DIGITAL LOGIC AND COMPUTER ORGANIZATION

II B.TECH I SEMESTER - CSE (AR 23)

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

LENDI INSTITUTE OF ENGINEERING AND TECHNOLOGY

(An Autonomous Institute, Approved by AICTE & Permanently Affiliated to JNTU-GV, Vizianagaram)

(Accredited By NAAC with A Grade and Accredited by NBA)

Jonnada (Village), Denkada (Mandal), Vizianagaram District – 535 005

Phone No. 08922-241111, 241112

E-Mail: lendi_2008@yahoo.com website: www.lendi.org

UNIT-V

MEMORY AND PROCESSING UNIT: Read Only Memory: ROM, PROM, EPROM, EEPROM, Flash Memory, Cache Memories: Mapping Functions, Secondary Storage: Magnetic Hard Disks, Optical Disks.

Fundamental Concepts: Register Transfers, Performing an Arithmetic or Logic Operation, Fetching a word from Memory, Execution of Complete Instruction, Hardwired Control, Micro-programmed Control.

INDEX

UNIT-V

MEMORY AND PROCESSING UNIT: Read Only Memory: ROM, PROM, EPROM, EEPROM, Flash Memory, Cache Memories: Mapping Functions, Secondary Storage: Magnetic Hard Disks, Optical Disks.

Fundamental Concepts: Register Transfers, Performing an Arithmetic or Logic Operation, Fetching a word from Memory, Execution of Complete Instruction, Hardwired Control, Micro-programmed Control.

MEMORY AND PROCESSING UNIT

INTRODUCTION

Programs and the data they operate on are held in the memory of the computer. The execution speed of programs is highly dependent on the speed with which instructions and data can be transferred between the processor and the memory. It is also important to have sufficient memory to facilitate execution of large programs having large amounts of data. **Ideally**, the **memory would be fast, large, and inexpensive**. Unfortunately, it is impossible to meet all three of these requirements simultaneously. Increased speed and size are achieved at increased cost. Much work has gone into developing structures that improve the effective speed and size of the memory, yet keep the cost reasonable. The memory of a computer comprises a hierarchy, including a cache, the main memory, and secondary storage.

Basic Concepts

The maximum size of the memory that can be used in any computer is determined by the addressing scheme. For example, a computer that generates 16-bit addresses is capable of addressing up to 2^{16} = 64K (kilo) memory locations. Machines whose instructions generate 32-bit addresses can utilize a memory that contains up to 2^{32} = 4G (giga) locations, whereas machines with 64-bit addresses can access up to 2^{64} = 16E (exa) ≈ 16 × 10^{18} locations. **The number of locations represents the size of the address space of the computer.**

The connection between the processor and its memory consists of **address, data, and control lines**, as shown in below figure. The processor uses the **address lines to specify the memory location involved in a data transfer operation**, and uses the **data lines to transfer the data**. At the same time, the **control lines carry the command indicating a Read or a Write operation** and whether a byte or a word is to be transferred. The **control lines also provide the necessary timing information and are used by the memory to indicate when it has completed the requested operation**.

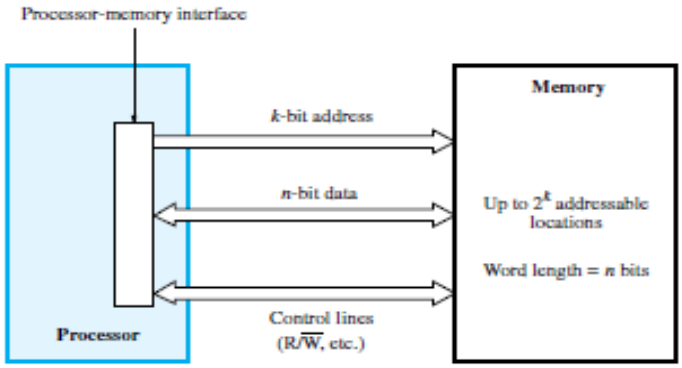

Figure: Connection of the memory to the processor

Measure of the speed of memory units:

MEMORY ACCESS TIME: The time that elapses between the initiation of an operation to transfer a word of data and the completion of that operation.

MEMORY CYCLE TIME: The minimum time delay required between the initiations of two successive memory operations. Example: The time between two successive Read operations.

The memory cycle time is usually slightly longer than the access time, depending on the implementation details of the memory unit. A memory unit is called a random-access memory (RAM) if the access time to any location is the same, independent of the location's address. This distinguishes such memory units from serial, or partly serial, access storage devices such as magnetic and optical disks.

READ-ONLY MEMORIES

Both static and dynamic RAM chips are volatile, which means that they retain information only while power is turned on. There are many applications requiring memory devices that retain the stored information when power is turned off. For example, bootstrap process of loading the operating system from a hard disk into the main memory. Many embedded applications do not use a hard disk and require non-volatile memories to store their software. Different types of non-volatile memories have been developed. Generally, their contents can be read in the same way as for their volatile counterparts. But, a special writing process is needed to place the information into a non-volatile memory. Since its normal operation involves only reading the stored data, a memory of this type is called a read-only memory (ROM).

ROM

A memory is called a read-only memory, or ROM, when information can be **written** into it **only once at the time of manufacture**. The below figure shows a possible configuration for a ROM cell. A logic value 0 is stored in the cell if the transistor is connected to ground at point P; otherwise, a 1 is stored. The bit line is connected through a resistor to the power supply. To read the state of the cell, the word line is activated to close the transistor switch. As a result, the voltage on the bit line drops to near zero if there is a connection between the transistor and ground. If there is no connection to ground, the bit line remains at the high voltage level, indicating a 1. A sense circuit at the end of the bit line generates the proper output value. The state of the connection to ground in each cell is determined when the chip is manufactured, using a mask with a pattern that represents the information to be stored.

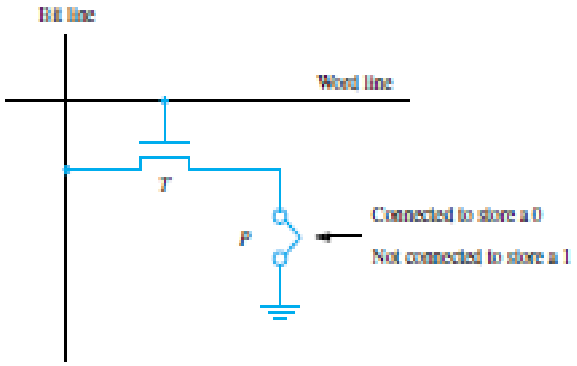

Figure: A ROM cell.

PROM

Some ROM designs **allow the data to be loaded by the user**, thus providing a programmable ROM (PROM). Programmability is achieved by inserting a fuse at point Pin in the above figure. Before it is programmed, the memory contains all 0s. The user can insert 1s at the required locations by burning out the fuses at these locations using high-current pulses. Of course, this process is irreversible.

PROMs provide flexibility and convenience not available with ROMs. The cost of preparing the masks needed for storing a particular information pattern makes ROMs cost effective only in large volumes. The alternative technology of PROMs provides a more convenient and considerably less expensive approach, because memory chips can be programmed directly by the user.

EPROM

Another type of ROM chip provides an even higher level of convenience. **It allows the stored data to be erased and new data to be written into it**. Such an erasable, reprogrammable ROM is usually called an EPROM. It provides considerable flexibility during the development phase of digital systems. Since EPROMs are capable of retaining stored information for a long time, they can be used in place of ROMs or PROMs while software is being developed. In this way, memory changes and updates can be easily made.

An EPROM cell has a structure similar to the ROM. However, the connection to ground at point P is made through a special transistor. The transistor is normally turned off, creating an open switch. It can be turned on by injecting charge into it that becomes trapped inside. Thus, an EPROM cell can be used to construct a memory in the same way as the PROM cell. **Erasure** requires dissipating the charge trapped in the transistors that form the memory cells. This **can be done by exposing the chip to ultraviolet light, which erases the entire contents of the chip.** To make this possible, EPROM chips are mounted in packages that have transparent windows.

EEPROM

An EPROM must be physically removed from the circuit for reprogramming. Also, the stored information cannot be erased selectively. The entire contents of the chip are erased when exposed to ultraviolet light. Another type of erasable PROM can be programmed, erased, and reprogrammed electrically. Such a chip is called an electrically erasable PROM, or EEPROM. It **does not have to be removed for erasure**. Moreover, it is **possible to erase the cell contents selectively.** One disadvantage of EEPROMs is that different voltages are needed for erasing, writing, and reading the stored data, which increases circuit complexity. However, this disadvantage is outweighed by the many advantages of EEPROMs. They have replaced EPROMs in practice.

FLASH MEMORY

An approach similar to EEPROM technology has given rise to flash memory devices. **A flash cell is based on a single transistor controlled by trapped charge, much like an EEPROM cell**. Also like an EEPROM, it is possible to read the contents of a single cell. The key difference is that, in a flash device, **it is only possible to write an entire block of cells**. Prior to writing, the previous contents of the block are erased. Flash devices have greater density, which leads to higher capacity and a lower cost per bit. They require a single power supply voltage, and consume less power in their operation.

The low power consumption of flash memories makes them attractive for use in portable, battery-powered equipment. Typical applications include hand-held computers, cell phones, digital cameras, and MP3 music players. In hand-held computers and cell phones, a flash memory holds the software needed to operate the equipment, thus obviating the need for a disk drive. A flash memory is used in digital cameras to store picture data.

In MP3 players, flash memories store the data that represent sound. Cell phones, digital cameras, and MP3 players are good examples of embedded systems. Single flash chips may not provide sufficient storage capacity for the applications mentioned above. Larger memory modules consisting of a number of chips are used where needed. There are two popular choices for the implementation of such modules: **flash cards and flash drives.**

Flash Cards

One way of constructing a larger module is to mount flash chips on a small card. Such flash cards have a standard interface that makes them usable in a variety of products. A card is simply plugged into a conveniently accessible slot. Flash cards with a USB interface are widely used and are commonly known as memory keys. They come in a variety of memory sizes. Larger cards may hold as much as 32 Gbytes. A minute of music can be stored in about 1 Mbyte of memory, using the MP3 encoding format. Hence, a 32-Gbyte flash card can store approximately 500 hours of music.

Flash Drives

Larger flash memory modules have been developed to replace hard disk drives, and hence are called flash drives. They are designed to fully emulate hard disks, to the point that they can be fitted into standard disk drive bays. However, the storage capacity of flash drives is significantly lower. Currently, the capacity of flash drives is on the order of 64 to 128 Gbytes. In contrast, hard disks have capacities exceeding a terabyte. Also, disk drives have a very low cost per bit.

The fact that flash drives are solid state electronic devices with no moving parts provides important advantages over disk drives. They have shorter access times, which result in a faster response. They are insensitive to vibration and they have lower power consumption, which makes them attractive for portable, battery-driven applications.

MEMORY HIERARCHY

The **ideal memory should be fast, large, and inexpensive**. A **very fast memory** can be implemented using **static RAM** chips. But, these chips are not suitable for implementing large memories, because their **basic cells are larger and consume more power than dynamic RAM cells.**

Although dynamic memory units with gigabyte capacities can be implemented at a reasonable cost, the affordable size is still small compared to the demands of large programs with voluminous data. A solution is provided by using **secondary storage**, mainly magnetic disks, to provide the required memory space. Disks are available at a **reasonable cost**, and they are used extensively in computer systems. However, they are much **slower than semiconductor memory units**. In summary, a very large amount of cost-effective storage can be provided by magnetic disks, and a large and considerably faster, yet affordable, main

memory can be built with dynamic RAM technology. This leaves the more expensive and much faster **static RAM technology to be used in smaller units where speed is of the essence, such as in cache memories.**

All of these different types of memory units are employed effectively in a computer system. The entire computer memory can be viewed as the hierarchy depicted in the below figure. The fastest access is to data held in processor registers. Therefore, if we consider the registers to be part of the memory hierarchy, then the processor registers are at the top in terms of speed of access. Of course, the registers provide only a minuscule portion of the required memory.

