative test of a computer in a particular application or system programming area.
- **chip** - Consists of many gates and / or memory cells plus a number of input and output attachment points, fabricated on a thin *wafer* of silicon, packaged in a housing that protects it and provides pins for attachment to devices. See *integrated circuit*.
- **data channel** - is an independent I/O module with its own processor and its own instruction set. The CPU does not execute detailed I/O instructions, these are executed by the data channel under the control of the CPU. This arrangement relieves the CPU of a considerable processing burden.
- **embedded system** - The use of electronics and software within a product, designed to perform a dedicated function, as opposed to a general-purpose computer.
- **execute cycle** - That portion of the execution cycle during which the CPU performs the operation specified by the instruction opcode.
- **fetch cycle** - That portion of the execution cycle during which the CPU fetches from memory the instruction to be executed.
- **input-output** (I/O) - Pertaining to either input or output, or both. Refers to the movement of data between a computer and a directly attached peripheral.
- **instruction buffer register** (IBR) - Temporarily holds the right-hand instruction from a word in memory.
- **instruction cycle** - The processing performed by a CPU to execute a single instruction.
- **instruction register** (IR) - A register that is used to hold an instruction for interpretation.
- **instruction set** - The collection of different machine code instructions that a processor can execute.
- **integrated circuit** (IC) - A tiny piece of solid material, such as silicon, upon which is etched or imprinted a collection of electronic components and their interconnections.
- **main memory** - Program addressable storage from which instructions and other data can be loaded directly into registers for subsequent execution or processing.
- **memory address register** (MAR) - A register, in a processing unit, that contains the address of the storage location being accessed.

- **memory buffer register** (MBR) - A register that contains data read from memory, or data to be written to memory.

- **microprocessor** - A processor whose elements have been minaturized into one or a few integrated circuits.

- **multicore** - The use of multiple processors on one chip, increases performance without increasing the clock rate.

- **multiplexer** - A combinational circuit that connects multiple inputs to a single output. At any time, only one of the inputs is selected to be passed to the output.

- **opcode** - Abbreviated form for *operation code*.

- **original equipment manufacturer** (OEM) - Manufacturer that acquires a product and includes that product into its own products.

- **program control unit** - CPU of an IAS computer is made up of *Arithmetic Logic Unit* (ALU) and *program control unit*.

- **program counter** (PC) - Instruction address register.

- **SPEC** (System Performance Evaluation Corporation) - An industry consortium, defined and maintains a collection of benchmark suites (eg: SPEC CPU2006).

- **stored program computer** - Program stored in a suitable format in memory alongside the data.

- **upward compatible** - Family of computers - Lower end instruction set / operating system is a subset of higher end, can move up, but not down.

- **von Neumann machine** - Stored program computer which includes - main memory, arithmetic logic unit (ALU), control unit, and I/O equipment.

- **wafer** - A thin *wafer* of silicon is divided into a matrix of small areas, each with an identical circuit pattern. The *wafer* is then broken up into *chips*, which are then packaged in a protective housing.

- **word** - An ordered set of bytes or bits that is the normal unit in which information may be stored, transmitted, or operated on within a given computer. Typically, if a processor has a fixed-length instruction set, then the instruction length equals the word length.

--ooOoo--

Chapter 3 - A Top-Level View of Computer Function & Interconnection

Key Points

- An instruction cycle consists of an instruction fetch, followed by zero or more operand fetches, followed by zero or more operand stores, followed by an interrupt check (if interrupts are enabled).

- The major computer system components (processor, main memory, I/O modules) need to be interconnected in order to exchange data and control signals. The most popular means of interconnection is the use of a shared system bus consisting of multiple lines. In contemporary systems, there typically is a hierarchy of buses to improve performance.

- Key design elements for buses include arbitration (whether permission to send signals on bus lines is controlled centrally or in a distributed fashion); timing (whether signals on the bus are synchronized to a central clock or are sent asynchronously based on the most recent transmission); and width (number of address lines and number of data lines).

Summary

3.1 Computer Components

von Neumann Architecture - is based on three key concepts:

- Data and instructions are stored in a single read-write memory;
- The contents of this memory are addressable by location, without regard to the type of data contained there;
- Execution occurs in a sequential fashion (unless explicitly modified) from one instruction to the next.

3.2 Computer Function

The basic function performed by a computer is execution of a program, which consists of a set of instructions stored in memory. In its simplest form, instructon processing consists of 2 steps: The processor reads (*fetches*) instructions from memory one at a time, and *executes* each instruction. The processing required for a single instruction is called an *instruction cycle*. The 2 steps are referred to as the *fetch cycle* and the *execute cycle*.

Instruction fetch and execute

At the beginning of each instruction cycle, the processor fetches an instruction from memory. A register, called the program counter (PC), holds the address of the instruction to be fetched next. The processor always increments the PC after each instruction, unless told otherwise, so that it will fetch the next instruction in sequence.

The fetched instruction is loaded into a register in the processor known as the instruction register (IR). The processor interprets the instruction and performs the required action, which falls into four categories:

- **Processor - memory** - Data may be transferred from processor to memory, or from memory to processor;
- **Processor - I/O** - Data may be transferred to or from a peripheral device by transferring between the processor and an I/O module;
- **Data processing** - The processor may perform some arithmetic or logic operation on data;
- **Control** - An instruction may specify that the sequence of execution be altered.

An instructions' execution may involve a combination of these actions.

Interrupts

Virtually all computers provide a mechanism by which other modules (I/O, memory) may interrupt the normal processing of the processor. Interrupts are provided primarily as a way to improve processing efficiency.

Classes of Interrupts -

- **Program** - generated by some condition that occurs as a result of an instruction execution. Eg: arithmetic overflow, divide by zero.
- **Timer** - generated by a timer within the processor. Allows the operating system to perform certain functions on a regular basis.
- **I/O** - generated by an I/O controller, to signal normal completion of an operaation, or an error condition.
- **Hardware failure** - generated by a power failure, or memory parity error.

Interrupts and the instruction cycle

With interrupts, the processor can be engaged in executing other instructions while an I/O operation is in progress. From the point of view of the user program, an interrupt is just that: an interruption of the normal sequence of execution. The user program does not have to contain any special code to accommodate interrupts; the processor and the operating system are responsible for suspending the user program and then resuming it at the same point.

To accommodate interrupts, an *interrupt cycle* is added to the *instruction cycle*. If no interrupts are pending, the processor fetches the next instruction of the current program. If an interrupt is pending, the processor:

- suspends execution of the current program and saves the address of the next instruction to be executed (contents of the program counter);
- sets the program counter to the starting address of an *interrupt handler* routine.

The processor then proceeds to the fetch cycle, and fetches the first instruction in the interrupt handler program. When the interrupt handler routine is completed, the processor resumes execution of the user program at the point of interruption.

Multiple interrupts

Multiple interrupts can occur and can be handled in two ways -

- *Disabled interrupt* means that the processor can ignore the interrupt request signal;
- *Priorities* for interrupts allow an interrupt of higher priority to cause a lower-priority interrupt handler to be interrupted itself.

3.3 Interconnection Structures

A computer consists of a set of components or modules of three basic types (processor, memory, I/O) that communicate with each other. There must be paths for connecting the modules. The collection of paths is called the *interconnection structure*.

The *interconnection structure* must support the following types of transfers -

- **Memory to processor** - The processor reads an instruction or a unit of data from memory;
- **Processor to memory** - The processor writes a unit of data to memory;
- **I/O to processor** - The processor reads data from an I/O device via an I/O module;
- **Processor to I/O** - The processor sends data to the I/O device;
- **I/O to or from memory** - An I/O module is allowed to exchange data directly with memory, without going through the processor, using direct memory access (DMA).

The most common interconnection structure is the bus, and various multiple-bus structures.

3.4 Bus Interconnection

A *bus* is a communication pathway connecting two or more devices. A key characteristic of a bus is that it is a shared transmission medium. Multiple devices connect to a bus, and a signal transmitted by any one device is available for reception by all other devices attached to the bus. Only one device can successfully transmit at a time.

Typically, a bus consists of multiple communication pathways, or lines. Several lines of a bus can be used to transmit binary digits simultaneously, in parallel. Computer systems contain a number of different buses that provide pathways between components at various levels of the computer system hierarchy. A bus that connects major components (processor, memory, I/O) is called a *system bus*.

Bus structure

A system bus consists, typically, of from about 50 to hundreds of separate lines. Each line is assigned a particular meaning or function. On any bus the lines can be classifies into three functional groups -

- **data lines** - provide a path for moving data among system modules. These lines are collectively known as the *data bus;*
- **address lines** - are used to designate the source or destination of the data on the data bus;
- **control lines** - are used to control access to, and the use of, the data and address lines.

3.5 PCI (Read only up to, but not including, 'PCI Commands')

The *peripheral component interconnect* (PCI) is a popular high-bandwidth, processor independent bus that can function as a mezzenine, or peripheral bus.

-ooOoo-

Key Terms

- **address bus** - That portion of a system bus used for the transfer of an address. Typically, the address identifies a main memory location or an I/O device.
- **asynchronous timing** - A technique in which the occurrence of one event on a bus follows and depends on the occurrence of a previous event.
- **bus** - A shared communications path consisting of one or a collection of lines. In some computer systems, CPU, memory, and I/O components are connected by a common bus. Since the lines are shared by all components, only one component at a time can successfully transmit.
- **bus arbitration** - The process of determining which competing bus master will be permitted access to the bus.
- **bus width** - The number of lines that the bus consists of.
- **centralized arbitration** - n/a
- **data bus** - That portion of a system bus used for the transfer of data.
- **disabled interrupt** - A condition, usually created by the CPU, during which the CPU will ignore interrupt request signals of a specified class.
- **distributed arbitration** - n/a
- **instruction cycle** - The processing performed by a CPU to execute a single instruction.
- **instruction execute** - Perform instruction fetched from memory.
- **instruction fetch** - Read instruction from its memory location into the processor.
- **interrupt** - A suspension of a process, such as the execution of a computer program, caused by an event external to that process, and performed in such a way that the process can be resumed. Synonymous with *interruption*.
- **interrupt handler** - a program to service a particular I/O device.
- **interrupt service routine** -
- **memory address register** (MAR) - A register, in a processing unit, that contains the address of the storage location being accessed.
- **memory buffer register** (MBR) - A register that contains data read from memory or data to be written to memory.
- **peripheral component interconnect** (PCI) - s a popular high-bandwidth, processor independent bus that can function as a mezzenine, or peripheral bus.
- **synchronous timing** - A technique in which the occurrence of events on a bus is determined by a clock. The clock defines equal-width time slots, and events begin only at the beginning of a time slot.
- **system bus** - A bus used to interconnect major computer components (CPU, memory, I/O).

--ooOoo--

Chapter 4 - Cache Memory

Key Points

- Computer memory is organized into a hierarchy. At the highest level (closest to the processor) are the processor registers. Next comes one or more levels of cache. When multiple levels are used, they are denoted L1, L2, and so on. Next comes main memory, which is usually made out of dynamic random-access memory (DRAM). All of these are considered internal to the computer system. The hierarchy continues with external memory, with the next level typically being a fixed hard disk, and one or more levels below that consisting of removable media such as optical disks and tape.

- As one goes down the memory hierarchy, one finds decreasing cost/bit, increasing capacity, and slower access time. It would be nice to use only the fastest memory, but because that is the most expensive memory, we trade off access time for cost by using more of the slower memory. The design challenge is to organize the data and programs in memory so that the accessed memory words are usually in the faster memory.

- In general, it is likely that most future accesses to main memory by the processor will be to locations recently accessed. So the cache automatically retains a copy of some of the recently used words from the DRAM. If the cache is designed properly, then most of the time the processor will request memory words that are already in the cache.

Summary

4.1 Computer Memory System Overview

Characteristics of Memory Systems

Location
Internal (e.g. processor registers, main memory, cache)
External (e.g. optical disks, magnetic disks, tapes)

Capacity
Number of words (8, 16, or 32 bits) - Used for internal memory.
Number of bytes (1 byte = 8 bits) - Used for external & internal memory.

Unit of Transfer
Word
Block - Units larger than a word.

Access Method
Sequential - Access in a specific linear sequence.

Direct - Access is accomplished by direct access to reach a general vicinity, plus sequential searching, counting, or waiting to reach the final location.

Random - Any location can be selected at random and directly addressed and accessed.

Associative - A random access type of memory that enables one to make a comparison of desired bit locations within a word for a specified match, and to do this for all words simultaneously. A word is retrieved based on a portion of its contents rather than its address.

Performance

Access time (latency) - For random-access memory this is the tme it takes to perform a read or write operation. For non-random-access memory, access time is the time it takes to position the read-write mechanism at the desired location.

Cycle time - Primarily applied to random-access memory, consists of access time plus any additional time required before a second access can commence. Concerned with system bus, not the processor.

Transfer rate - Rate at which data can be transferred into or out of a memory unit.

Physical Type

Semiconductor
Magnetic
Optical
Magneto-optical

Physical Characteristics

Volatile / nonvolatile - Volatile: data decays or is lost when power is lost. Non-volatile: Data remains until deliberately changed.
Erasable / nonerasable - Non-erasable memory cannot be altered, except by destroying it.

Organization

Memory modules - A key issue for random-access memory. The physical arrangement of bits to form words.

The Memory Hierarchy

There is a trade-off among the three key characteristics of memory: capacity, access time, and cost. The following relationships hold -

- Faster access time, greater cost per bit;
- Greater capacity, smaller cost per bit;
- Greater capacity, slower access time.

The dilemma facing the designer is clear. The designer would like to use memory technologies that provide for large-capacity memory, both because the capacity is needed and because the cost per bit is low. However, to meet performance requirements, the designer needs to use expensive, relatively lower-capacity memories with short access times.

The way out of this dilemma is not to rely on a single memory component or technology, but to employ a **memory hierarchy** -

- Inboard memory - Registers, cache, main memory;
- Outboard storage - Magnetic disk, CD-ROM, CD-RW, DVD-RW, DVD-RAM;
- Off-line storage - Magnetic tape.

As one goes down the hierarchy, the following occur -

- Decreasing cost per bit;
- Increasing capacity;
- Increasing capacity time;
- Decreasing frequency of access of the memory by the processor.

So, smaller, more expensive, faster memories are supplemented by larger, cheaper, slower memories. The key to the success of this organization is "decreasing frequency of access", which is generally valid.

The basis for this validity is a principle known as **locality of reference** -

During the course of execution of a program, memory references by the processor, for both instructions and data, tend to cluster. Programs typically contain a number of iterative loops and subroutines. Once a loop or subroutine is entered, there are repeated references to a small set of instructions. Similarly, operations on tables and arrays involve access to a clustered set of data words. Over a long period of time, the clusters in use change, but over a short period of time, the processor is primarily working with fixed clusters of memory references.

Accordingly, it is possible to organize data across the hierarchy such that the percentage of accesses to each successively lower level is substantially less than that of the level above.

4.2 Cache Memory Principles

Cache memory is intended to give memory speed approaching that of the fastest memories available, and at the same time provide a large memory size at the price of less expensive types of semiconductor memories. There is a relatively large and slow main memory together with a smaller, faster cache memory. The cache contains a copy of portions of main memory.

When the processor attempts to read a word of memory, a check is made to determine if the word is in the cache. If so, the word is delivered to the processor. If not, a block of main memory, consisting of some fixed number of words, is read into the cache and then the word is delivered to the processor. Because of the phenomenon of locality of reference, when a block of data is fetched into the cache to satisfy a single memory reference, it is likely that there will be future references to that same memory location or to other words in the block.

4.3 Elements of Cache Design (Read only, study last page)

Unified vs Split Caches

Originally many designs consisted of a single cache used to store references to both data and instructions. More recently it has become common to split the cache into two: one dedicated to instructions and one dedicated to data.

Advantages of a unified cache -

- has a higher hit rate than split caches because it balances the load between the instruction and data fetches automatically;
- Only one cache needs to be designed and implemented.

Trend is towards split caches.

Key advantage of split cache design -

- It eliminates contention for the cache between the instruction fetch / decode unit and the execution unit. Cache contention can degrade performance, split cache overcomes this difficulty.

4.4 Pentium 4 Cache Organization

There are 3 caches, 2 of which are on-chip -

1. L1 instruction cache (on-chip). 12K in size and holds micro operations. Sits between the instruction decode logic and execution core.
2. L1 data cache (on-chip). 8Kb, 4-way set associative organisation. Uses a write-block policy.
3. L2 cache of 256 Kb. Feeds both L1 data and instruction caches. The organisation of L2 is 8-way set-associative.

4.5 ARM Cache Organization (Read only)

-ooOoo-

Key Terms

- **access time** - For random-access memory, this is the time it takes to perform a read or write operation, that is, the time from the instant that an address is presented to the memory to the instant that data have been stored or made available for use. For non-random-access memory, access time is the time it takes to position the read–write mechanism at the desired location.
- **associative mapping** - n/a
- **cache** - A relatively small fast memory interposed between a larger, slower memory and the logic that accesses the larger memory. The cache holds recently accessed data, and is designed to speed up subsequent access to the same data.
- **cache hit** -
- **cache line** - A block of data associated with a cache tag and the unit of transfer betweeen cache and memory.
- **cache memory** - A special buffer storage, smaller and faster than main storage, that is used to hold a copy of instructions and data in main storage that are likely to be needed next by the processor and that have been obtained automatically from main storage.
- **cache miss** -
- **cache set** -
- **data cache** - Cache dedicated to data in a split cache system.
- **direct access** - The capability to obtain data from a storage device, or to enter data into a storage device, in a sequence independent of their relative position, by means of addresses that indicate the physical location of the data.
- **direct mapping** - n/a
- **high-performance computing** (HPC) - A research area dealing with suercomputers and the software that runs on supersomputers. The emphasis is on scientific, applications which may involve heavy use of vector and matrix computation, and parallel algorithms.
- **hit ratio** -
- **instruction cache** - Cache dedicated to instructions in a split cache system.
- **L1 cache** - Level 1 - Internal (on chip) cache.
- **L2 cache** - Level 2 - External (off chip) cache.
- **L3 cache** - Level 3
- **locality** -
- **logical cache** - also known as a **virtual cache**, stores data using *virtual addresses*.

- **memory hierachy** - Arrangement of different types of memory: Inboard, Outboard, Off-line.
- **multilevel cache** - Cache with L1 on-chip cache, and L2 external cache.
- **physical cache** - stores data using main memory *physical addresses*.
- **random access** - Each addressable location in memory has a unique physical address. So, any location can be selected at random and directly addressed and accessed. Main memory and some cache systems are random access.
- **replacement algorithm** -
- **sequential access** - Access must be made in a specific linear sequence.
- **set-associative mapping** -
- **spatial locality** -
- **split cache** - Cache split into two: one dedicated to instructions and one dedicated to data.
- **tag** -
- **temporal locality** -
- **unified cache** - Single cache for data and instructions.
- **virtual cache** - See *logical cache*.
- **write back** -
- **write once** -
- **write through** -

--ooOoo--

Chapter 5 - Internal Memory Technology

Key Points

- The two basic forms of semiconductor random access memory are dynamic RAM (DRAM) and static RAM (SRAM). SRAM is faster, more expensive, and less dense than DRAM, and is used for cache memory. DRAM is used for main memory.
- Error correction techniques are commonly used in memory systems. These involve adding redundant bits that are a function of the data bits to form an error-correcting code. If a bit error occurs, the code will detect and, usually, correct the error.

Summary

5.1 Semiconductor Main Memory

Organization

The basic element of a semiconductor memory is the memory cell. Common properties are -

- They exhibit two stable (or semistable) states, which can be used to represent binary 1 an 0;
- They are capable of being written into (at least once), to set the state;
- They are capable of being read to sense state.

DRAM and SRAM

Random-access memory (RAM) - Individual words of memory are directly accessible;
Possible to read data from memory, and to write new data into memory;
RAM is volatile, must be provided with a constant power supply.

<table>
<tr><th>Memory Type</th><th>Category</th><th>Erasure</th><th>Write Mechanism</th><th>Volatility</th></tr>
<tr><td>Random-access memory (RAM)</td><td>Read-write memory</td><td>Electrically, byte-level</td><td>Electrically</td><td>Volatile</td></tr>
<tr><td>Read-only memory (ROM)</td><td rowspan="2">Read-only memory</td><td rowspan="2">Not possible</td><td>Masks</td><td rowspan="5">Nonvolatile</td></tr>
<tr><td>Programmable ROM (PROM)</td><td rowspan="4">Electrically</td></tr>
<tr><td>Erasable PROM (EPROM)</td><td rowspan="3">Read-mostly memory</td><td>UV light, chip-level</td></tr>
<tr><td>Electrically Erasable PROM (EEPROM)</td><td>Electrically, byte-level</td></tr>
<tr><td>Flash memory</td><td>Electrically, block-level</td></tr>
</table>

Dynamic RAM (DRAM)

A dynamic RAM (DRAM) is made with cells that store data as charge on capacitors. The presence or absense of charge in a capacitor is interpreted as a binary 1 or 0. Dynamic RAM's require periodic charge refreshing to maintain data storage. The term *dynamic* refers to this tendency of the stored charge to leak away, even with power continuously applied.