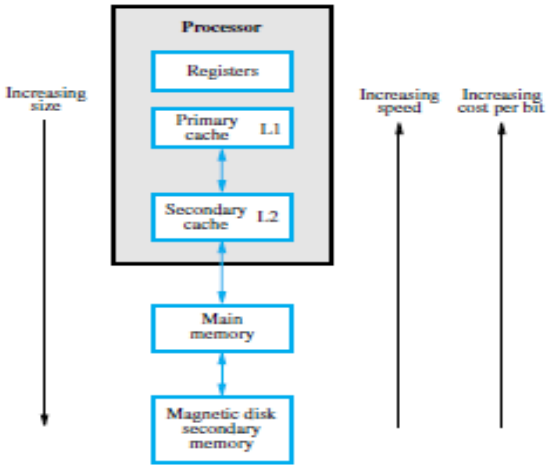

Figure: Memory hierarchy

At the next level of the hierarchy is a relatively small amount of **memory that can be implemented directly on the processor chip**. This memory, called a **processor cache**, holds copies of the instructions and data stored in a much larger memory that is provided externally. There are often two or more levels of cache. A **primary cache** is always located on the processor chip. This cache is **small and its access time is comparable to that of processor registers.** The primary cache is referred to as the level 1 (L1) cache. A larger, and hence somewhat slower, **secondary cache** is placed between the primary cache and the rest of the memory. It is referred to as the level 2 (L2) cache. Often, the L2 cache is also housed on the processor chip.

Some computers have a level 3 (L3) cache of even larger size, in addition to the L1 and L2 caches. An L3 cache, also implemented in SRAM technology, may or may not be on the same chip with the processor and the L1 and L2 caches.

The next level in the hierarchy is the main memory. This is a large memory implemented using dynamic memory components. The main memory is much larger but significantly slower than cache memories. In a computer with a processor clock of 2 GHz or higher, the access time for the main memory can be as much as 100 times longer than the access time for the L1 cache.

Disk devices provide a very large amount of inexpensive memory, and they are widely used as secondary storage in computer systems. They are very slow compared to the main memory. They represent the bottom level in the memory hierarchy. During program execution, the speed of memory access is of utmost importance. The key to managing the operation of the

hierarchical memory system is to bring the instructions and data that are about to be used as close to the processor as possible.

CACHE MEMORIES

The cache is a small and very fast memory, interposed between the processor and the main memory. Its **purpose is to make the main memory appear to the processor to be much faster than it actually is**. The effectiveness of this approach is based on a property of computer programs called **locality of reference**. Analysis of programs shows that most of their execution time is spent in routines in which many instructions are executed repeatedly. These instructions may constitute a simple loop, nested loops, or a few procedures that repeatedly call each other. The actual detailed pattern of instruction sequencing is not important—the point is that **many instructions in localized areas of the program are executed repeatedly during some time period**. This behavior manifests itself in two ways: **temporal and spatial**. The **temporal** means that a **recently executed instruction is likely to be executed again very soon**. The **spatial** aspect means that **instructions close to a recently executed instruction are also likely to be executed soon.**

Conceptually, operation of a cache memory is very simple. The memory control circuitry is designed to take advantage of the property of locality of reference. **Temporal locality suggests that whenever an information item, instruction or data, is first needed, this item should be brought into the cache,** because it is likely to be needed again soon**. Spatial locality suggests that instead of fetching just one item from the main memory to the cache, it is useful to fetch several items that are located at adjacent addresses as well.** The term **cache block refers to a set of contiguous address locations of some size**. Another term that is often used to refer to a cache block is a **cache line**.

Consider the arrangement in the below figure. When the processor issues a Read request, **the contents of a block of memory words containing the location specified are transferred into the cache**. Subsequently, when the program references any of the locations in this block, the desired contents are read directly from the cache. Usually, the **cache memory can store a reasonable number of blocks at any given time**, but this number is small compared to the total number of blocks in the main memory. The **correspondence between the main memory blocks and those in the cache is specified by a mapping function**. When the cache is full and a memory word (instruction or data) that is not in the cache is referenced, the **cache control hardware must decide which block should be removed to create space for the new block that contains the referenced word**. The collection of rules for making this decision constitutes the **cache's replacement algorithm.**

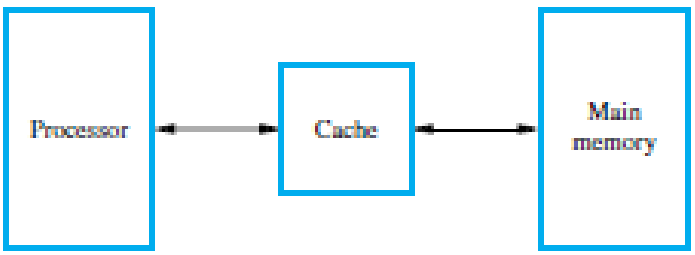

Figure: Use of a Cache memory

Cache Hits

The processor does not need to know explicitly about the existence of the cache. It simply issues Read and Write requests using addresses that refer to locations in the memory. **The cache control circuitry determines whether the requested word currently exists in**

the cache. If it does, the Read or Write operation is performed on the appropriate cache location. In this case, a **read or write hit is said to have occurred**. The **main memory is not involved when there is a cache hit in a Read operation**. For a **Write operation**, the system can **proceed in one of two ways**. In the **first technique**, called the **write-through protocol**, both the cache location and the main memory location are updated.

The **second technique** is to update only the cache location and to mark the block containing it with an associated flag bit**, often called the dirty or modified bit**. The main memory location of the word is updated later, when the block containing this marked word is removed from the cache to make room for a new block. This technique is known as the **write-back, or copy-back, protocol.**

The **write-through protocol is simpler than the write-back protocol**, but it results in unnecessary Write operations in the main memory when a given cache word is updated several times during its cache residency. The write-back protocol also involves unnecessary Write operations, because all words of the block are eventually written back, even if only a single word has been changed while the block was in the cache. The **write-back protocol is used most often**, to take advantage of the high speed with which data blocks can be transferred to memory chips.

Cache Misses

A **Read operation for a word that is not in the cache constitutes a Read miss**. It causes the **block of words containing the requested word to be copied from the main memory into the cache**. After the entire block is loaded into the cache, the particular word requested is forwarded to the processor. Alternatively, this word may be sent to the processor as soon as it is read from the main memory. The latter approach, which is called **load-through, or early restart, reduces the processor's waiting time somewhat, at the expense of more complex circuitry.**

When a **Write miss occurs** in a computer that **uses the write-through protocol**, the information is written directly into the main memory. For the write-back protocol, the block containing the addressed word is first brought into the cache, and then the desired word in the cache is overwritten with the new information.

MAPPING FUNCTIONS

There are **several possible methods for determining where memory blocks are placed in the cache**. It is instructive to describe these methods using a specific small example. Consider **a cache consisting of 128 blocks of 16 words each**, for a total of 2048 (2K) words, and assume that the main memory is addressable by a 16-bit address. **The main memory has** 64K words, which we will view as **4K blocks of 16 words each**. For simplicity, we have assumed that consecutive addresses refer to consecutive words.

DIRECT MAPPING

The simplest way to determine cache locations in which to store memory blocks is the **direct-mapping** technique. In this technique**, block j of the main memory maps onto block j modulo 128 of the cache**, as depicted in below figure. Thus, whenever one of the main memory blocks 0, 128, 256 . . . is loaded into the cache, it is stored in cache block 0. Blocks 1, 129, 257 . . . are stored in cache block 1, and so on. Since more than one memory block is

mapped onto a given cache block position, contention may arise for that position even when the cache is not full.

For example, instructions of a program may start in block 1 and continue in block 129, possibly after a branch. As this program is executed, both of these blocks must be transferred to the block-1 position in the cache. Contention is resolved by allowing the new block to overwrite the currently resident block.

With direct mapping, the replacement algorithm is trivial (less important). Placement of a block in the cache is determined by its memory address. The **memory address** can be divided into **three fields**. The **low-order 4 bits select one of 16 words in a block**. When a new block enters the cache, the **7-bit cache block field determines the cache position in which this block must be stored.** The **high-order 5 bits of the memory address of the block are stored in 5 tag bits associated with its location in the cache**. **The tag bits identify which of the 32 main memory blocks mapped into this cache position is currently resident in the cache**.

As execution proceeds, the 7-bit cache block field of each address generated by the processor points to a particular block location in the cache. The high-order 5 bits of the address are compared with the tag bits associated with that cache location. If they match, then the desired word is in that block of the cache. If there is no match, then the block containing the required word must first be read from the main memory and loaded into the cache. The direct-mapping technique is easy to implement, but it is not very flexible.

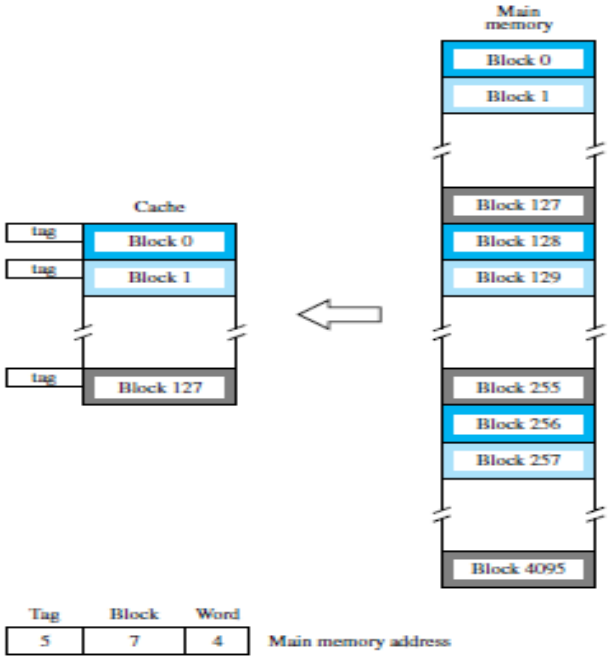

Figure: Direct-mapped cache

ASSOCIATIVE MAPPING

The below figure shows the **most flexible mapping method**, in which **a main memory block can be placed into any cache block position**. In this case, **12 tag bits are required to identify a memory block when it is resident in the cache.** The tag bits of an address received

from the processor are compared to the tag bits of each block of the cache to see if the desired block is present. This is called the **associative-mapping technique**.

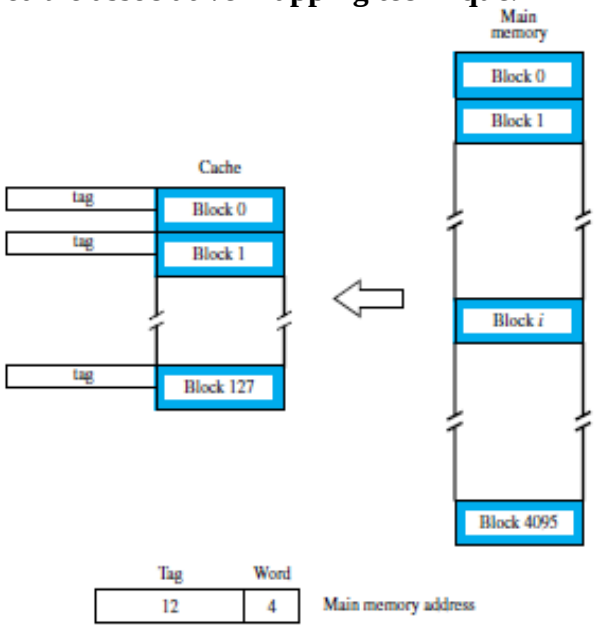

Figure: Associative Mapped cache

It gives complete freedom in choosing the cache location in which to place the memory block, resulting in a more efficient use of the space in the cache. **When a new block is brought into the cache, it replaces (ejects) an existing block only if the cache is full**. In this case, we need an algorithm to select the block to be replaced.

The **complexity** of an associative cache is **higher than that of a direct-mapped cache**, because of the **need to search all 128 tag patterns** to determine whether a given block is in the cache. To avoid a long delay, the **tags must be searched in parallel**. A search of this kind is called an **associative search.**

SET-ASSOCIATIVE MAPPING

Another approach is to use a combination of the direct- and associative-mapping techniques. The **blocks of the cache are grouped into sets, and the mapping allows a block of the main memory to reside in any block of a specific set**. Hence, the contention problem of the direct method is eased by having a few choices for block placement. At the same time, the hardware cost is reduced by decreasing the size of the associative search. An example of this **set-associative-mapping** technique is shown in below figure for a cache with two blocks per set.