Although the DRAM cell is used to store a single bit (0 or 1), it is essentially an analog device. The capacitor can store any charge within a range; a threshold value determines whether the charge is interpreted as 1 or 0.

Static RAM (SRAM)

A static RAM (SRAM) is a digital device that uses that uses the same logic elements used in a processor. Binary values are stored using traditional flip-flop logic gate configurations. A static RAM will hold its data as long as power is supplied to it.

SRAM vs DRAM

- Both are volatile, power must be supplied continuously to preserve the bit values;
- A DRAM is smaller and simpler than an SRAM;
- A DRAM is therefore more dense (Smaller cells = more cells per unit area), and less expensive than an SRAM;
- A DRAM requires the supporting refresh circuitry. For larger memories, the fixed cost of the refresh circuitry is more than compensated for by the smaller variable cost of DRAM cells. So, DRAM's tend to be favoured for large memory requirements;
- SRAM's are generally faster tham DRAM's;
- Because of these characteristics, SRAM is used for cache memory, and DRAM is used for main memory.

Types of ROM

Read-only memory (ROM) - contains a permanent pattern of data that cannot be changed.

- A ROM is non-volatile, no power source is required to read a ROM.
- While it is possible to read a ROM, it is not possible to write new data into it.
- ROM's are used for -
  - Microprogramming;
  - Library subroutines for frequently wanted functions;
  - System programs;
  - Function tables.
- The advantage of ROM is that the data or program is permanently in main memory, and need never be loaded from a secondary storage device.

Programmable ROM (PROM) - is a less expensive alternative when only a small number of ROM's with a particular memory content are needed. PROM is non-volatile and may be written into only once. The writing process is performed electrically, using special equipment, and may be performed by a supplier or customer at a time later than the original chip fabrication. PROM's provide flexibility and convenience.

Another variation on read-only memory is the **read-mostly memory**, which is useful for applications in which read operations are far more frequent than write operations, but for which non-volatile storage is required.

There are three common forms of read-mostly memory -

- **Erasable programmable read-only memory** (EPROM) - is read and written electrically, as with PROM. Before a write operation all the storage cells must be erased by exposure of the chip to ultraviolet radiation. EPROM can be altered multiple times, and holds its data virtually indefinitely. It is more expensive than PROM, but has the advantage of multiple update capability.

- **Electically erasable programmable read-only memory** (EEPROM) - can be written into at any time without erasing prior contents, only the byte or bytes addressed are updated. EEPROM combines the advantage of non-volatility with the flexibility of being updatable in place, using ordinary bus control, address, and data lines. More expensive than EPROM, and also less dense, supporting fewer bits per chip.

- **Flash memory** - is intermediate between EPROM and EEPROM in both cost and functionality. Flash uses an electrical erasing technology, like EEPROM. An entire flash memory can be erased in one, or a few, seconds which is much faster than EPROM. It is possible to erase blocks of memory rather than an entire chip, but there is no byte-level erasure.

5.2 Error Correction

A semiconductor memory system is subject to errors -

- **Hard failure** - is a permanent physical defect, which can be caused by harsh environment, manufacturing defects, and wear.

- **Soft error** - is a random, non-destructive event that alters the contents of one or more memory cells without damaging the memory. Can be caused by power supply problems or alpha particles (radioactive decay).

Hard and soft errors are undesirable, and most modern main memory systems include logic for both detecting and correcting errors.

Hamming code - simplest of the error-correcting codes. Uses *parity bits* for even parity. Parity bits can be checked. If there is an error, the appropriate bit is changed.

5.3 Advanced DRAM Organization

The basic building block of main memory remains the DRAM chip. There are a number of enhancements to the basic DRAM architecture -

Synchronous DRAM (SDRAM) - one of the most widely used forms of DRAM, is synchronized to an external clock, unlike traditional DRAM which is asynchronous. Prevents delays in reading or writing data. There is an enhanced version of SDRAM, known as double data rate SDRAM (DDR-SDRAM), which overcomes the once-per-cycle limitation of SDRAM, and can send data to the processor twice per clock cycle.

Rambus DRAM (RDRAM) - is a competitor to SDRAM. Vertical package, with all pins on one side, and a high speed bus.

Cache DRAM (CDRAM) - integrates a small SRAM cache onto a generic DRAM chip.

-ooOoo-

Key Terms

- **cache DRAM** (CDRAM) - integrates a small SRAM cache onto a generic DRAM chip.
- **dynamic RAM** (DRAM) - A RAM whose cells are implemented using capacitors. A dynamic RAM will gradually lose its data unless it is periodically refreshed.
- **electrically erasable programmable ROM** (EEPROM) - is a read-mostly memory that can be written into at any time without erasing prior contents; only the byte or bytes addressed are updated.
- **erasable programmable ROM** (EPROM) - is read and written electrically, as with PROM. However, before a write operation, all the storage cells must be erased to the same initial state by exposure of the packaged chip to ultraviolet radiation.
- **error correcting code** (ECC) - A code in which each character or signal conforms to specific rules of construction so that deviations from these rules indicate the presence of an error, and in which some or all of the detected errors can be corrected automatically.
- **error correction** - most modern main memory systems include logic for both detecting and correcting errors.
- **flash memory** - form of semiconductor memory, so named because of the speed with which it can be reprogrammed. Intermediate between EPROM and EEPROM in both cost and functionality.
- **Hamming code** - The simplest of the error-correcting codes. Uses a *parity bit* with even parity.
- **hard failure** - is a permanent physical defect so that the memory cell or cells affected cannot reliably store data but become stuck at 0 or 1, or switch erratically between 0 and 1. Hard errors can be caused by harsh environmental abuse, manufacturing defects, and wear.
- **nonvolatile memory** - Memory whose contents are stable and do not require a constant power source.
- **programmable ROM** (PROM) - Semiconductor memory whose contents may be set only once. The writing process is performed electrically and be performed by the user at a time later thanoriginal chip fabrication.
- **RamBus DRAM** (RDRAM) - High speed alternative to the SDRAM.
- **read-mostly memory** - Another variation on read-only memory is the read-mostly memory, which is useful for applications in which read operations are far more frequent than write operations, but for which nonvolatile storage is required. There are three common forms of read-mostly memory: EPROM, EEPROM, and flash memory.
- **read-only memory** (ROM) - Semiconductor memory whose contents cannot be altered, except by destroying the storage unit. Nonerasable memory.
- **semiconductor memory** - the use of semiconductor chips for main memory is almost universal.
- **single-error-correcting code** (SEC) - n/a
- **single-error-correcting, double-error-detecting** (SED-DED) - n/a

- **soft error** - is a random, nondestructive event that alters the contents of one or more memory cells without damaging the memory. Soft errors can be caused by power supply problems or alpha particles.

- **static RAM** (SRAM) - A RAM whose cells are implemented using flip-flops. A static RAM will hold its data as long as power is supplied to it; no periodic refresh is required.

- **synchronous DRAM** (SDRAM) - the SDRAM exchanges data with the processor synchronized to an external clock signal, and running at the full speed of the processor/memory bus without imposing wait states. (Unlike the traditional DRAM, which is asynchronous).

- **syndrome** - the result of doing a bit-by-bit comparison of 2 inputs by taking the XOR of the 2 inputs. Used in the *Hamming code*.

- **volatile memory** - A memory in which a constant electrical power source is required to maintain the contents of memory. If the power is switched off, the stored information is lost.

--ooOoo--

Chapter 6 - External Memory

Key Points

- Magnetic disks remain the most important component of external memory. Both removable and fixed, or hard, disks are used in systems ranging from personal computers to mainframes and supercomputers.
- To achieve greater performance and higher availability, servers and larger systems use RAID disk technology. RAID is a family of techniques for using multiple disks as a parallel array of data storage devices, with redundancy built in to compensate for disk failure.
- Optical storage technology has become increasingly important in all types of computer systems. While CD-ROM has been widely used for many years, more recent technologies, such as writable CD and DVD, are becoming increasingly important.

Summary

6.1 Magnetic Disk

A disk is a circular platter constructed of nonmagnetic material, called the substrate, coated with a magnetizable material. Traditionally, the substrate has been an aluminum or aluminum alloy material. More recently, glass substrates have been introduced. The glass substrate has a number of benefits, including the following:

- Improvement in the uniformity of the magnetic film surface to increase disk reliability
- A significant reduction in overall surface defects to help reduce read-write errors
- Ability to support lower fly heights (described subsequently)
- Better stiffness to reduce disk dynamics
- Greater ability to withstand shock and damage

Data Organization & Formatting

Tracks - Concentric rings on the platter. Same width as the head.

Gaps - Space between tracks.

Sectors - Sub-division of a track. Data are transferred to & from disk in sectors. Fixed or variable length, fixed length more common, usually 512 bytes.

Constant angular velocity (CAR) - Disk rotated at a fixed speed. Data near centre passes slower than data on the outside. Increased spacing between bits in segments towards outside of disk so that head can read all bits at the same rate.

Advantage - Blocks of data easy to locate.
Disadvantage - Amount of data stored on long outer tracks is same as on shorter inner tracks.

Density - in bits per inch, increases from outer tracks towards inner tracks. Storage capacity in a CAV system is limited by the maximum recording density that can be achieved on the innermost track.

Multiple Zone Recording - To increase density modern systems use multiple zone recording. Surface is divided into a number of concentric zones (typically 16). Within a zone, the number of bits per track is constant. Zones further out contain more bits (more sectors) than zones closer to centre. Allows for greater overall storage capacity, but at expense of more more complex circuitry.

Physical Characteristics

The disk is mounted in a disk drive, which consists of the arm, a spindle that rotates the disk, and electronic circuitry.

Head Motion
- Fixed head (one per track)
- Movable head (one per surface)

Disk Portability
- Nonremovable disk
- Removable disk

Sides
- Single sided
- Double sided

Platters
- Single platter
- Multiple platter

Head Mechanism
- Contact (floppy)
- Fixed gap
- Aerodynamic gap (Winchester)

Fixed-head disk - One read-write head per track. Heads mounted on a rigid arm that extends across all tracks. (Rare today)

Moveable-head disk - Only one read-write track. Head mounted on an arm which can be extended or retracted.

Non-removable disk - is permanently mounted in the disk drive.

Removable disk - can be removed and replaced with another disk.

Double sided - Magnetizable coating applied to both sides of the platter.

Single sided - Magnetizable coating applied to one side of platter. (Less expensive disk systems).