In this case, memory blocks 0, 64, 128. . . 4032 map into cache set 0, and they can occupy either of the two block positions within this set. Having 64 sets means that the 6-bit set field of the address determines which set of the cache might contain the desired block. The tag field of the address must then be associatively compared to the tags of the two blocks of the set to check if the desired block is present. This two-way associative search is simple to implement.

The number of blocks per set is a parameter that can be selected to suit the requirements of a particular computer. For the main memory and cache sizes in the above figure, four blocks per set can be accommodated by a 5-bit set field, eight blocks per set by a 4-bit set field, and

so on. The extreme condition of 128 blocks per set requires no set bits and corresponds to the fully-associative technique, with 12 tag bits. The other extreme of one block per set is the direct-mapping method. A cache that has **k blocks** per set is referred to as a **k-way set-associative cache.**

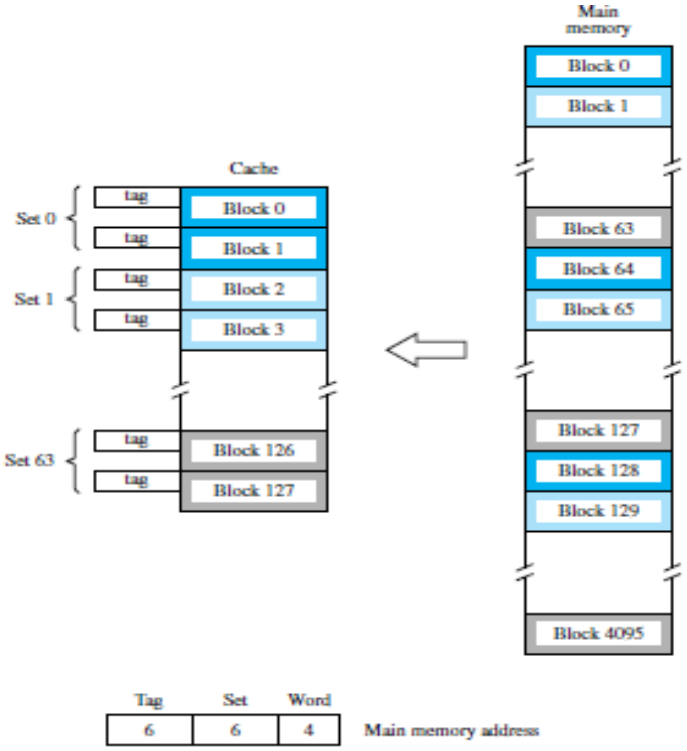

Figure: Set-associative-mapped cache with two blocks per set.

REPLACEMENT ALGORITHMS

In a direct-mapped cache, the position of each block is predetermined by its address; hence, the replacement strategy is trivial. In associative and set-associative caches there exists some flexibility. **When a new block is to be brought into the cache and all the positions that it may occupy are full**, the cache controller must decide which of the **old blocks to overwrite**. This is an important issue, because the decision can be a strong determining factor in system performance.

In general, the objective is to keep blocks in the cache that are likely to be referenced in the near future. But, it is not easy to determine which blocks are about to be referenced. The property of **locality of reference** in programs gives a clue to a reasonable strategy. Because program execution usually stays in localized areas for reasonable periods of time, there is a high probability that the blocks that have been referenced recently will be referenced again soon. Therefore, when a block is to be overwritten, it is sensible to overwrite the one that has gone the longest time without being referenced. **This block is called the least recently used (LRU) block, and the technique is called the LRU replacement algorithm.**

The LRU algorithm has been used extensively. Although it performs well for many access patterns, it can lead to poor performance in some cases. For example, it produces disappointing results when accesses are made to sequential elements of an array that is slightly too large to fit into the cache. Performance of the LRU algorithm can be improved by introducing a small amount of randomness in deciding which block to replace.

Several other replacement algorithms are also used in practice. An intuitively reasonable rule would be **to remove the "oldest" block from a full set when a new block must be brought in.** However, because this algorithm does not take into account the recent pattern of access to blocks in the cache, it is generally not as effective as the LRU algorithm in choosing the best blocks to remove.

The simplest algorithm is to **randomly choose the block to be overwritten**. Interestingly enough, this simple algorithm has been found to be quite effective in practice.

PERFORMANCE CONSIDERATIONS:

The key factors in success of a computer are performance and cost. Performance depends on how fast machine instructions can be brought into the processor for execution and how fast they can be executed. The speed and efficiency of data transfer of the between various levels of memory hierarchy are also important for performance. It is beneficial if transfers to and from the faster units to and from the faster units can be **done at a rate equal to that of the faster unit.** This is **not possible if both the slow and the fast units are accessed in the same manner**, **but it can be achieved when parallelism is used in the organization of the slower unit.** An effective way to introduce parallelism is to use an interleaved organization.

Hit Rate and Miss Penalty:

An indicator of the effectiveness of a particular implementation of the memory hierarchy is the success rate in accessing information at various levels of the hierarchy. The successful access to data in a cache is called a **hit**. **The number of hits stated as a fraction of all attempted accesses is called the hit rate, and the miss rate is the number of misses stated as a fraction of attempted accesses.**

Ideally, the entire memory hierarchy would appear to the processor as a single memory unit that has the access time of the cache on the processor chip and the size of the magnetic disk. High hit rates well over 0.9 are essential for high-performance computers.

Performance is adversely affected by the actions that need to be taken when a miss occurs. **A performance penalty is incurred because of the extra time needed to bring a block of data from a slower unit in the memory hierarchy to a faster unit**. During that period, the processor is stalled waiting for instructions or data. The total access time seen by the processor when a miss occurs as the **miss penalty**.

Consider a system with only one level of cache. In this case, the miss penalty consists almost entirely of the time to access a block of data in the main memory. **Let h be the hit rate, M the miss penalty, and C the time to access information in the cache.** Thus, the **average access time** experienced by the processor is

$$t_{avg} = hC + (1 - h)M$$

Caches on the Processor Chip

In high-performance processors, two levels of caches are normally used, separate L1 caches for instructions and data and a larger L2 cache. These caches are often implemented on the processor chip. In this case, the L1 caches must be very fast, as they determine the memory access time seen by the processor. The L2 cache can be slower, but it should be much larger

than the L1 caches to ensure a high hit rate. Its speed is less critical because it only affects the miss penalty of the L1 caches. A typical computer may have L1 caches with capacities of tens of kilobytes and an L2 cache of hundreds of kilobytes or possibly several megabytes.

Including an L2 cache further reduces the impact of the main memory speed on the performance of a computer. Its effect can be assessed by observing that the average access time of the L2 cache is the miss penalty of either of the L1 caches. For simplicity, we will assume that the hit rates are the same for instructions and data. Thus, the average access time experienced by the processor in such a system is:

$$t_{avg} = h1C1 + (1 - h1)(h2C2 + (1 - h2)M)$$

where
h1 is the hit rate in the L1 caches.
h2 is the hit rate in the L2 cache.
C1 is the time to access information in the L1 caches.
C2 is the miss penalty to transfer information from the L2 cache to an L1 cache.
M is the miss penalty to transfer information from the main memory to the L2 cache.

Of all memory references made by the processor, the number of misses in the L2 cache is given by (1 − h1)(1 − h2). If both h1 and h2 are in the 90 percent range, then the number of misses in the L2 cache will be less than one percent of all memory accesses. This makes the value of M, and in turn the speed of the main memory, less critical.

SECONDARY STORAGE

The semiconductor memories cannot be used to provide all of the storage capability needed in computers. Their main limitation is the cost per bit of stored information. The large storage requirements of most computer systems are economically realized in the form of magnetic and optical disks, which are usually referred to as secondary storage devices.

MAGNETIC HARD DISKS

The storage medium in a magnetic-disk system consists of one or more disk platters mounted on a common spindle. A thin magnetic film is deposited on each platter, usually on both sides. The assembly is placed in a drive that causes it to rotate at a constant speed. The magnetized surfaces move in close proximity to read/write heads, as shown in Figure (a). Data are stored on concentric tracks, and the read/write heads move radially to access different tracks.

Each read/write head consists of a magnetic yoke and a magnetizing coil, as indicated in Figure (b). Digital information can be stored on the magnetic film by applying current pulses of suitable polarity to the magnetizing coil. This causes the magnetization of the film in the area immediately underneath the head to switch to a direction parallel to the applied field. The same head can be used for reading the stored information. In this case, changes in the magnetic field in the vicinity of the head caused by the movement of the film relative to the yoke induce a voltage in the coil, which now serves as a sense coil. The polarity of this voltage is monitored by the control circuitry to determine the state of magnetization of the film. Only changes in the magnetic field under the head can be sensed during the Read operation. Therefore, if the binary states 0 and 1 are represented by two opposite states of magnetization, a voltage is induced in the head only at 0-to-1 and at 1-to-0 transitions in the bit stream. A long string of 0s or 1s causes an induced voltage only at the beginning and end of the string. Therefore, to determine

the number of consecutive 0s or 1s stored, a clock must provide information for synchronization.

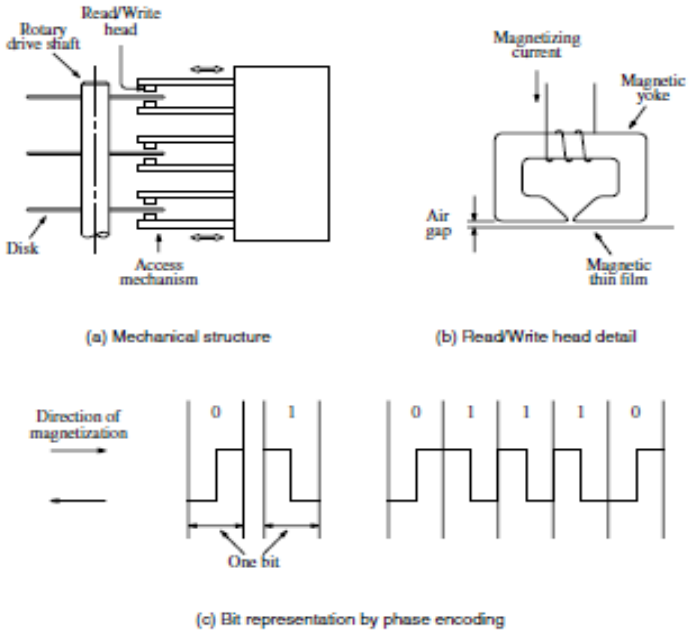

Figure: Magnetic disk principles.

In some early designs, a clock was stored on a separate track, on which a change in magnetization is forced for each bit period. Using the clock signal as a reference, the data stored on other tracks can be read correctly. The modern approach is to combine the clocking information with the data. Several different techniques have been developed for such encoding. One simple scheme, depicted in Figure (c), is known as **phase encoding or Manchester encoding.** In this scheme, changes in magnetization occur for each data bit, as shown in the figure. Clocking information is provided by the change in magnetization at the midpoint of each bit period. **The drawback of Manchester encoding is its poor bit-storage density.** The space required to represent each bit must be large enough to accommodate two changes in magnetization.

Read/write heads must be maintained at a very small distance from the moving disk surfaces in order to achieve high bit densities and reliable Read and Write operations. When the disks are moving at their steady rate, air pressure develops between the disk surface and the head and forces the head away from the surface. This force is counterbalanced by a spring-loaded mounting arrangement that presses the head toward the surface. The flexible spring connection between the head and its arm mounting permits the head to fly at the desired distance away from the surface in spite of any small variations in the flatness of the surface.

In most modern disk units, the disks and the read/write heads are placed in a sealed, air filtered enclosure. This approach is known as **Winchester technology**. In such units, the read/write heads can operate closer to the magnetized track surfaces, because dust particles, which are a problem in unsealed assemblies, are absent. The closer the heads are to a track surface, the more densely the data can be packed along the track, and the closer the tracks can be to each other. Thus, Winchester disks have a larger capacity for a given physical size compared to unsealed units. Another advantage of Winchester technology is that data integrity tends to be greater in sealed units, where the storage medium is not exposed to contaminating elements.

The **read/write heads of a disk system are movable**. There is one head per surface. All heads are mounted on a comb-like arm that can move radially across the stack of disks to provide access to individual tracks, as shown in Figure (a). To read or write data on a given track, the read/write heads must first be positioned over that track.