Multiple platters - Some disk drives accomodate multiple platters stacked vertically, and have multiple movable arms, with one read-write head per platter surface.

Cylinder - Set of all tracks in the same relative position on the platters.

Disk Performance Parameters

Seek time - The time taken to position the head at the track, on a movable head system.

Rotational delay (Rotational latency) - Time taken for beginning of sector to rotate and line up with the head.

Access time - The time it takes to get into position to read or write. The sum of the seek time and rotational delay.

Transfer time - Read / write operation is performed as the sector moves under the head. Time required for transfer is the transfer time. Depends on the rotation speed of the disk.

$$T = b / r \times N$$

where:

T = transfer time
b = number of bytes to be transferred
N = Number of bytes on a track
r = rotational speed, in revolutions per second

Total average access time -

T_a = Average seek time + average rotational delay + read all sectors

$T_a = T_s + (1 / 2\ r) + (b / (r\ N))$ where T_s is the average seek time.

Sequential organization - File stored as compactly as possible - occupies all sectors on adjacent tracks.

Example: Disk with - average seek time 20ms, rotational delay 8.3ms, 512byte sectors, 32 sectors per track. Disk rotates at 3600rpm.

Assume file is 256 sectors and stored as compactly as possible (ie: sequential organization).

Calculate the time it takes to read the entire file.

Average seek time = 20ms, Average rotational delay = 8.3ms

Number of tracks - 256 sectors / 32 sectors per track = 8 complete tracks

First track -

Data to be read - b: Bytes to be transferred -
32 sectors x 512 bytes = 16 384 bytes

Bytes in a track - N: 32 sectors x 512 bytes = 16 384 bytes.

Rotational speed - r: 3600rpm / 60sec = 60 revolutions per second.

T_a = 20 ms + 8.3ms + 16 384 bytes / (60 rps x 16 384 bytes)

= 20 ms + 8.3 ms + 0.0167secs (16.7 ms)

= <u>45 ms</u>

Remaining 7 tracks -

Sequential file, no seek time - only rotational delay plus byte transfer time.

Each track is read in: 8.3 ms + 16.7 ms = <u>25 ms</u>

Total time to read entire file -

45 ms (First track) + (7 x 25 ms) (7 other tracks) = 220ms = <u>0.22 secs</u>

6.2 Raid (Read only)

RAID (Redundant Array of Independent Disks) - RAID scheme consists of 7 levels, 0 - 6, which designate different design architectures that share common characteristics, but do not imply a hierarchical relationship -

1. RAID is a set of physical disk drives viewed by the operating system as a single logical drive.
2. Data are distributed across the physical drives of an array in a scheme known as striping.
3. Redundant disk capacity is used to store parity information, which guarantees data recoverability in case of a disk failure.

The RAID strategy employs multiple disk drives and distributes data in such a way as to enable simultaneous access to data from multiple drives, thereby improving I/O performance and allowing easier incremental increases in capacity.

The unique contribution of the RAID proposal is to address effectively the need for redundancy. Although allowing multiple heads and actuators to operate simultaneously achieves higher I/O and transfer rates, the use of multiple devices increases the probability of failure. To compensate for this decreased reliability, RAID makes use of stored parity information that enables the recovery of data lost due to a disk failure.

RAID Level 0 - Not a true RAID system, does not include redundancy to improve performance. User and system data are distributed across all disks, using striping.

Striping - Data is viewed as stored on a logical disk, which is divided into strips (physical blocks, sectors etc). The strips are mapped to physical disks in the RAID array.

RAID Level 1 - Redundancy is achieved by duplicating all data. RAID Levels 2 - 6 use some form of parity calculation to introduce redundancy.

Advantages of RAID Level 1 -

1. A read request can be serviced by either of the two disks that contains the requested data, whichever one involves the minimum seek time plus rotational latency.
2. A write request requires that both corresponding strips be updated, but this can be done in parallel.Thus, the write performance is dictated by the slower of the two writes (i.e., the one that involves the larger seek time plus rotational latency). However, there is no “write penalty” with RAID 1. RAID levels 2 through 6 involve the use of parity bits. Therefore, when a single strip is updated, the array management software must first compute and update the parity bits as well as updating the actual strip in question.
3. Recovery from a failure is simple.When a drive fails, the data may still be accessed from the second drive.

Disadvantage - Cost, requires twice the disk space of the logical drive that it supports.

6.3 Optical Memory

CD
Compact Disk. A nonerasable disk that stores digitized audio information.The standard system uses 12-cm disks and can record more than 60 minutes of uninterrupted playing time.

Disk is formed from resin, such as polycarbonate. Pitted surface is coated with highly reflective surface, usually aluminium or gold. Shiny surface protected by a top coat of clear acrylic. Label can be silkscreened onto the acrylic.

Disk contains a single spiral track, beginning near the centre, and spiraling out to the outer edge. Sectors near the outside are the same length as those near the inside. Information is scanned at the same rate by rotating the disk at a variable speed.

Constant Linear Velocity (CLV) - Due to the variable speed, pits are read by the laser at a constant linear velocity. Disk rotates more slowly for accesses near the outer edge, than for those near the centre.

CD-ROM
Compact Disk Read-Only Memory. A nonerasable disk used for storing computer data.The standard system uses 12-cm disks and can hold more than 650 Mbytes. Similar technology as CD, but CD-ROM has error correction devices to ensure that data are properly transferred.

CD-R
CD Recordable. Similar to a CD-ROM, but includes a dye layer which changes reflectivity, and is activated by high-intensity laser. The user can write to the disk only once.

CD-RW
CD Rewritable. Similar to a CD-ROM.The user can erase and rewrite to the disk multiple times - but will eventually lose its desirable properties. (500,000 to 1,000,000 cycles).

DVD
Digital Versatile Disk. A technology for producing digitized, compressed representation of video information, as well as large volumes of other digital data (Currently 7 times as much as CD-ROM). Both 8 and 12 cm diameters are used, with a double-sided capacity of up to 17 Gbytes.The basic DVD is read-only (DVD-ROM).

Three differences from CD -

1. Bits are packed more closely on a DVD, seven-fold increase in capacity to about 4.7GB.
2. DVD has a second layer of bits on top of first layer. Almost doubles capacity of disk to 8.5GB.
3. DVD-ROM can be double sided, whereas CD is only one-sided. Increases capacity up to 17GB.

DVD-R
DVD Recordable. Similar to a DVD-ROM.The user can write to the disk only once. Only one-sided disks can be used.

DVD-RW
DVD Rewritable. Similar to a DVD-ROM.The user can erase and rewrite to the disk multiple times. Only one-sided disks can be used.

Blu-Ray DVD
High definition video disk. Provides considerably greater data storage density than DVD, using a 405-nm (blue-violet) laser. A single layer on a single side can store 25 Gbytes.

6.4 Magnetic Tape (Read only)

A tape drive is a sequential access device. The tape is only in motion during a read or write operation.

Data on the tape are structured as a number of parallel tracks.

Parallel recoding - Data laid out across all tracks which are recorded at the same time.

Serial recording - Date laid out as a sequence of bits along each track - similar to magnetic disks.

Serpentine recording - First set of bits is recorded along the whole length of the tape. At the end of the tape the heads are repositioned, and the tape is again recorded along its whole length, in the opposite direction.

-ooOoo-

Key Terms

- **access time** - The time it takes the read / write head to get into position to read or write. The sum of the seek time and rotational delay.
- **Blu-ray** - A type of high-definition optical disk. Can store 25GB on a single layer.
- **CD** - A nonerasable disk that stores digitized audio information.
- **CD-ROM** - Compact Disk Read-Only Memory. A nonerasable disk used for storing computer data. The standard system uses 12-cm disks and can hold more than 550 Mbytes.
- **CD-R** - Compact Disk Recordable. Can only be written to once.
- **CD-RW** - Compact Disk Rewritable. Can be repeatedly written and overwritten.
- **constant angular velocity** (CAV) - Disk rotated at a fixed speed.
- **cylinder** - Set of all tracks in the same relative position on the platters.
- **DVD** - Digital Versatile Disk. Can store vast volumes of data, much greater capacity than CD.
- **DVD-ROM** - Can be double sided, dual layer. Basic DVD.
- **DVD-R** - DVD Recordable. Can be written to once, only one-sided disk.
- **DVD-RW** - DVD Rewritable. Can be written / rewritten multiple times, only one-sided disk.
- **fixed-head disk** - One read-write head per track. Heads mounted on a rigid arm that extends across all tracks. (Rare today)
- **floppy disk** - Disk with a small flexible platter. Least expensive type of disk.
- **gap** - Space between adjacent tracks on a disk.
- **head** - Conducting coil used to read and write data to a disk.
- **land** - Area between pits, reflects laser light at high intensity.
- **magnetic disk** - A flat circular plate with a magnetizable surface layer, on one or both sides of which data can be stored.
- **magnetic tape** - A tape with a magnetizable surface layer on which data can be stored by magnetic recording.
- **magnetoresistive** - The MR material has an electrical resistance that depends on the direction of the magnetization of the medium moving under it. By passing a current through the MR sensor, resistance changes are detected as voltage signals. The MR design allows higher-frequency operation, which equates to greater storage densities and operating speeds.
- **movable-head disk** - Only one read-write track. Head mounted on an arm which can be extended or retracted.
- **multiple zoned recording** - the surface is divided into a number of concentric zones (16 is

typical).Within a zone, the number of bits per track is constant. Zones farther from the center contain more bits (more sectors) than zones closer to the center. This allows for greater overall storage capacity at the expense of somewhat more complex circuitry.

- **nonremovable disk** - Disk permanently mounted in the disk drive.
- **optical memory** - a low-powered laser housed in an optical-disk player, is used to read or write to a suitable storage medium.
- **pit** - Area of disk with a rough surface, scatters laser light.
- **platter** - Circular disk constructed of non-magnetic material, called the substrate, which is coated with a magnetizable material.
- **RAID** - Redundant Array of Independent Disks - A disk array in which part of the physical storage capacity is used to store redundant information about user data stored on the remainder of the storage capacity. The redundant information enables regenration of user data in the event that one of the arrays' member disks or the access path to it fails.
- **removable disk** - Disk that can be removed and replaced with another disk.
- **rotational delay** (Rotational latency) - Time taken for beginning of sector to rotate and line up with the head.
- **sector** - Sub-division of a track. Data are transferred to & from a disk in sectors. Fixed or variable length, fixed length more common, usually 512 bytes.
- **seek time** - The time taken to position the head at the track, on a movable head system.
- **serpentine recording** - Magnetic tape, each track recorded separately.
- **striped data** - RAID - Data stored on multiple disks in same area / section of disk.
- **substrate** - Non-magnetic material, used to construct a platter.
- **track** - Concentric rings on the platter. Same width as the head.
- **transfer time** - Read / write operation is performed as the sector moves under the head. Time required for transfer is the transfer time. Depends on the rotation speed of the disk.