The **disk system consists of three key parts. One part** is the assembly of **disk platters**, which is usually **referred to as the disk**. The **second part** comprises the electromechanical mechanism that spins the disk and moves the read/write heads; it is called the **disk drive**. The **third part is the disk controller,** which is the electronic circuitry that **controls the operation of the system**.

Organization and accessing of data on a disk

The organization of data on a disk is illustrated in the below figure. **Each surface is divided into concentric tracks, and each track is divided into sectors.** The set of corresponding tracks on all surfaces of a stack of disks forms a logical cylinder. All tracks of a cylinder can be accessed without moving the read/write heads. **Data are accessed by specifying the surface number, the track number, and the sector number.** Read and Write operations always start at sector boundaries.

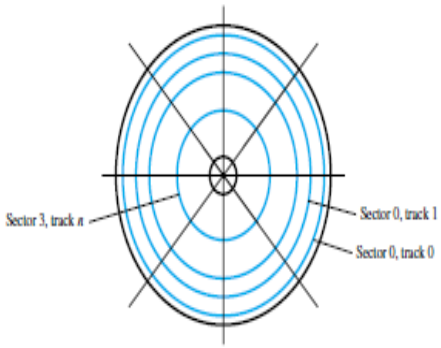

Figure: Organization of one surface of a disk.

Data bits are stored serially on each track. Each sector may contain 512 or more bytes. The data are preceded by a sector header that contains identification (addressing) information used to find the desired sector on the selected track. Following the data, there are additional bits that constitute an error-correcting code (ECC). The ECC bits are used to detect and correct errors that may have occurred in writing or reading the data bytes. There is a small inter-sector gap that enables the disk control circuitry to distinguish easily between two consecutive sectors.

An unformatted disk has no information on its tracks. The formatting process writes markers that divide the disk into tracks and sectors. During this process, the disk controller may discover some sectors or even whole tracks that are defective. The disk controller keeps a record of such defects and excludes them from use. The formatting information comprises sector headers, ECC bits, and inter-sector gaps. The capacity of a formatted disk, after accounting for the formatting information overhead, is the proper indicator of the disk's storage capability. After formatting, the disk is divided into logical partitions.

The above figure indicates that each track has the same number of sectors, which means that all tracks have the same storage capacity. In this case, the stored information is packed more densely on inner tracks than on outer tracks. It is also possible to increase the storage

density by placing more sectors on the outer tracks, which have longer circumference. This would be at the expense of more complicated access circuitry.

Access Time

There are **two components involved in the time delay** between the disk receiving an address and the beginning of the actual data transfer. The **first, called the seek time, is the time required to move the read/write head to the proper track**. This time depends on the initial position of the head relative to the track specified in the address. Average values are in the 5- to 8-ms range. The **second** component is the **rotational delay, also called latency time, which is the time taken to reach the addressed sector after the read/write head is positioned over the correct track.** On average, this is the time for half a rotation of the disk. **The sum of these two delays is called the disk access time**. If only a few sectors of data are accessed in a single operation, the access time is at least an order of magnitude longer than the time it takes to transfer the data.

Data Buffer/Cache

A disk drive is connected to the rest of a computer system using some standard interconnection scheme, such as SCSI or SATA. The interconnection hardware is usually capable of transferring data at much higher rates than the rate at which data can be read from disk tracks. **An efficient way to deal with the possible differences in transfer rates is to include data buffer in the disk unit**. The buffer is a semiconductor memory, capable of storing a few megabytes of data. The requested data are transferred between the disk tracks and the buffer at a rate dependent on the rotational speed of the disk. Transfers between the data buffer and the main memory can then take place at the maximum rate allowed by the interconnect between them.

The data buffer in the disk controller can also be used to provide a caching mechanism for the disk. When a Read request arrives at the disk, the controller can first check to see if the desired data are already available in the buffer. If so, the data are transferred to the memory in microseconds instead of milliseconds. Otherwise, the data are read from a disk track in the usual way, stored in the buffer, and then transferred to the memory. Because of locality of reference, a subsequent request is likely to refer to data that sequentially follow the data specified in the current request. In anticipation of future requests, the disk controller may read more data than needed and place them into the buffer. When used as a cache, the buffer is typically large enough to store entire tracks of data. So, a possible strategy is to begin transferring the contents of the track into the data buffer as soon as the read/write head is positioned over the desired track.

Disk Controller

Operation of a disk drive is controlled by a disk controller circuit, which also provides an interface between the disk drive and the rest of the computer system. One disk controller may be used to control more than one drive.

A disk controller that communicates directly with the processor contains a number of registers that can be read and written by the operating system. Thus, communication between the OS and the disk controller is achieved in the same manner as with any I/O interface. The disk controller uses the DMA scheme to transfer data between the disk and the main memory.

Actually, these transfers are from/to the data buffer, which is implemented as a part of the disk controller module. The OS initiates the transfers by issuing Read and Write requests, which entail loading the controller's registers with the necessary addressing and control information. Typically, this information includes:

Main memory address— The address of the first main memory location of the block of words involved in the transfer.

Disk address— The location of the sector containing the beginning of the desired block of words.

Word count— The number of words in the block to be transferred.

On the disk drive side, the controller's major functions are:

Seek— Causes the disk drive to move the read/write head from its current position to the desired track.

Read— Initiates a Read operation, starting at the address specified in the disk address register. Data read serially from the disk are assembled into words and placed into the data buffer for transfer to the main memory. The number of words is determined by the word count register.

Write— Transfers data to the disk, using a control method similar to that for Read operations.

Error checking— Computes the error correcting code (ECC) value for the data read from a given sector and compares it with the corresponding ECC value read from the disk. In the case of a mismatch, it corrects the error if possible; otherwise, it raises an interrupt to inform the OS that an error has occurred. During a Write operation, the controller computes the ECC value for the data to be written and stores this value on the disk.

FLOPPY DISKS

The disks discussed above are known as **hard or rigid disk units**. Floppy disks are **smaller, simpler, and cheaper disk units that consist of a flexible, removable, plastic diskette coated with magnetic material.** The diskette is enclosed in a plastic jacket, which has an opening where the read/write head can be positioned. A hole in the center of the diskette allows a spindle mechanism in the disk drive to position and rotate the diskette.

The main feature of floppy disks is their low cost and shipping convenience. However, they have much smaller storage capacities, longer access times, and higher failure rates than hard disks. Floppy disks are replaced by CDs, DVDs, and flash cards as portable storage media.

RAID DISK ARRAYS

Processor speeds have increased dramatically. At the same time, access times to disk drives are still on the order of milliseconds, because of the limitations of the mechanical motion involved. One way to reduce access time is to use multiple disks operating in parallel. In 1988, researchers at the University of California-Berkeley proposed such a storage system. They called it RAID, for Redundant Array of Inexpensive Disks. (Since all disks are now inexpensive, the acronym was later reinterpreted as **Redundant Array of Independent**

Disks.) Using multiple disks also makes it possible to improve the reliability of the overall system. Different configurations were proposed, and many more have been developed since.

The basic configuration, known as RAID 0, is simple. **A single large file is stored in several separate disk units by dividing the file into a number of smaller pieces and storing these pieces on different disks. This is called data striping.** When the file is accessed for a Read operation, all disks access their portions of the data in parallel. As a result, the rate at which the data can be transferred is equal to the data rate of individual disks times the number of disks. However, access time, that is, seek and rotational delay needed to locate the beginning of the data on each disk, is not reduced. Since each disk operates independently, access times vary. Individual pieces of the data are buffered, so that the complete file can be reassembled and transferred to the memory as a single entity.

Various RAID configurations form a hierarchy, with each level in the hierarchy providing additional features. For example, RAID 1 is intended to provide better reliability by storing identical copies of the data on two disks rather than just one. The two disks are said to be mirrors of each other. If one disk drive fails, all Read and Write operations are directed to its mirror drive. Other levels of the hierarchy achieve increased reliability through various parity-checking schemes, without requiring a full duplication of disks. Some also have error-recovery capability.

The RAID concept has gained commercial acceptance. RAID systems are available from many manufacturers for use with a variety of operating systems.

OPTICAL DISKS

Storage devices can also be implemented using optical means. The compact disk (CD), used in audio systems, was the first practical application of this technology. Soon after, the optical technology was adapted to the computer environment to provide a high capacity read-only storage medium known as a CD-ROM.

The first generation of CDs was developed in the mid-1980s by the Sony and Philips companies. The technology exploited the possibility of using a digital representation for analog sound signals. To provide high-quality sound recording and reproduction, 16-bit samples of the analog signal are taken at a rate of 44,100 samples per second. Initially, CDs were designed to hold up to 75 minutes, requiring a total of about 3×10^9 bits (3 gigabits) of storage. Since then, higher-capacity devices have been developed.

CD Technology

The optical technology that is used for CD systems makes use of the fact that laser light can be focused on a very small spot. A laser beam is directed onto a spinning disk with tiny indentations arranged to form a long spiral track on its surface. The indentations reflect the focused beam toward a photodetector, which detects the stored binary patterns. The laser emits a coherent light beam that is sharply focused on the surface of the disk. Coherent light consists of synchronized waves that have the same wavelength. If a coherent light beam is combined with another beam of the same kind, and the two beams are in phase, the result is a brighter beam. But, if the waves of the two beams are 180 degrees out of phase, they cancel each other. Thus, a photodetector can be used to detect the beams. It will see a **bright spot** in the first case and a **dark spot** in the second case.

Across-section of a small portion of a CD is shown in below figure (a). The bottom layer is made of transparent **polycarbonate plastic**, which serves as a clear glass base. The surface of this plastic is programmed to store data by indenting it with **pits.** The unindented parts are called **lands.** A thin layer of reflecting aluminium material is placed on top of a programmed disk. The aluminium is then covered by a protective acrylic. Finally, the topmost layer is deposited and stamped with a label. The total thickness of the disk is 1.2 mm, almost all of it contributed by the polycarbonate plastic. The other layers are very thin.

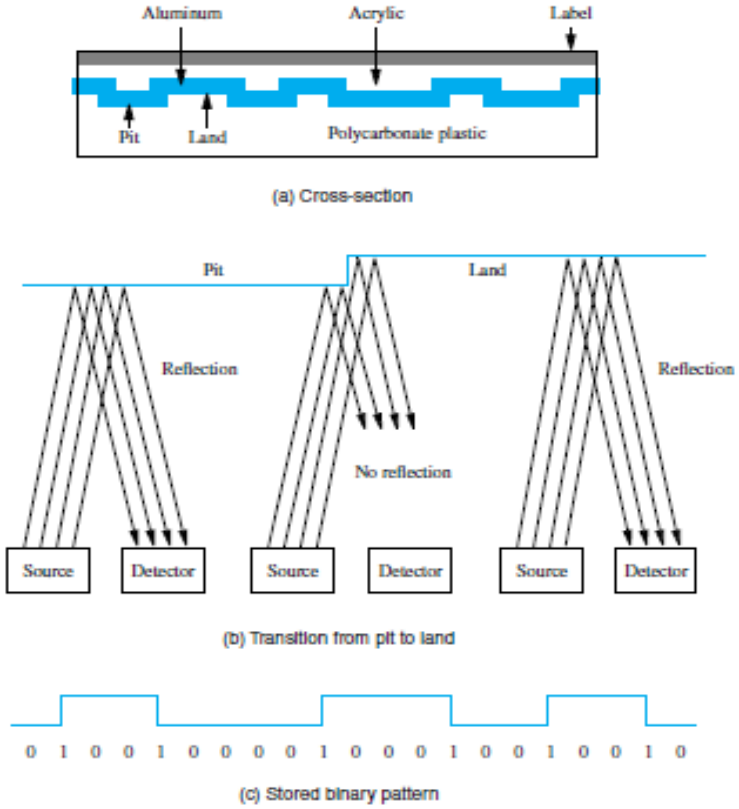

Figure: Optical disk

The laser source and the photodetector are positioned below the polycarbonate plastic. The emitted beam travels through the plastic layer, reflects off the aluminium layer, and travels back toward the photodetector. Note that from the laser side, the pits actually appear as bumps rising above the lands.

Figure (b) shows what happens as the laser beam scans across the disk and encounters a transition from a pit to a land. Three different positions of the laser source and the detector are shown, as would occur when the disk is rotating. When the light reflects solely from a pit, or from a land, the detector sees the reflected beam as a bright spot. But, a different situation arises when the beam moves over the edge between a pit and the adjacent land. The pit is one quarter of a wavelength closer to the laser source. Thus, the reflected beams from the pit and the adjacent land will be 180 degrees out of phase, cancelling each other. Hence, the detector will not see a reflected beam at pit-land and land-pit transitions, and will detect a dark spot.

Figure (c) depicts several transitions between lands and pits. If each transition, detected as a dark spot, is taken to denote the binary value 1, and the flat portions represent 0s, then the detected binary pattern will be as shown in the figure. This pattern is not a direct representation

of the stored data. CDs use a complex encoding scheme to represent data. Each byte of data is represented by a 14-bit code, which provides considerable error detection capability.

The pits are arranged on a long track on the surface of the disk, spiralling from the middle of the disk toward the outer edge. But, it is customary to refer to each circular path spanning 360 degrees as a separate track, which is analogous to the terminology used for magnetic disks. The CD is 120mmin diameter, with a 15-mm hole in the center. The tracks cover the area from a 25-mm radius to a 58-mm radius. The space between the tracks is 1.6 microns. Pits are 0.5 microns wide and 0.8 to 3 microns long. There are more than 15,000 tracks on a disk. If the entire track spiral were unravelled, it would be over 5 km long.

CD-ROM

Since CDs store information in a binary form, they are suitable for use as a storage medium in computer systems. **The main challenge is to ensure the integrity of stored data.** Because the pits are very small, it is difficult to implement all of the pits perfectly. In audio and video applications, some errors in the data can be tolerated, because they are unlikely to affect the reproduced sound or image in a perceptible way. However, such errors are not acceptable in computer applications. Since physical imperfections cannot be avoided, it is necessary to use additional bits to provide error detection and correction capability. **The CDs used to store computer data are called CD-ROM**s, because, like semiconductor ROM chips, their contents can only be read.

Stored data are organized on CD-ROM tracks in the form of blocks called sectors. There are several different formats for a sector. One format, known **as Mode 1, uses 2352-byte sectors**. There is a **16-byte header** that contains a synchronization field used to detect the beginning of the sector and addressing information used to identify the sector. This is followed by **2048 bytes of stored data**. At the end of the sector, there are **288 bytes** used to implement the error-correcting scheme. The number of sectors per track is variable; there are more sectors on the longer outer tracks. With the Mode 1 format, a CD-ROM has a storage capacity of about 650 Mbytes.

Error detection and correction is done at more than one level. Each byte of information stored on a CD is encoded using a 14-bit code that has some error-correcting capability. This code can correct single-bit errors. Errors that occur in short bursts, affecting several bits, are detected and corrected using the error-checking bits at the end of the sector.

CD-ROM drives **operate at a number of different rotational speeds**. The basic speed, known as **1X**, is 75 sectors per second. This provides a data rate of 153,600 bytes/s **(150 Kbytes/s)**, using the Mode 1 format. Higher speed CD-ROM drives are identified in relation to the basic speed. Thus, a 56X CD-ROM has a data transfer rate that is 56 times that of the 1X CD-ROM, or about **6 Mbytes/s**. This transfer rate is considerably lower than the transfer rates of magnetic hard disks, which are in the range of tens of megabytes per second.

Another significant difference in performance is the seek time, which in CD-ROMs may be several hundred milliseconds. So, **in terms of performance, CD-ROMs are clearly inferior to magnetic disks**. Their attraction lies in their small physical size, low cost, and ease of handling as a removable and transportable mass-storage medium. As a result, they are widely used for the distribution of software, textbooks, application programs, video games, and so on.

CD-Recordable

The CDs described above are read-only devices, in which the information is stored at the time of manufacture. First, a master disk is produced using a high-power laser to burn holes that correspond to the required pits. A mold is then made from the master disk, which has bumps in the place of holes. Copies are made by injecting molten polycarbonate plastic into the mold to make CDs that have the same pattern of holes (pits) as the master disk. This process is clearly suitable only for volume production of CDs containing the same information.

A new type of CD was developed in the late 1990s on which data can be easily recorded by a computer user. It is known as CD-Recordable **(CD-R).**A shiny spiral track covered by an organic dye is implemented on a disk during the manufacturing process. Then, a laser in a CD-R drive burns pits into the organic dye. The burned spots become opaque. They reflect less light than the shiny areas when the CD is being read. This process is irreversible, which means that the written data are stored permanently. **Unused portions of a disk can be used to store additional data at a later time.**

CD-Rewritable

The most flexible CDs are those that can be written multiple times by the user. They are known as **CD-RWs** (CD-Rewritable). The basic structure of CD-RWs is similar to the structure of CD-Rs. Instead of using an organic dye in the recording layer, an alloy of silver, indium, antimony, and tellurium is used. This alloy has interesting and useful behavior when it is heated and cooled. If it is heated above its melting point (500 degrees C) and then cooled down, it goes into an amorphous state in which it absorbs light. But, if it is heated only to about 200 degrees C and this temperature is maintained for an extended period, a process known as annealing takes place, which leaves the alloy in a crystalline state that allows light to pass through. If the crystalline state represents land area, pits can be created by heating selected spots past the melting point. The stored data can be erased using the annealing process, which returns the alloy to a uniform crystalline state. A reflective material is placed above the recording layer to reflect the light when the disk is read.

A **CD-RW drive uses three different laser powers**. The **highest power** is used to **record the pits**. The **middle power** is used to **put the alloy into its crystalline state**; it is referred to as the "**erase power**." The **lowest power** is used to **read the stored information**.

CD drives designed to read and write CD-RW disks can usually be used with other compact disk media. They can read CD-ROMs and can read and write CD-Rs. They are designed to meet the requirements of standard interconnection interfaces, such as SATA and USB.

CD-RW disks provide low-cost storage media. They are suitable for archival storage of information that may range from databases to photographic images. They can be used for low-volume distribution of information, just like CD-Rs, and for backup purposes. The CD-RW technology has made CD-Rs less relevant because it offers superior capability at only slightly higher cost.

DVD Technology

The success of CD technology and the continuing quest for greater storage capability has led to the development of **DVD (Digital Versatile Disk)** technology. The first DVD standard was defined in 1996 by a consortium of companies, with the objective of being able to store a full-length movie on one side of a DVD disk. The physical size of a DVD disk is the

same as that of CDs. The disk is 1.2 mm thick, and it is 120 mm in diameter. Its storage capacity is made much larger than that of CDs by several design changes:

• A red-light laser with a wavelength of 635 nm is used instead of the infrared light laser used in CDs, which has a wavelength of 780 nm. The shorter wavelength makes it possible to focus the light to a smaller spot.
• Pits are smaller, having a minimum length of 0.4 micron.
• Tracks are placed closer together; the distance between tracks is 0.74 micron.

Using these improvements leads to **a DVD capacity of 4.7 Gbytes.**

Further increases in capacity have been achieved by going to two-layered and two-sided disks. The **single-layered single-sided disk**, defined in the standard as **DVD-5**, has a structure that is almost the same as the CD. A **double-layered disk makes use of two layers on which tracks are implemented on top of each other**. The first layer is the clear base, as in CD disks. But, instead of using reflecting aluminium, the lands and pits of this layer are covered by a translucent material that acts as a semi-reflector. The surface of this material is then also programmed with indented pits to store data. A reflective material is placed on top of the second layer of pits and lands. The disk is read by focusing the laser beam on the desired layer. When the beam is focused on the first layer, sufficient light is reflected by the translucent material to detect the stored binary patterns. When the beam is focused on the second layer, the light reflected by the reflective material corresponds to the information stored on this layer. In both cases, the layer on which the beam is not focused reflects a much smaller amount of light, which is eliminated by the detector circuit as noise. The total storage capacity of both layers is **8.5 Gbytes**. This disk is called **DVD-9** in the standard.

Two single-sided disks can be put together to form a sandwich-like structure where the top disk is turned upside down. This can be done with single-layered disks, as specified in **DVD-10**, giving a composite disk with a capacity **of 9.4 Gbytes**. It can also be done with the double-layered disks, as specified **in DVD-18**, yielding a capacity of **17 Gbytes**. Access times for DVD drives are similar to CD drives. However, when the DVD disks rotate at the same speed, the data transfer rates are much higher because of the higher density of pits.

FUNDAMENTAL CONCEPTS

A typical computing task consists of a series of operations specified by a sequence of machine-language instructions that constitute a program. The processor fetches one instruction at a time and performs the operation specified. Instructions are fetched from successive memory locations until a branch or a jump instruction is encountered. The processor uses the program counter, PC, to keep track of the address of the next instruction to be fetched and executed. After fetching an instruction, the contents of the PC are updated to point to the next instruction in sequence. A branch instruction may cause a different value to be loaded into the PC.

When an instruction is fetched, it is placed in the instruction register, IR, from where it is interpreted, or decoded, by the processor's control circuitry. The IR holds the instruction until its execution is completed.

Consider a 32-bit computer in which each instruction is contained in one word in the memory, as in RISC-style instruction set architecture. To execute an instruction, the processor has to perform the following steps:

1. Fetch the contents of the memory location pointed to by the PC. The contents of this location are the instruction to be executed; hence they are loaded into the IR. In register transfer notation, the required action is **IR ← [[PC]]**

2. Increment the PC to point to the next instruction. Assuming that the memory is byte addressable, the PC is incremented by 4; that is **PC ← [PC] + 4**

3. Carry out the operation specified by the instruction in the IR.

Fetching an instruction and loading it into the IR is usually referred to as the instruction **fetch phase**. Performing the operation specified in the instruction constitutes the instruction **execution phase**.

The main building blocks of a processor are given in the below figure. The can be organized and interconnected in a variety of ways. Given figure shows an organization in which the ALU and all other registers are interconnected via single common bus.

The data and address lines of the external memory bus are connected to the internal processor bus via the memory data register (MDR) and memory address register (MAR) respectively. **MDR has 2 inputs and 2 outputs**. Data may be loaded into MDR either from memory-bus **(external)** or from processor-bus **(internal).** The data stored in MDR may be placed on either bus.

The input of MAR is connected to internal-bus, and its output is connected to external-bus. The control lines of the memory bus are connected to Instruction-decoder and control logic block. This unit is responsible for issuing the signals that control the operation of all the units inside the processor and for interacting with memory bus.

The number and use of processor registers R0 through R (n-1) vary considerably from one processor to another. Registers may be provided for general purpose use by programmer. Some may be dedicated as special purpose registers, such as index registers or stack pointers. Three registers **Y, Z, and TEMP** are used by processor for temporary storage during execution of some

instructions. These are **transparent to the programmer** i.e. programmer need not be concerned with them because they are never referenced explicitly by any instruction. These registers are never used for storing data generated by one instruction for later use by another instruction.

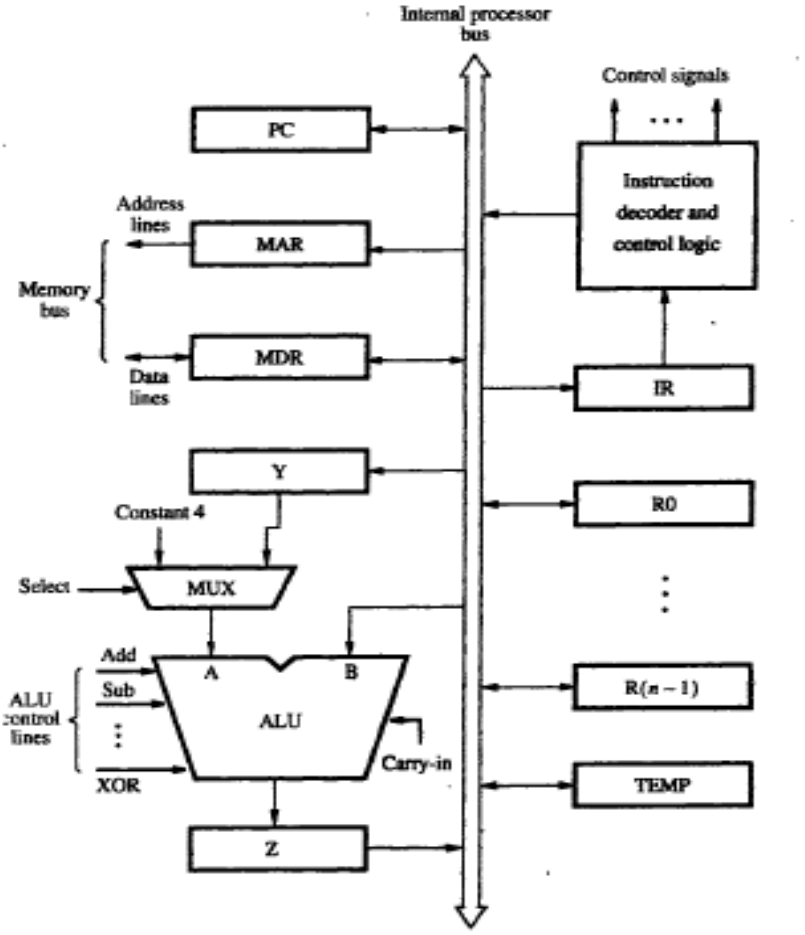

Figure: Single bus organization of the data path inside a processor

The Multiplexer MUX selects either output of register Y or constant-value 4 (is used to increment PC content) to be provided as input A of ALU. We will refer to the two possible values of the MUX control input select as select4 and selectY for selecting constant 4 or register Y, respectively. B input of ALU is obtained directly from processor-bus.