--ooOoo--

Chapter 7 - Input / Output

Key Points

- The computer system's I/O architecture is its interface to the outside world. This architecture provides a systematic means of controlling interaction with the outside world and provides the operating system with the information it needs to manage I/O activity effectively.
- The are three principal I/O techniques: **programmed I/O**, in which I/O occurs under the direct and continuous control of the program requesting the I/O operation; **interrupt-driven I/O**, in which a program issues an I/O command and then continues to execute, until it is interrupted by the I/O hardware to signal the end of the I/O operation; and **direct memory access (DMA)**, in which a specialized I/O processor takes over control of an I/O operation to move a large block of data.
- Two important examples of external I/O interfaces are **FireWire** and **Infiniband**.

Summary

7.1 External Devices

Keyboard / Monitor

The most common means of computer / user interaction is a keyboard / monitor arrangement.

Disk Drive

A disk drive contains electronics for exchanging data, control, and status signals with an I/O module, plus the electronics for controlling the disk read / write mechanism.

7.2 I/O Modules

Module Function

- **Control and Timing** - requirement to co-ordinate the flow of traffic between internal resources and external devices;
- **Processor communication** - The I/O module must communicate with the processor and with the external device - command decoding, data, status reporting, and address recognition
- **Device communication** - The I/O module must perform device communication - commands, status information, and data.
- **Data buffering** - Transfer rate of data between main memory / processor and peripheral devices can vary substantially, data must be buffered (stored temporarily).

- **Error detection** - I/O module is responsible for error detection and reporting. Error-detecting code uses a parity bit on each character of data.

I/O Module Structure

I/O modules vary considerably in complexity and the number of external devices that they control.

7.3 Programmed I/O

Three techniques are possible for I/O operations -

- **Programmed I/O** - data are exchanged between the processor and the I/O module. The processor executes a program that gives it direct control of the I/O operation, including sensing device status, sending a read or write command, and transferring the data. When the processor issues a command to the I/O module, it must wait until the I/O operation is complete. If the processor is faster than the I/O module, this is wasteful of processor time.

- **Interrupt-driven I/O** - the processor issues an I/O command, continues to execute other instructions, and is interrupted by the I/O module when the latter has completed its work. With both programmed and interrupt I/O, the processor is responsible for extracting data from main memory for output and storing data in main memory for input.

- **Direct memory access** (DMA) - the I/O module and main memory exchange data directly, without processor involvement.

Overview of programmed I/O

When the processor is executing a program and encounters an instruction relating to I/O, it executes that instruction by issuing a command to the appropriate I/O module. With programmed I/O, the I/O module will perform the requested action and then set the appropriate bits in the I/O status register. The I/O module takes no further action to alert the processor. In particular, it does not interrupt the processor. Thus, it is the responsibility of the processor to check the status of the I/O module periodically until it finds that the operation is complete.

The main disadvantage of this technique is that it is a time-consuming process that keeps the processor busy needlessly.

I/O Commands

- **Control** - Used to activate a peripheral and tell it what to do;

- **Test** - Used to test various status conditions associated with an I/O module and its peripherals;

- **Read** - Causes the I/O module to obtain an item of data from the peripheral and place it in an internal buffer;

- **Write** - Causes the I/O module to take an item of data (byte or word) from the data bus and subsequently transmit that data item to the peripheral.

I/O Instructions

With programmed I/O, there is a close correspondence between the I/O-related instructions that the processor fetches from memory, and the I/O commands that the processor issues to an I/O module to execute the instructions. That is, the instructions are easily mapped into I/O commands, and there is often a simple one-to-one relationship. The form of the instruction depends on the way in which external devices are addressed.

7.4 Interrupt-Driven I/O (Read only - Intel Interrupt Controller to end)

The problem with programmed I/O is that the processor has to wait a long time for the I/O module of concern to be ready for either reception or transmission of data. The processor, while waiting, must repeatedly interrogate the status of the I/O module. As a result, the level of the performance of the entire system is severely degraded.

An alternative is for the processor to issue an I/O command to a module and then go on to do some other useful work. The I/O module will then interrupt the processor to request service when it is ready to exchange data with the processor. The processor then executes the data transfer, as before, and then resumes its former processing.

Interrupt I/O is more efficient than programmed I/O because it eliminates needless waiting. However, interrupt I/O still consumes a lot of processor time, because every word of data that goes from memory to I/O module, or from I/O module to memory, must pass through the processor.

7.5 Direct Memory Access

Drawbacks of Programmed and Interrupt-Driven I/O

Interrupt-driven I/O, though more efficient than simple programmed I/O, still requires the active intervention of the processor to transfer data between memory and an I/O module, and any data transfer must traverse a path through the processor.

Thus, both these forms of I/O suffer from two inherent drawbacks:

- The I/O transfer rate is limited by the speed with which the processor can test and service a device.
- The processor is tied up in managing an I/O transfer, as a number of instructions must be executed for each I/O transfer.

When large volumes of data are to be moved, a more efficient technique is required:
Direct Memory Access (DMA).

DMA Function

DMA involves an additional module on the system bus. The DMA module is capable of mimicking the processor and, indeed, of taking over control of the system from the processor. It needs to do this to transfer data to and from memory over the system bus. For this purpose, the DMA module must use the bus only when the processor does not need it, or it must force the processor to suspend operation temporarily. The latter technique is more common and is referred to as *cycle stealing,* because the DMA module in effect steals a bus cycle.

7.6 I/O Channels & Processors

The Evolution of the I/O Function

- CPU directly controls a peripheral device;
- Controller or I/O module is added;
- Same, but interrupts are used;
- I/O module given direct access to memory via DMA;
- I/O module enhanced to become a processor in its own right;
- I/O module has a local memory of its own, and is in fact a computer in its own right. With this architecture, a large set of I/O devices can be controlled, with minimal CPU involvement. This form of I/O module is referred to as an I/O channel.

Characteristics of I/O Channels

The I/O channel represents an extension of the DMA concept. An I/O channel has the ability to execute I/O instructions, which gives it complete control over I/O operations. In a computer system with such devices, the CPU does not execute I/O instructions. Such instructions are stored in main memory to be executed by a special-purpose processor in the I/O channel itself.

7.7 The External Interface: Firewire and Infiniband

n/a

-ooOoo-

Key Terms

- **cycle stealing** - DMA module forces the processor to suspend operation temporarily.
- **direct memory access** (DMA) - A form of I/O in which a special module, called a *DMA module*, controls the exchange of data between main memory and an I/O module. The CPU sends a request for the transfer of a block of data to the DMA module and is interrupted only after the entire block has been transferred.
- **FireWire** - n/a
- **InfiniBand** - n/a
- **interrupt** - A suspension of a process, such as the execution of a computer program, caused by an event external to that process, and performed in such a way that the process can be resumed. Synonymous with *interruption*.
- **interrupt-driven I/O** - A form of I/O. The CPU issues an I/O command, continues to execute subsequent instructions, and is interrupted by the I/O module when the latter has completed its work.
- **I/O channel** - A relatively complex I/O module that relieves the CPU of the details of I/O operations. An I/O channel will execute a sequence of I/O commands from main memory without the need for CPU involvement.
- **I/O command** - To execute an I/O-related instruction, the processor issues an address, specifying the particular I/O module and external device, and an I/O command. There are four types of I/O commands that an I/O module may receive when it is addressed by a processor: Control, Test, Read, Write.
- **I/O module** - One of the major component types of a computer. It is responsible for the control of one or more external devices (peripherals), and for the exchange of data between those devices and main memory and / or CPU registers.
- **I/O processor** - An I/O module with its own processor, capable of executing its own specialized I/O instructions or, in some cases, general-purpose machine instructions.
- **isolated I/O** - A method of addressing I/O modules and external devices. The I/O address space is treated separately from main memory address space. Specific I/O machine instructions must be used. Compare *memory-mapped I/O*.
- **memory-mapped I/O** - A method of addressing I/O modules and external devices. A single address space is used for both main memory and I/O addresses, and the same machine instructions are used for memory read / write and for I/O.
- **multiplexor channel** - A channel designed to operate with a number of I/O devices simultaneously. Several I/O devices can transfer records at the same time by interleaving items of data. See also *byte multiplexor channel*, *block multiplexor channel*.
- **parallel I/O** - there are multiple lines connecting the I/O module and the peripheral, and multiple bits are transferred simultaneously.
- **peripheral device** - In a computer system, with respect to a particular processing unt, any equipment that provides the processing unit with outside communication.

- **programmed I/O** - A form of I/O in which the CPU issues an I/O command to an I/O module and must then wait for the operation to be completed before proceeding.

- **selector channel** - An I/O channel designed to operate with only one I/O device at a time. Once the I/O device is selected, a complete record is transferred one byte at a time. Contrast with *block multiplexor channel*, *multiplexor channel*.

- **serial I/O** - there is only one line used to transmit data, and bits must be transmitted one at a time.

--ooOoo--

Chapter 9 - Computer Arithmetic

Key Points

- The two principal concerns for computer arithmetic are the way in which numbers are represented (the binary format) and the algorithms used for the basic arithmetic operations (add, subtract, multiply, divide). These two considerations apply both to integer and floating-point arithmetic.

- Floating-point numbers are expressed as a number (significand) multiplied by a constant (base) raised to some integer power (exponent). Floating-point numbers can be used to represent very large and very small numbers.

- Most processors implement the IEEE 754 standard for floating-point representation and floating-point arithmetic. IEEE 754 defines both a 32-bit and a 64-bit format.

Summary

9.1 The Arithmetic and Logic Unit

The ALU is that part of the computer that actually performs arithmetic and logical operations on data. All of the other elements of the computer system - control unit, registers, memory, I/O - are there mainly to bring data into the ALU for it to process, and then to take the results back out.

Data are presented to the ALU in registers, and the results of an operation are stored in registers. These registers are temporary storage locations within the processor that are connected by signal paths to the ALU.

The ALU may also set flags as the result of an operation. For example, an overflow flag is set to 1 if the result of a computation exceeds the length of the register into which it is to be stored. The flag values are also stored in registers within the processor.

9.2 Integer Representation

Sign-Magnitude Representation

The most significant (leftmost) bit is treated as a sign bit (1 = -ve, 0 = +ve). There are several drawbacks to sign-magnitude representation -

- One is that addition and subtraction require a consideration of both the signs of the numbers and their relative magnitudes to carry out the required operation.
- There are also two representations of zero.

Twos Complement Representation

Like sign magnitude, twos complement representation uses the most significant bit as a sign bit, making it easy to test whether an integer is positive or negative. It differs from the use of the sign-magnitude representation in the way that the other bits are interpreted.

To convert to twos complement - Copy existing bits from rhs, up to and including first one, then reverse all remaining bits.

Converting between different bit lengths

Sign-magnitude - Move sign bit to leftmost position and fill with zeroes.

Twos complement - Move sign bit to leftmost position and fill with copies of the sign bit.

9.3 Integer Arithmetic (Negation, addition, subtraction only)

Arithmetic functions on numbers in twos complement representation.

Negation

Sign-magnitude - invert the sign bit.

Twos complement - take the twos complement of the number.

Addition and Subtraction

Addition proceeds as if the two numbers are unsigned integers.

Overflow - occurs if the result is larger than can be held in the word size being used. ALU must signal this, so that no attempt is made to use the result.

Subtraction rule - Take the twos complement of the second number, and add it to the first number.

9.4 Floating-Point Representation

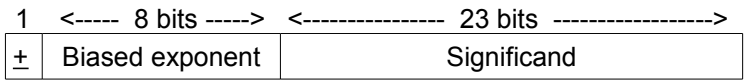
Principles

Binary numbers can be represented in scientific notation -

$$\pm S \times B^{\pm E}$$

Can be stored in a binary word with three fields -

Sign: plus or minus (0 plus, 1 minus);
Significand: S
Exponent: E



32 bit floating point format

Exponent: biased representation - A fixed value, called the bias, is subtracted from the field to get the true exponent value. The bias equals $(2^{k-1} - 1)$, where k is the number of bits in the binary exponent. An 8-bit field yields the numbers 0 to 255. With a bias of 127 $(2^7 - 1)$, the true exponent values are in the range 127 to 128. The value 127 is added to the true exponent to be stored in the exponent field.

Normalized number - is one in which the most signficant digit of the significand in non-zero. Because the most significant bit is always one, it is unnecessary to store this bit, it is implicit.

IEEE Standard for Binary Floating-point Representation

As above, standard developed to facilitate the portability of programs from one processor to another. The standard has been widely adopted and is used on virtually all contemporary processors and arithmetic coprocessors.

9.5 Floating-Point Arithmetic (Omit)

-ooOoo-

Key Terms

- **arithmetic & logic unit** (ALU) - A part of a computer that performs arithmetic operations, logic operations, and related operations.
- **arithmetic shift** -
- **base** - In the numeration system commonly used in scientific papers, the number that is raised to the power denoted by the exponent and them multiplied by the mantissa to determine the real number represented (eg: the number 10 in the expression 2.7' 10^2 = 270).
- **biased representation** -
- **denormalized number** -
- **divided** -
- **divisor** -
- **exponent** -
- **exponent overflow** -
- **exponent underflow** -
- **fixed-point representatio**n - A radix numeration system in which the radix point is implicitly fixed in the series of digit places by some convention upon which agreement has been reached.
- **floating-point representation** - A numeration system in which a real number is represented by a pair of distinct numerals, the real number being the product of the fixed-point part, one of the numerals, and a value obtained by raising the implicit floating-point base to a power denoted by the exponent in the floating-point representation, indicated by the second numeral.
- **guard bits** -
- **mantissa** -
- **minuend** -
- **multiplicand** -
- **multiplier** -
- **negative overflow** -
- **negative underflow** -
- **normalized number** -
- **ones complement representation** - Used to represent binary integers. A positive integer is represented as in sign magnitude. A negative integer is represented by reversing each bit in the representation of a positive integer of the same magnitude.
- **overflow** -
- **partial product** -

- **positive overflow** -
- **positive underflow** -
- **product** -
- **quotient** -
- **radix point** -
- **remainder** -
- **rounding** -
- **sign bit** -
- **significand** -
- **significand overflow** -
- **significand underflow** -
- **sign-magnitude representation** - Used to represent binary integers. In an N-bit word, the leftmost bit is the sign (0 = positive, 1 = negative) and the remaining N - 1 bits comprise the magnitude of the number.
- **subtrahend** -
- **twos complement representation** - Used to represent binary integers. A positive integer is represented as in sign magnitude. A negative number is represented by taking the Boolean complement of each bit of the corresponding positive number, then adding 1 to the resulting bit pattern viewed as an unsigned integer.

--ooOoo--

Chapter 10 - Instruction Sets

Key Points

- The essential elements of a computer instruction are the opcode, which specifies the operation to be performed; the source and destination operand references, which specify the input and output locations for the operation; and a next instruction reference, which is usually implicit.
- Opcodes specify operations in one of the following general categories: arithmetic and logic operations; movement of data between two registers, register and memory, or two memory locations; I/O; and control.
- Operand references specify a register or memory location of operand data. The type of data may be addresses, numbers, characters, or logical data.
- A common architectural feature in processors is the use of a stack, which may or may not be visible to the programmer. Stacks are used to manage procedure calls and returns and may be provided as an alternative form of addressing memory. The basic stack operations are PUSH, POP, and operations on the top one or two stack locations. Stacks typically are implemented to grow from higher addresses to lower addresses.
- Byte-addressable processors may be categorized as big endian, little endian, or bi-endian. A multibyte numerical value stored with the most significant byte in the lowest numerical address is stored in big-endian fashion. The little-endian style stores the most significant byte in the highest numerical address. A bi-endian processor can handle both styles.

Summary

10.1 Machine Instruction Characteristics

Machine instructions / Computer instructions - Instructions executed by the processor.

Instruction set - The collection of different instructions that the processor can execute.

Elements of a Machine Instruction

- Operation code (opcode) - Specifies the operation to be performed. Eg: ADD
- Source operand reference - The input operand for the operation.
- Result operand reference - The operation may produce a result.
- Next instruction reference - Tells the CPU where to fetch the next instruction.

Source and result operands can be in -

- Main / virtual memory
- CPU register
- Immediate - The value of the operand is contained in a field in the instruction being executed.
- I/O device

Instruction Representation

Each instruction is represented by a sequence of bits -

4 bits	6 bits	6 bits
Opcode	Operand reference	Operand reference

Opcodes are represented by abbreviations, known as *mnemonics.*

Instruction Types

- **Data processing** - Arithmetic and logic instructions.
- **Data storage** - Memory instructions.
- **Data movement** - I/O instructions.
- **Control** - Test and branch instructions.

Number of Addresses

Most instructions have one or two addresses.

Instruction Set Design

Fundamental design issues -

- **Operation repertoire** - How many, and which operations to provide.
- **Data types** - Different types of data on which operations are performed.
- **Instruction format** - Length, number of addresses, size of fields, etc.
- **Registers** - Number of CPU registers that can be referenced.
- **Addressing** - The mode/s by which the address of an operand is specified.

10.2 Types of Operands

Machine instructions operate on data. The most important general categories of data are:

- **Addresses** - can be considered as unsigned integers, and can be used in calculations.
- **Numbers** - Three types of numerical data: Integer / fixed point, Floating point, Decimal.
- **Characters** - Most common character code is ASCII, which represents 128 different characters, 7 bits each.

- **Logical data** - Sometimes useful to consider an n-bit unit as consisting of n 1-bit items, each having the value 0 or 1. Data viewed this way are considered to be *logical data*.

Appendix B.1 - Assembly language (Read sections B.2 to B.5)

10.3 Intel x86 and ARM Data Types

x86 Data Types

The Pentium can deal with data types of 8 (byte), 16 (word), 32 (doubleword), and 64 (quadword) bits in length. It uses little-endian style: the least significant type is stored in the lowest address.

10.4 Types of Operations

- **Data transfer** - The following must be specified: *location* (memory, register, top of stack) of the source and destination operands; *length* of data to be transferred; and the *mode of addressing* for each operand.
- **Arithmetic** - Basic arithmetic operations: add, subtract, multiply, divide, as well as absolute, negate, increment, and decrement.
- **Logical** - Most common: NOT, AND, OR, and XOR.
  - Bit shifting -
    - Logical shift - bits are shifted left or right, end bits are lost.
    - Arithmetic shift - data treated as a signed integer, so sign bit is not shifted. Right shift - sign bit is replicated. Left shift - Shift performed on all bits except the sign bit.
    - Rotate - All bits are preserved because the shift is cyclic.
- **Conversion** - Change the format, or operate on the format, of the data. Eg: converting from decimal to binary.
- **I/O** - Variety of approaches: programmed I/O, DMA, and use of an I/O processor.
- **System Control** - Instructions reserved for the use of the OS.
- **Transfer of Control** - Instructions that change the sequence of instruction execution.
  - Repetition / loops
  - Decision making / branching
  - Procedures / sub-routines - return address stored: register, start of called procedure, stack.

10.5 Intel x86 and ARM OperationTypes

(Study up to, but not including, section on x86 SIMD. Read rest)

Call / return instructions - The Pentium provides four instructions to support procedure call / return: CALL, ENTER, LEAVE, RETURN.

Memory Management - A set of specialized instructions deals with memory segmentation. These are privileged instructions that can only be executed from the OS.

Condition codes - are bits in special registers that may be set by certain operations and uased in conditional branch instructions. Thses conditions are set by arithmetic and compare operations.

Stacks

Stack pointer - Contains the address of the top of the stack.

Stack base - Contains the address of th bottom of the stack.

Stack limit - Contains the address of the end of the reserved block.

The stack grows from higher addresses to lower addresses. To speed up stack operations, the top two stack elements are often stored in registers, ad the stack pointer contains the address of the third element.

Expression evaluation

Mathematical formulae are usually expressed in **infix** notation. A binary operator appears bewtween the operands.

Reverse Polish Notation (Postfix) the operator follows its two operands. Advantage: easily evaluated using a stack.

Little-endian, big-endian, bi-endian

Big endian - Bytes are stored from left to right.
Little endian - Bytes are stored from right to left.

Example value: 12345678

Address	100	101	102	103
Big-endian value	12	34	56	78
Little-endian value	78	56	34	12

Advantages of Big-endian style -

- A big-endian processor is faster in comparing character strings.
- Values can be printed left to right without causing confusion.
- Big-endian processors store their integers and character strings in the same order (most significant byte comes first).

Advantages of Little-endian style -

- It is easier to perform higher-precision arithmetic with the Little-endian style because you don't have to find the least-significant byte and move backwards.
- A big-endian processor has to perform addition when converting a 32-bit integer address to a 16-bit integer address, to use the least significant bytes.