As instruction execution progresses, data are transferred from one register to another, often passing through ALU to perform arithmetic or logic operation. **The instruction decoder and control logic unit** is responsible for implementing the actions specified by the instruction loaded in the IR register. The decoder generated the control signals needed to select the registers involved and direct the data transfers. **The registers, ALU, and the interconnecting bus are** collectively referred to as the **data path.**

With few exceptions, the operation specified by an instruction can be carried out by performing one or more of the following actions:

• Read the contents of a given memory location and load them into a processor register.
• Read data from one or more processor registers.
• Perform an arithmetic or logic operation and place the result into a processor register.
• Store data from a processor register into a given memory location.

REGISTER TRANSFERS

Instruction execution involves a sequence of steps in which data are transferred from one register to another. For each register, two control signals are used to place the contents of that register on the bus or to load the data on the bus into the register. This is represented symbolically in below figure.

The input and output of register Ri is connected to bus via switches controlled by the signals: Ri_{in} & Ri_{out} respectively. These are called gating signals. **When Ri_{in} =1, data on bus is loaded into Ri. Similarly, when Ri_{out}=1, the contents of Ri is placed on bus. When Ri_{out}=0, bus can be used for transferring data from other registers.**

Suppose that we wish to transfer the contents of register R1 to register R4. This can be accomplished as follows.

- Enable the output of register R1 by setting $R1_{out}$ to 1. This places the contents of R1 on the processor bus.
- Enable the input of register R4 by setting $R4_{in}$ to 1. This loads data from the processor bus into register R4.

All operations and data transfers within the processor take place within time-periods defined by **the processor- clock.** The control signals that govern a particular transfer are asserted at the start of the clock cycle.

When edge-triggered flip-flops are not used, two or more clock-signals may be needed to guarantee proper transfer of data. This is known as **multiphase clocking**.

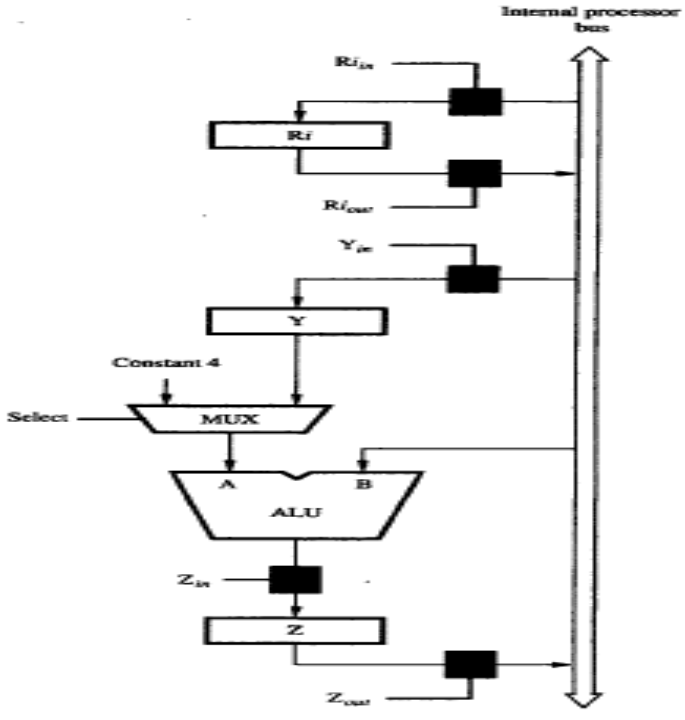

Figure: Input and output gating of processor registers.

An implementation for one bit of register Ri is shown below. A 2-input multiplexer is used to select the data applied to the input of an edge-triggered D flip-flop. When control input **Ri_{in} =1**, the multiplexer **selects data on the bus**. This data will be loaded into flip-flop at rising-edge of clock. When **Ri_{in} =0**, the multiplexer **feeds back the value currently stored in flip-flop.**

The Q output of flip-flop is connected to bus via a tri-state gate. **When Ri_{out} =0, the gate's output is in the high-impedance state** (electrically disconnected)**.** This corresponds to the open- circuit state of a switch. When **Ri_{out} =1**, the **gate drives the bus to 0 or 1**, **depending on the value of Q.**

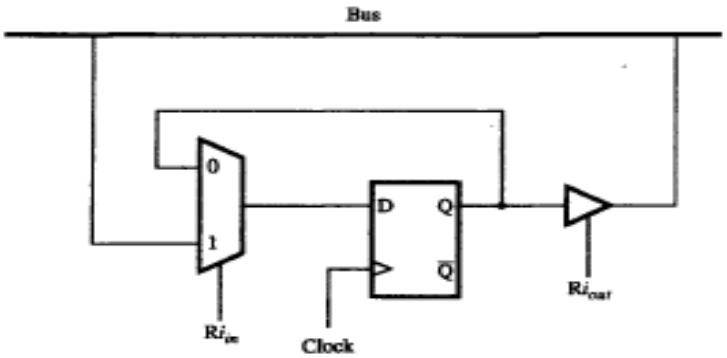

Figure: Input and output gating for one register bit

PERFORMING AN ARITHMETIC OR LOGIC OPERATION

The **ALU** is a combinational circuit that has no internal storage. It performs arithmetic and logic operations on the **two operands applied to its A and B inputs**. One of the operands is the output of the multiplexer MUX and other operand is obtained directly from bus. The result produced by the ALU is stored temporarily in register Z.

The sequence of operations to add the contents of register R1 to those of register R2 and store the result in register R3 is

1. R1out, Yin //transfer the contents of R1 to Y register
2. R2out, SelectY, Add, Zin //R2 contents are transferred directly to B input of ALU.
   // the numbers of added. Sum stored in register Z
3. Zout, R3in //sum is transferred to register R3

The signals are activated for the duration of the clock cycle corresponding to that step. All other signals are inactive.

FETCHING A WORD FROM MEMORY

To fetch a word of information (instruction/ data) from memory, the processor has to specify the address of the memory location where this information is stored and request a read operation. This applies whether the information to be fetched represents an instruction in a program or an operand specified by an instruction. The **processor transfers required address to MAR**, whose output is connected to address-lines of memory-bus. At the same time, the processor uses the control lines of the memory bus to indicate that a **read operation** is needed. When the requested data are received from memory they are stored in **MDR,** from where they can be transferred to other registers in the processor.

The connections for register MDR are illustrated in the below figure. It has four control signals **MDR_{in} and MDR_{out} control the connection to the internal bus,** and **MDR_{inE} and MDR_{outE} control the connection of external bus.**

During memory read and write operations, the timing of the internal processor operations must be coordinated with the response of the addressed device on the memory bus. **The processor completes one internal data transfer in one clock cycle.**

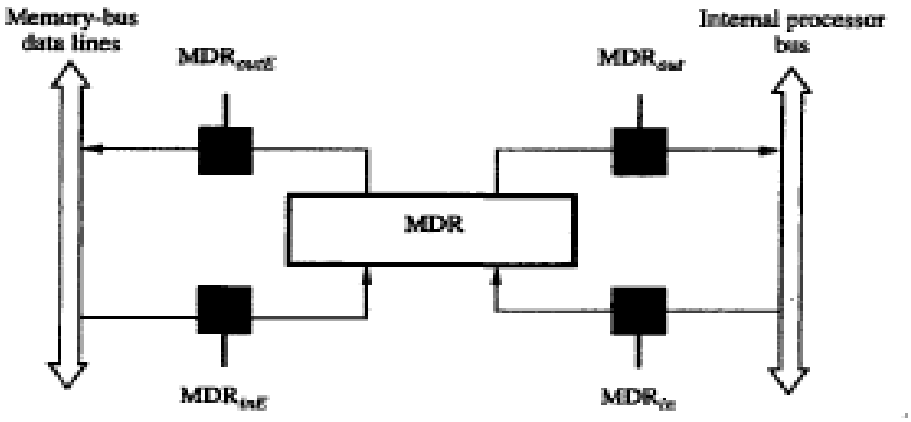

Figure: Connections and control signals for register MDR

To **accommodate the variability in response time**, the processor waits until it receives an **indication that the requested read operation has been completed**. The control signal **MFC (Memory Function Completed)** is used for this purpose. The addressed device sets this signal to 1 to indicate that the contents of the specified location have been read and are available on data lines of memory bus.

As an example of a read operation, consider the instruction **Move (R1), R2**. The actions needed to execute this instruction are:

1. MAR ← [R1]
2. Start a Read operation on the memory bus
3. Wait for the MFC response from the memory
4. Load MDR from the memory bus
5. R2 ← [MDR]

These actions may be carried out as separate steps, but some can be combined into single step. Each action can be completed in one clock cycle, except action 3 which requires one or more clock cycles, depending on the speed of the addressed device.

For simplicity, let us assume that the output of MAR is enabled all the times. Thus, **the contents of MAR are always available on the address lines of the memory bus**. This is the case **when the processor is bus master.** When a new address is loaded into MAR, it will appear on the memory bus at the beginning of the next clock cycle as shown in below figure.

A Read control signal is activated at the same time MAR is loaded. This signal will cause the bus interface circuit to send a read command, MR, on the bus. With this arrangement, we have combined actions 1 and 2 above into a single control step. Actions 3 and 4 can also be combined by activating control signal MDR_{inE} while waiting for a response from the memory. Thus the data received from the memory are loaded into MDR at the end of the clock cycle in which MFC is received. In the next clock cycle, MDR_{out} is activated to transfer the data to register R2. This means that the memory read operation requires three steps, which can be described by the signals being activated as follows:

1. $R1_{out}$, MAR_{in}, Read
2. MDR_{inE}, WMFC
3. MDR_{out}, $R2_{in}$

Where WMFC is the control signal that caused the processor's control circuitry to wait for the arrival of MFC signal.

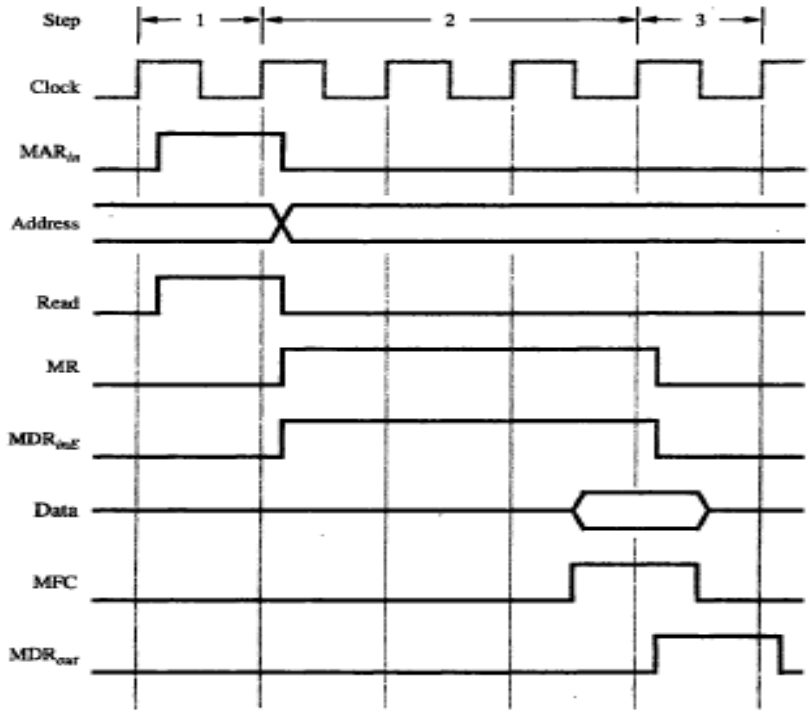

Figure: Timing of memory read operation

STORING A WORD IN MEMORY

For writing a word into a memory location, the desired address is loaded into MAR. Then, the data written are loaded into MDR, and a Write command is issued.