Key Terms

- **accumulator** - The name of the CPU register in a single-address instruction format. The accumulator, or AC, is implicitly one of the two operands for the instruction.
- **address** -
- **arithmetic shift** - operation treats the data as a signed integer and does not shift the sign bit. On a right arithmetic shift, the sign bit is replicated into the bit position to its right. On a left arithmetic shift, a logical left shift is performed on all bits but the sign bit, which is retained. Left shift - multiply by 2, Right shift - divide by 2.
- **bi-endian** - n/a
- **big endian** - stores the most significant byte in the lowest numerical byte adress
- **branch** -
- **conditional branch** - branch is made only if a certain condition is met.
- **instruction set** - The collection of different instructions that the processor can execute.
- **jump** -
- **little endian** - stores the least significant byte in the lowest numerical byte adress.
- **logical shift** - the bits of a word are shifted left or right. On one end, the bit shifted out is lost. On the other end, a 0 is shifted in. Logical shifts are useful primarily for isolating fields within a word.
- **machine instruction** - The operation of the processor is determined by the instructions it executes, referred to as *machine instructions* or *computer instructions*.
- **operand** - An entity on which an operation is performed.
- **operation** -
- **packed decimal** -
- **pop** - operation removes the top item from the stack.
- **procedure call** -
- **procedure return** -
- **push** - operation appends one new item to the top of the stack.
- **reentrant procedure** -
- **reverse Polish notation** - (Postfix) In this notation, the operator follows its two operands.
- **rotate** - or cyclic shift, operations preserve all of the bits being operated on. One use of a rotate is to

bring each bit successively into the leftmost bit, where it can be identified by testing the sign of the data (treated as a number).

- **skip** -

- **stack** - An ordered list in which items are appended to and deleted from the same end of the list, known as the top. That is, the next item appended to the list is put on the top, and the next item to be removed from the list is the item that has been in the list the shortest time. This method is characterized as last-in-first-out.

--ooOoo--

Chapter 11 - Instruction Sets: Addressing Modes and Formats

Key Points

- An operand reference in an instruction either contains the actual value of the operand (immediate) or a reference to the address of the operand. A wide variety of addressing modes is used in various instruction sets. These include direct (operand address is in address field), indirect (address field points to a location that contains the operand address), register, register indirect, and various forms of displacement, in which a register value is added to an address value to produce the operand address.
- The instruction format defines the layout fields in the instruction. Instruction format design is a complex undertaking, including such consideration as instruction length, fixed or variable length, number of bits assigned to opcode and each operand reference, and how addressing mode is determined.

Summary

11.1 Addressing

Immediate Addressing

The operand (a value) is present in the instruction. Eg: mov ax, 3

Direct Addressing

The address field contains the the effective address of the operand. Eg: mov ax, [102h] (Stores the contents of memory address 102h in ax).

Indirect Addressing

One of the operands is the address of the actual operand. Eg: mov bx, prompt (Stores the address of the label *prompt* in bx).

Register Addressing

Similar to direct addressing, but the address field refers to a register rather than a memory address.
Eg: mov ax, bx

Register Indirect Addressing

Similar to indirect addressing, but the address field refers to a register rather than a memory address.
Eg: mov dl, [bx]

Displacement Addressing

Combination of direct addressing and register indirect addressing.
Eg: mov cx, [name + si]

Stack Addressing

The SP register contains the offset from the beginning of the stack to the top of the stack. We use PUSH and POP to push values onto the stack, and to pop them off the stack respectively.
Eg: pop ax

11.2 x86 and ARM Addressing Modes (Omit ARM addressing modes)

Pentium has a variety of addressing modes.

Segment register - determines the segment that is the subject of the reference.

- DS - Data segment
- ES - Extra
- SS - Stack
- CS - Code

Base register - bx

Index register - SI (Source index), DI (Destination index)

11.3 Instruction Formats (Study first 3 pages, read rest)

A n instruction format defines the layout of the bits of an instructon. The design of an instruction format is complex, and a variety of designs have been implemented.

11.4 x86 and ARM Instruction Formats (Read only)

11.5 Assembly Language

See Appendix B

Key Terms

- **autoindexing** - A form of indexed addressing in which the index register is automatically incremented or decremented with each memory reference.
- **base-register addressing** - The referenced register contains a main memory address, and the address field contains a displacement (usually an unsigned integer representation) from that address.
- **direct addressing** - the address field contains the effective address of the operand.
- **displacement addressing** - A very powerful mode of addressing combines the capabilities of direct addressing and register indirect addressing.
- **effective address** - either a main memory address or a register.
- **immediate addressing** - the operand value is present in the instruction.
- **indexing** - A technique of address modification by means of index registers.
- **indirect addressing** - With direct addressing, the length of the address field is usually less than the word length, thus limiting the address range. One solution is to have the address field refer to the address of a word in memory, which in turn contains a full-length address of the operand.
- **instruction format** - The layout of a computer instruction as a sequence of bits. The format divides the instruction into fields, corresponding to the constituent elements of the instruction (eg: opcode, operands).
- **postindexing** -
- **preindexing** -
- **register addressing** - the address field refers to a register.
- **register indirect addressing** - Just as register addressing is analogous to direct addressing, register indirect addressing is analogous to indirect addressing. In both cases, the only difference is whether the address field refers to a memory location or a register.
- **relative addressing** - the implicitly referenced register is the program counter (PC).
- **word** - An ordered set of bytes or bits that is the normal unit in which information may be stored, transmitted, or operated on within a given computer. Typically, if a processor has a fixed-length instruction set, then the instruction length equals the word length.

--ooOoo--

Chapter 12 - Processor Structure and Function

Key Points

- A processor includes both user-visible registers and control/status registers. The former may be referenced, implicitly or explicitly, in machine instructions. User-visible registers may be general purpose or have a special use, such as fixed-point or floating-point numbers, addresses, indexes, and segment pointers. Control and status registers are used to control the operation of the processor. One obvious example is the program counter.

  Another important example is a program status word (PSW) that contains a variety of status and condition bits. These include bits to reflect the result of the most recent arithmetic operation, interrupt enable bits, and an indicator of whether the processor is executing in supervisor or user mode.

- Processors make use of instruction pipelining to speed up execution. In essence, pipelining involves breaking up the instruction cycle into a number of separate stages that occur in sequence, such as fetch instruction, decode instruction, determine operand addresses, fetch operands, execute instruction, and write operand result. Instructions move through these stages, as on an assembly line, so that in principle, each stage can be working on a different instruction at the same time. The occurrence of branches and dependencies between instructions complicates the design and use of pipelines.

Summary

12.1 Processor Organization

The processor must -

- Fetch instruction from memory
- Interpret instruction
- Fetch data, if required
- Process data
- Write data to memory or I/O module.

12.2 Register Organization

User-Visible Registers

- **General purpose registers** -
  - AX Primary accumulator - Mainly used for data movement IMUL, IDIV etc;
  - BX Base register - Only register that can be used as a pointer;
  - CX Count register - Used to control loops or shift operations;
  - DX Data register.

- **Data registers** - Can only be used to hold data.
- **Address registers** -
  - Segment pointers - segment register holds the base address of the segment. Segment address always ends with 0, therefore the last digit is not stored.
    - CS - Code segment. Contains machine instructions of the program;
    - DS - Data segment. Contains variables, constants and work areas;
    - SS - Stack segment. Contains the program stack.
  - Index registers - used for indexed addressing. Contan the offset relative to the start of the segment.
    - SI - Source index. Usually contains an offset from the DS register, but it can address any variable.
    - DI - Destination index. Usually contains an offset from the ES register, but it can address any variable.
  - Stack pointer - SP holds the address of the last element pushed onto the stack.

Control and Status Registers

Condition codes are bits set by the CPU as a result of operations. The code can be tested as part of a conditional branch operation, but cannot be altered by a programmer.

Flag	Debug representation
CF Carry flag	CY = CarrY NC = No Carry
PF Parity flag	PE = Parity Even PO = Parity Odd
AF Auxiliary flag	AC = Auxilliary Carry NA = No Auxilliary
ZF Zero flag	NZ = Not Zero ZR = ZeRo
SF Sign flag	PL = PLus NG = NeGative
TF Trap flag	Not shown
IF Interrupt flag	EI = Enable Interrupts DI = Disable Interrupts
DF Direction flag	UP = UP (right) DN = DowN (left
OF Overflow flag	NV = No Overflow OV = OVerflow

12.3 Instruction Cycle

The main subcycles of the instruction cycle are: Fetch, Execute, Interrupt.

Indirect Cycle - is a subcycle for when indirect addressing is used.

12.4 Instruction Pipelining (Up to but not including "Pipeline Performance" - Read rest)

The major need for pipelining is to utilize the CPu to its maximum capacity during fetch and executecycles. Involves partially executing sub-cycles in parallel.

12.5 The x86 Processor Family

Reverse byte order in memory - The CPU expects numeric data in MEMORY (but not registers) to be stored in reverse byte order. The CPU reverses the bytes again when loading the data from memory into registers.

12.6 The ARM Processor (Read only)

Key Terms

- **branch prediction** - A mechanism used by the processor to predict the outcome of a program branch prior to its execution.
- **condition code** - A code that reflects the result of a previous operation (eg: arithmetic). A CPU may include one or more condition codes, which may be stored separately within the CPU or as part of a larger control register. Also known as a *flag*.
- **delayed branch** - n/a
- **flag** - See *condition code*.
- **instruction cycle** - The processing performed by a CPU to execute a single instruction.
- **instruction pipeline** - new inputs are accepted at one end before previously accepted inputs appear as outputs at the other end.
- **instruction prefetch** - n/a
- **program status word** (PSW) -

--ooOoo--

Chapter 13 - Reduced Instruction Set Computers (RISC)

Key Points

- Studies of the execution behavior of high-level language programs have provided guidance in designing a new type of processor architecture: the reduced instruction set computer (RISC). Assignment statements predominate, suggesting that the simple movement of data should be optimized. There are also many IF and LOOP instructions, which suggest that the underlying sequence control mechanism needs to be optimized to permit efficient pipelining. Studies of operand reference patterns suggest that it should be possible to enhance performance by keeping a moderate number of operands in registers.

- These studies have motivated the key characteristics of RISC machines: (1) a limited instruction set with a fixed format, (2) a large number of registers or the use of a compiler that optimizes register usage, and (3) an emphasis on optimizing the instruction pipeline.

- The simple instruction set of a RISC lends itself to efficient pipelining because there are fewer and more predictable operations performed per instruction. A RISC instruction set architecture also lends itself to the delayed branch technique, in which branch instructions are rearranged with other instructions to improve pipeline efficiency.

Summary

The key elements of most RISC designs are -

- A large number of general purpose registers;
- A limited and simple instruction set;
- An emphasis on optimising the instruction pipeline.

13.1 Instruction Execution Characteristics

Characteristics of RISC architectures -

- use a large number of registers, or use a compiler to optimize register usage;
- careful attention to design of instruction pipelines;
- a simplified (reduced) instruction set.

13.2 The Use of a Larger Register File

A strategy is needed that will allow the most frequently accessed operands to be kept in registers, and to minimize register-memory operations.

Two possible approaches -

- Software approach - rely on the compiler to maximize register usage.
- Simply use more registers so that more variables can be held in registers for longer periods of time.

The register file, organised into windows, acts as a small, fast buffer for holding a subset of all variables that are likely to be used the most heavily, acting like a faster cache memory.

Larger register file versus cache

Large register file	Cache
Holds *all* the local scalar variables of the most recent N-1 procedure activations.	Holds a *selection* of recently used scalar variables.
Efficient use of *time*, because all local scalar variables are retained.	Efficient use of *space*, because it is reacting to the situation dynamically.
Disadvantage: Inefficient use of space, because not all procedures will need the full window space allotted to them.	Disadvantage: Data are read into the cache in *blocks*, so some of the data will not be used.
Can hold some global scalars, but it is difficult for a compiler to determine which ones will be heavily used.	Can handle global as well as local variables.
Infrequent use of memory.	Set associative memories with a small size, so data might overwrite frequently used variables.
To reference a local scalar, a 'virtual' register number and a window number are used. These can pass through a simple decoder to select one of the physical registers.	A full-width memory address must be generated. The access time is much longer.

From the point of view of performance, the window-based register file is superior to cache for local scalars.

13.3 Compiler-Based Register Optimization

You could have a small number of registers and let the compiler optimise their usage. Graph colouring is the most commonly used technique.

In general, there is a trade-off between the use of a large set of registers and compiler-based optimisation. (The larger the number of registers, the smaller the benefit of register optimisation).

13.4 Reduced Instructon Set Architecture

Characteristics of reduced instruction set architectures

- One machine instruction per machine cycle;
- Register-to-register operations;
- Simple addressing modes;
- Simple instruction formats.

Benefits of the RISC approach

Performance benefits -

- More effective *optimising compilers* can be developed (With more primitive instructions there are more opportunities for code efficiency);

- Most instructions generated by a compiler are relatively *simple* anyway (so a control unit built specifically for those instructions could execute them faster than a comparable CISC);
- *Instruction pipelining* can be applied much more effectively with a reduced instruction set;
- RISC processors are more responsive to *interrupts* because interrupts are checked between rather elementary operations.

VLSI implementation benefits -

With the advent of VLSI, it is possible to put an entire processor on a single chip. For a single-chip processor, there are two motivations for following a RISC strategy:

- *Performance* issue (On-chip delays are of much shorter duration than inter-chip delays);
- *Design-and-implementation* time (A RISC processor is far easier to develop than a VLSI processor)

13.5 RISC Pipelining (Read only)

13.6 MIPS R4000 (Read only)

13.7 SPARC (Read only)

13.8 RISC v CISC Controversy

RISC designs may benefit from the inclusion of some CISC features and vice versa.
The PowerPC uses a RISC design and the Pentium II uses a CISC design, but neither is pure.

Key Terms

- **complex instruction set computer** (CISC) -
- **delayed branch** - n/a
- **delayed load** - n/a
- **high-level language** (HLL) -
- **reduced instruction set computer** (RISC) - Dramatic departure from the historical trend in processor architecture -
  - A large number of general-purpose registers, and / or the use of compiler technology to optimize register usage;
  - A limited and simple instruction set;
  - An emphasis on optimizing the instruction pipeline.
- **register file** - organized into windows, acts as a small, fast buffer for holding a subset of all variables that are likely to be used the most heavily.
- **register window** - multiple small sets of registers.
- **SPARC** (Scalable Processor Architecture) - An architecture defined by Sun Microsystems, its instruction set and register organization is based closely on the Berkeley RISC model.

--ooOoo--

Appendix B - Assembly Language and Related Topics

Key Points (Appendix B - Read only B.2 to B.5)

- An **assembly language** is a symbolic representation of the machine language of a specific processor, augmented by additional types of statements that facilitate program writing, and that provide instructions to the assembler.
- An **assembler** is a program that translates assembly language into machine code.
- The first step in the creation of an active process is to load a program into main memory and create a process image.
- A **linker** is used to resolve any references between loaded modules.

Summary

Reasons why it is worthwhile to study assembly language programming -

- It clarifies the execution of instructions.
- It shows how data is represented in memory.
- It shows how a program interacts with the operating system, processor, and the I/O system.
- It clarifies how a program accesses external devices.
- Understanding assembly language programmers makes students better high-level language (HLL) programmers, by giving them a better idea of the target language that the HLL must be translated into.

Disadvantages of using Assembly language, rather than a Higher Level Language -

- **Development time.** Writing code in assembly language takes much longer than writing in a high-level language.
- **Reliability and security.** It is easy to make errors in assembly code.The assembler is not checking if the calling conventions and register save conventions are obeyed. Nobody is checking for you if the number of PUSH and POP instructions is the same in all possible branches and paths.There are so many possibilities for hidden errors in assembly code that it affects the reliability and security of the project unless you have a very systematic approach to testing and verifying.
- **Debugging and verifying.** Assembly code is more difficult to debug and verify because there are more possibilities for errors than in high-level code.
- **Maintainability.** Assembly code is more difficult to modify and maintain because the language allows unstructured spaghetti code and all kinds of tricks that are difficult for others to understand. Thorough documentation and a consistent programming style are needed.
- **Portability.** Assembly code is platform-specific. Porting to a different platform is difficult.

- **System code can use intrinsic functions instead of assembly.** The best modern C++ compilers have intrinsic functions for accessing system control registers and other system instructions. Assembly code is no longer needed for device drivers and other system code when intrinsic functions are available.

- **Application code can use intrinsic functions or vector classes instead of assembly.** The best modern C++ compilers have intrinsic functions for vector operations and other special instructions that previously required assembly programming.

- **Compilers have been improved a lot in recent years.** The best compilers are now quite good. It takes a lot of expertise and experience to optimize better than the best C compiler.

Advantages of using Assembly language -

- **Debugging and verifying.** Looking at compiler-generated assembly code or the disassembly window in a debugger is useful for finding errors and for checking how well a compiler optimizes a particular piece of code.

- **Making compilers.** Understanding assembly coding techniques is necessary for making compilers, debuggers and other development tools.

- **Embedded systems.** Small embedded systems have fewer resources than PCs and mainframes.Assembly programming can be necessary for optimizing code for speed or size in small embedded systems.

- **Hardware drivers and system code.** Accessing hardware, system control registers etc. may sometimes be difficult or impossible with high level code.

- **Accessing instructions that are not accessible from high-level language.** Certain assembly instructions have no high-level language equivalent.

- **Self-modifying code.** Self-modifying code is generally not profitable because it interferes with efficient code caching. It may, however, be advantageous, for example, to include a small compiler in math programs where a user-defined function has to be calculated many times.

- **Optimizing code for size.** Storage space and memory is so cheap nowadays that it is not worth the effort to use assembly language for reducing code size. However, cache size is still such a critical resource that it may be useful in some cases to optimize a critical piece of code for size in order to make it fit into the code cache.

- **Optimizing code for speed.** Modern C++ compilers generally optimize code quite well in most cases. But there are still cases where compilers perform poorly and where dramatic increases in speed can be achieved by careful assembly programming.

- **Function libraries.** The total benefit of optimizing code is higher in function libraries that are used by many programmers.

- **Making function libraries compatible with multiple compilers and operating systems.** It is possible to make library functions with multiple entries that are compatible with different compilers and different operating systems. This requires assembly programming.

Assembly Language Elements

A statement in Assembly language consists of four elements - *label*, *mnemonic*, *operand*, and *comment*.

Type of Assembly Language Statements

Assembly language statements are one of four types - *instruction*, *directive*, *macro definition*, and *comment*.

Key Terms (A)

- **Assembler** - A program that translates assembly language into machine code.
- **Assembly Language** - A symbolic representation of the machine language of a specific processor, augmented by additional types of statements that facilitate program writing and that provide instructions to the assembler.
- **Compiler** - A program that converts another program from some source language (or programming language) to machine language (object code). Some compilers output assembly language which is then converted to machine language by a separate assembler. A compiler is distinguished from an assembler by the fact that each input statement does not, in general, correspond to a single machine instruction or fixed sequence of instructions. A compiler may support such features as automatic allocation of variables, arbitrary arithmetic expressions, control structures such as FOR and WHILE loops, variable scope, input/ouput operations, higher-order functions and portability of source code.
- **Executable Code** - The machine code generated by a source code language processor such as an assembler or compiler. This is software in a form that can be run in the computer.
- **Instruction Set** - The collection of all possible instructions for a particular computer; that is, the collection of machine language instructions that a particular processor understands.
- **Linker** - A utility program that combines one or more files containing object code from separately compiled program modules into a single file containing loadable or executable code.
- **Loader** - A program routine that copies an executable program into memory for execution.
- **Machine Language, or Machine Code** - The binary representation of a computer program which is actually read and interpreted by the computer. A program in machine code consists of a sequence of machine instructions (possibly interspersed with data). Instructions are binary strings which may be either all the same size (e.g., one 32-bit word for many modern RISC microprocessors) or of different sizes.
- **Object Code** - The machine language representation of programming source code. Object code is created by a compiler or assembler and is then turned into executable code by the linker.

Key Terms (B)

- **comment** - All assembly languages allow comments in the program. Begins with a special character (;) that signals that the rest of the line is a comment and to be ignored.
- **directive** (pseudo-instruction) - Assembly language statement that is not directly translated into machine language. An instruction to the assembler to perform a specified action during the assembly process.
- **dynamic linker** -
- **instruction** - Symbolic representation of machine language instruction. Invariably, there is a one-to-one relationship between an assembly language instruction and a machine instructiion.
- **label** - Most frequently used in branch instructions. The assembler defines the label as equivalent to the address into which the first byte of the object code generated for that instruction will be loaded.
- **linkage editor** -
- **linking** -
- **load-time dynamic linking** -
- **loading** -
- **macro** - A section of code that is written once, and can be used many times. Similar to a subroutine.
- **mnemonic** - The name of the operation or function of the assembly language statement. Can correspond to a machine instruction, an assembler directive, or a macro.
- **one-pass assembler** -
- **operand** - An entity on which an operation is performed. Identifies an immediate value, a register value, or a memory location.
- **relocation** -
- **run-time dynamic linking** -
- **two-pass assembler** -

--ooOoo--