Consider the instruction Move R2, (R1). This requires the following sequence:

1. R1out, MAR_{in} ; desired address is loaded into MAR
2. R2out, MDR_{in}, Write; data to be written are loaded into MDR & Write command is issued
3. MDR_{outE}, WMFC; load data into memory location pointed by R1 from MDR

EXECUTION OF COMPLETE INSTRUCTION

Consider the instruction **Add (R3), R1** which adds the contents of a memory location pointed to by R3 to register R1. Executing this instruction requires the following actions:

1. Fetch the instruction.
2. Fetch the first operand (the contents of the memory location pointed to by R3).
3. Perform the addition.
4. Load the result into R1.

The sequence of control steps required for execution of this instruction is as follows:

1. **PC_{out}, MAR_{in}, Read, Select4, Add, Zin**
2. **Zout, PC_{in}, Yin, WMFC**
3. **MDR_{out}, IR_{in}**
4. **R3out, MAR_{in}, Read**
5. **R1out, Y_{in}, WMFC**
6. **MDR_{out}, SelectY, Add, Zin**
7. **Zout, R1in, End**

Instruction execution proceeds as follows:

Step1: The instruction fetch operation is initiated by loading the contents of PC into MAR and sending a Read request to memory. The Select signal is set to Select4, which causes the Mux to select constant 4. This value is added to operand at input B (PC's content), and the result is stored in Z.
Step2: Updated value in Z is moved back to PC, while waiting for the memory to respond.

Step3: Fetched instruction is moved into MDR and then to IR.

Step4: The contents of R3 are loaded into MAR & a memory read operation is initiated.

Step5: Contents of R1 are transferred to Y to prepare for addition.

Step6: When Read operation is completed, memory-operand is available in MDR, and the addition is performed.

Step7: The contents of MDR are gated to the bus, and register Y is selected as the second input to the ALU by choosing selectY. The sum is stored in Z, then transferred to R1.The End signal causes a new instruction fetch cycle to begin by returning to step1.

Steps 1 through 3 constitute the **instruction fetch phase**, which is the same for all instructions. The **instruction decoding** circuit interprets the contents of IR at the **beginning of step 4**. This enables the control circuitry to activate the control signals for steps 4 through 7, which constitute **the execution phase**.

Note: The above discussion accounts for all control signals, expect Yin in step2. There is no need to copy the updated contents of PC into register Y when executing arithmetic and logic instructions. But, in branch instructions the updated value of the PC is needed to compute the **Branch target address**. To speed up the execution of branch instructions, this value is copied into register Y in step2. Since step2 is part of fetch phase, the same action will be performed for all instructions.

Example: Write the complete control sequence for the instruction **Move (Rs), Rd**

Solution: The given instruction copies the contents of memory-location pointed to by Rs into Rd. This is a memory read operation. This requires the following actions

1. Fetch the instruction
2. Fetch the operand (i.e. the contents of the memory-location pointed by Rs).
3. Transfer the data to Rd.

The control-sequence is written as follows

1. PCout, MARin, Read, Select4, Add, Zin
2. Zout, PCin, Yin, WMFC
3. MDR_{out}, IRin
4. Rs, MAR_{in}, Read
5. MDR_{inE}, WMFC
6. MDR_{out}, Rd, End

BRANCH INSTRUCTIONS

A branch instruction replaces the contents of the PC with the branch target address. This address is usually obtained by adding an offset X, which is given in the branch instruction, to the updated value of PC.

The Control sequence that implements an unconditional branch instruction is as follows:

1. PCout, MAR_{in}, Read, Select4, Add, Zin
2. Zout, PCin, Yin, WMFC
3. MDRout, IRin
4. Offset-field-of-IR_{out}, Add, Zin
5. Zout, PCin, End

The processing starts, as usual, the **fetch phase ends in step3**. The offset value is extracted from the IR by the instruction decoding circuit. Since the updated value of PC is already available in register Y, the offset X is gated onto the bus, and an addition operation is performed in **step4**. In **step 5**, the result, which is the branch-address, is loaded into the PC.

The offset X used in a branch instruction is usually the difference between the branch target-address and the address immediately following the branch instruction.

For example, if the branch instruction is at location 2000 and if the branch target address is 2050, then the value of X must be 46. The reason for this is PC is incremented during the fetch phase, before knowing the type of instruction being executed (PC will be containing the address 2004 after fetching the instruction at location 2000). Thus, when the branch address is computed in step4, the PC value used is the updated value, which points to the instruction following the branch instruction in memory.

In case of conditional branch, we need to check the status of the condition-codes before loading a new value into the PC. **For example**, for a **Branch-on-negative (Branch < 0)** instruction, the above given **step 4** for unconditional branch instruction is replaced with

Offset-field-of-IRout, Add, Zin, If N=0 then End

Thus, if N=0, processor returns to step 1 immediately after step4. If N=1, step5 is performed to load a new value into PC, thus performing branch operation.

MULTIPLE BUS ORGANIZATION

In the single bus structure, the resulting control sequences are quite long because only one data item can be transferred over the bus in a clock cycle. To reduce the number of steps needed, most commercial processors provide multiple internal paths that enable several transfers to take place in parallel.

The below figure depicts a three bus structure used to connect the registers and the ALU of a processor. All general-purpose registers are combined into a single block called the **register file.** The register-file has **three ports**. There are **two outputs**, allowing the contents of two different registers to be accessed simultaneously and have their contents placed on the buses A and B. The third **input port** allows data on bus C to be loaded into a third register during the same clock-cycle.

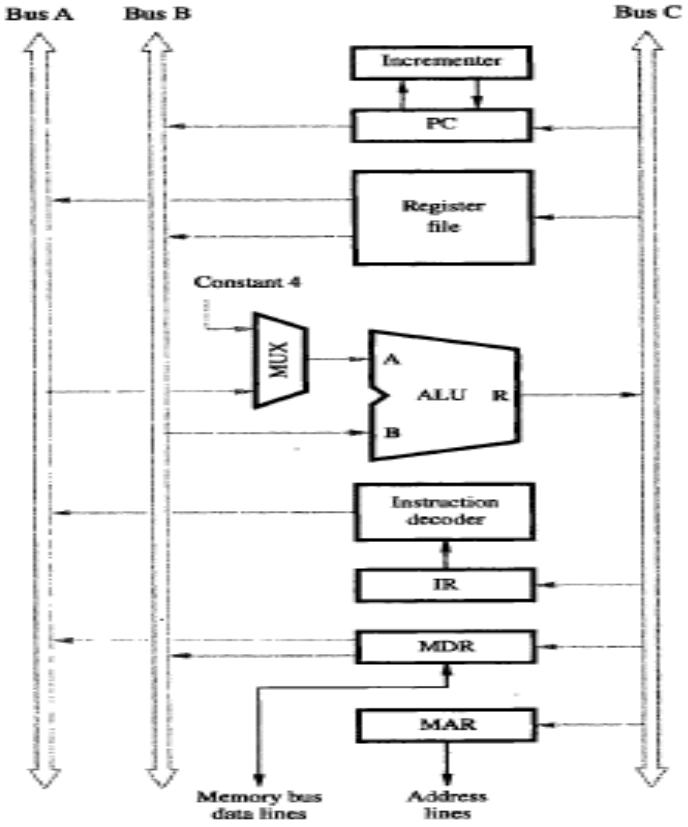

Figure: Three bus organization of the datapath

Buses A and B are used to transfer source operands to the A and B inputs of ALU, where an arithmetic or logic operation may be performed. The result is transferred to the destination over bus C. If needed, the ALU may simply pass one of its two input operands unmodified to bus C. We will call the ALU control signals for such an operation R=A or R=B. The three bus arrangement removes the need for registers Y and Z, which are in single bus structure.

The other feature in three bus organization is the **Incrementer unit**, which is used to increment PC by 4. Using this unit, eliminates the need to add 4 to the PC using main ALU. **The source for the constant 4 at the ALU input multiplexer is still useful**. It can be used to increment other addresses, such as memory addresses in LoadMultiple and StoreMultiple instructions.

Example: Consider the three operand instruction **Add R4, R5, R6**. The control sequence for the this instruction is as follows

1. PC_{out}, R=B, MAR_{in}, Read, IncPC
2. WMFC
3. MDR_{outB}, R=B, IR_{in}
4. $R4_{outA}$, $R5_{outB}$, SelectA, Add, $R6_{in}$, End

The instruction execution proceeds as follows:
Step1: The contents of PC are passed through ALU, using R=B control signal and loaded into MAR to start a memory Read operation. At the same time, PC is incremented by 4.
Step2: The processor waits for MFC signal from memory.
Step3: The processor loads requested data into MDR, and then transfers them to IR.
Step4: The instruction is decoded and add operation take place in a single step.
By providing more paths for data transfer a significant reduction in the number of clock cycles needed to execute an instruction is achieved.

HARDWIRED CONTROL

To execute instructions, **the processor must have some means of generating the control signals** needed in the proper sequence. Computer designers use a wide variety of techniques to generate control signals in the processor. The approaches used to fall into one of two categories:

1. **Hardwired control and**
2. **Microprogrammed control.**

Example: Consider the instruction **Add (R3), R1** the sequence of control steps required for execution of this instruction is as follows:

1. PC_{out}, MAR_{in}, Read, Select4, Add, Zin
2. Zout, PC_{in}, Yin, WMFC
3. MDR_{out}, IR_{in}
4. R3out, MAR_{in}, Read
5. R1out, Y_{in}, WMFC
6. MDR_{out}, SelectY, Add, Zin
7. Zout, R1in, End

Each step in this sequence is completed in one clock period. A **counter** may be used to keep **track of control steps**, as shown in the below figure. Each state, or count, of this counter corresponds to one control step. The required control signals are determined by the following information:

- Contents of the control step counter
- Contents of instruction register
- Contents of condition code flags
- External input signals, such as MFC and interrupt requests

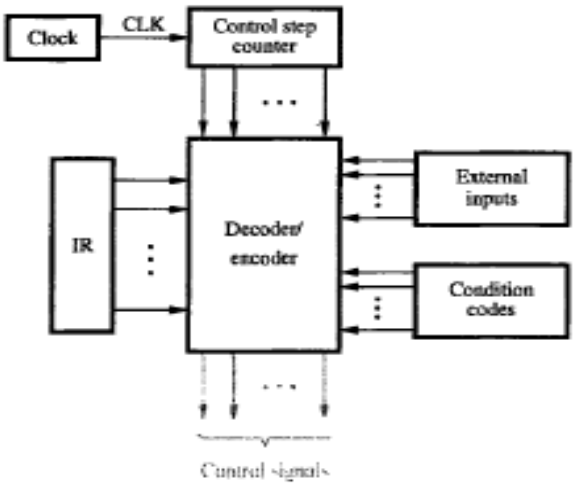

Figure: Control unit organization

The decoder/encoder block is a combinational-circuit that generates required control-outputs depending on state of all its inputs. By separating the decoding and encoding functions, we obtain more detailed block diagram as shown in below figure.

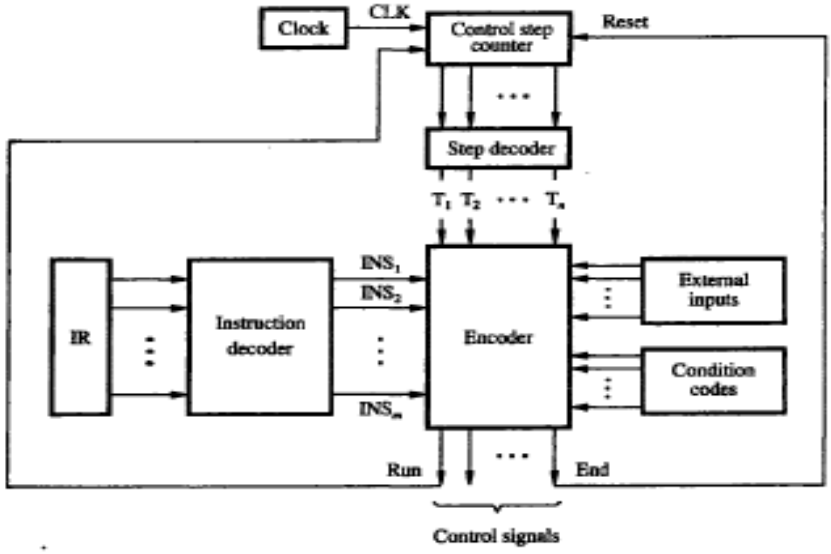

Figure: Separation of decoding and encoding functions

The **Step-decoder** provides a **separate signal line for each step or time slot, in the control sequence**. Similarly, the output of instruction-decoder consists of a separate line for each machine instruction. For any instruction loaded in IR, one of the output lines INS_1 through INS_m is set to 1, and all other lines set to 0. The **input signals to encoder-block** of the above figure are combined to generate the individual control signals Y_{in}, PC_{out}, Add, End and so on.

For example, $\mathbf{Z_{in}=T_1+T_6.ADD+T_4.BR}$; this signal is asserted during time slot $\mathbf{T_1}$ **for all instructions**, during $\mathbf{T_6}$ **for an Add** instruction, during $\mathbf{T_4}$ **for unconditional branch** instruction and so on. The logic function for $\mathbf{Z_{in}}$ is derived from the control sequences as shown below:

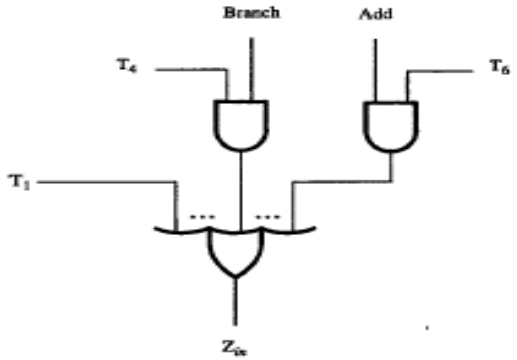

Figure: Generation of Z_{in} control signal for the processor

Another example, $\mathbf{End = T_7.\ ADD + T_5.BR + (T_5.N + T_4.\ N').\ BRN+ \ldots}$ gives a circuit that generate the End control signal from the above logic function. The end signal starts a new instruction fetch cycle by resetting the control step counter to its starting value. When RUN=1, counter is incremented by 1 at the end of every clock cycle. When RUN=0, counter stops counting. This is needed whenever the WMFC signal is issued, to cause the processor to wait for the reply from the memory.

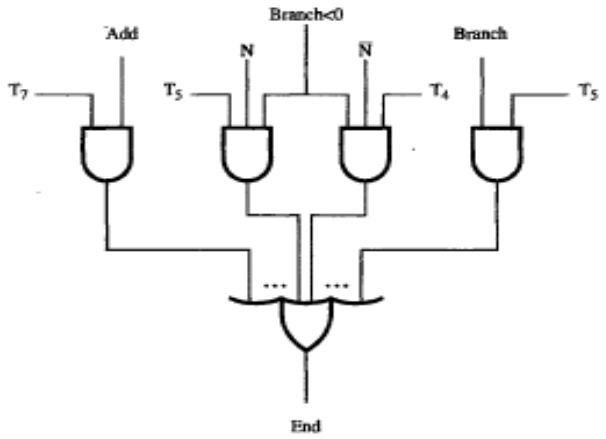

Figure: Generation of End control signal

The sequence of operations carried out by this machine is determined **by wiring of logic elements**, hence the name **"hardwired".** A controller that uses this approach can operate at **high speed**. However, it has **limited flexibility**, and the complexity of the instruction set it can implement is limited.

MICRO PROGRAMMED CONTROL

The control signals required inside the processor can be generated using a control step counter and decoder/encoder circuit in hardwire control. In **micro programmed control**, the **control signals are generated by a program similar to machine language programs**.

Terminology used in microprogram control:

Control word:

A control word (CW) is a word whose individual bits represent various control signals (like Add, End, Zin). Each of the control steps in control sequence of an instruction defines a unique combination of 1s and 0s in the CW.

Microinstruction:

The individual control words in this micro-routine are referred to as microinstructions.

Micro-routine:

A sequence of CWs corresponding to the control sequence of a machine instruction constitutes the micro-routine for that instruction.

Control store: The micro-routines for all instructions in the instruction set of a computer are stored in a special memory called the control store. The control unit can generate the control signals for any instruction by sequentially reading CWs of corresponding micro-routine from control store.

Microprogram counter (µPC):

It is used to read control words sequentially from the control store. Every time a new instruction is loaded into IR, output of block labeled "**starting address generator**" is loaded into µPC. The µPC is automatically incremented by clock, causing successive microinstructions to be read from control store. Hence, control signals are delivered to various parts of processor in correct sequence.

Organization of Microprogrammed Control Unit (To support Conditional Branching)

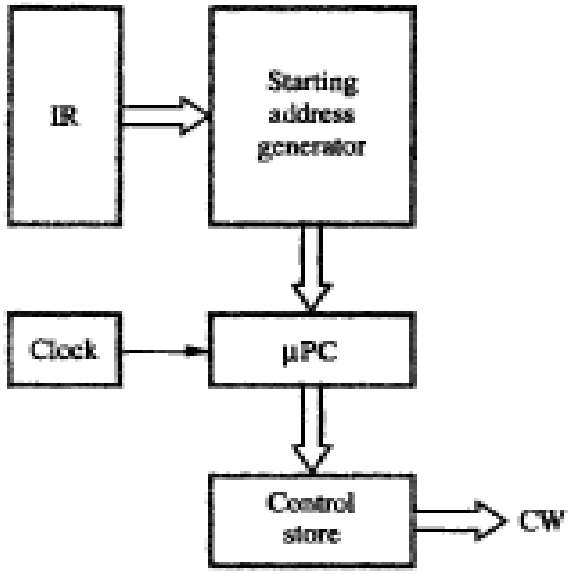

Figure: Basic organization of microprogrammed control

In the basic organization of microprogrammed control, the control unit can generate control signals to implement the basic arithmetic and logic functions. But when a situation arises when the control unit is required to check the status of the condition codes or external inputs to choose between alternative courses of action, this cannot be implemented on basic organization of microprogrammed control.

An alternative approach is to use conditional branch microinstructions. In addition to branch address, these microinstructions specify which of the external inputs, condition codes should be checked as a condition for branching to take place.

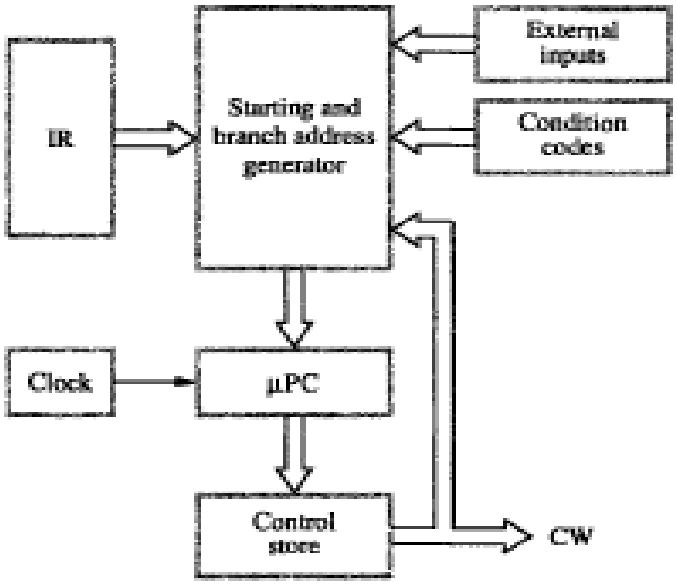

Figure: Organization of the control unit to allow conditional branching in microprogram

The instruction Branch < 0 may now be implemented by a micro-routine as shown below:

Address	Microinstruction
0	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
1	Z_{out}, PC_{in}, Y_{in}, WMFC
2	MDR_{out}, IR_{in}
3	Branch to starting address of appropriate microroutine
...	...
25	If N=0, then branch to microinstruction 0
26	Offset-field-of-IR_{out}, SelectY, Add, Z_{in}
27	Z_{out}, PC_{in}, End

Figure: Micro-routine for the instruction Branch < 0

After loading this instruction into IR, a branch microinstruction transfers control to the corresponding micro-routine, which is assumed to start at location 25 in the control store. This address is the output of the starting address generator block. The microinstruction at location 25 tests the N bit of the condition codes. If this bit is equal to 0, a branch takes place to location 0 to fetch the new machine instruction. Otherwise, the microinstruction at location 26 is executed to put the branch target address into register Z. The micro instruction in location 27 loads this address into PC.

The starting and branch address generator block loads a new address into μPC when a microinstruction instructs it to do so. To allow **implementation of a conditional branch**, inputs to this block consist of external inputs and condition codes as well as the contents of IR. In this control unit, the μPC is incremented every time a new microinstruction is fetched from microprogram memory, except in following situations

1. When a new instruction is loaded into IR, the μPC is loaded with starting-address of micro-routine for that instruction.

2. When a Branch microinstruction is encountered and branch condition is satisfied, μPC is loaded with branch address.

3. When an End microinstruction is encountered, μPC is loaded with address of first CW in micro-routine for instruction fetch cycle.

MICROINSTRUCTIONS

Micro-instruction	...	PC_{in}	PC_{out}	MAR_{in}	Read	MDR_{out}	IR_{in}	Y_{in}	Select	Add	Z_{in}	Z_{out}	$R1_{out}$	$R1_{in}$	$R3_{out}$	WMFC	End	...
1		0	1	1	1	0	0	0	1	1	1	0	0	0	0	0	0	
2		1	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	
3		0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	
4		0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	
5		0	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	
6		0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	
7		0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	

Figure: An example of microinstructions

Drawback of microprogrammed control:

Assigning individual bits to each control signal results in long microinstructions because the number of required signals is usually large. Moreover, only a few bits are set to 1 (to be used for active gating) in any given microinstruction, which means the available bit-space is poorly used.

Consider a simple processor and assume it contains only four general purpose registers R0, R1, R2 and R3**. Some of the connections in this processor are permanently enabled**, such as the output of IR to the decoding circuits and both inputs to ALU. The remaining connections to various registers require a total of 20 gating signals. Some additional control signals like Read, Write, Select, WMFC, and End signals are also needed. Let us assume ALU performs 16 different functions including Add, Subtract, AND and XOR. In total, 42 control signals are needed. The **length of the microinstructions can be reduced** easily.

Microinstruction

F1	F2	F3	F4	F5

F1 (4 bits)	F2 (3 bits)	F3 (3 bits)	F4 (4 bits)	F5 (2 bits)
0000: No transfer	000: No transfer	000: No transfer	0000: Add	00: No action
0001: PC_{out}	001: PC_{in}	001: MAR_{in}	0001: Sub	01: Read
0010: MDR_{out}	010: IR_{in}	010: MDR_{in}	⋮	10: Write
0011: Z_{out}	011: Z_{in}	011: $TEMP_{in}$		
0100: $R0_{out}$	100: $R0_{in}$	100: Y_{in}	1111: XOR	
0101: $R1_{out}$	101: $R1_{in}$		16 ALU functions	
0110: $R2_{out}$	110: $R2_{in}$			
0111: $R3_{out}$	111: $R3_{in}$			
1010: $TEMP_{out}$				
1011: $Offset_{out}$				

F6	F7	F8	...

F6 (1 bit)	F7 (1 bit)	F8 (1 bit)
0: SelectY	0: No action	0: Continue
1: Select4	1: WMFC	1: End

Figure: An example of partial format for field encoded microinstructions

If we use the simple encoding scheme, 42 bits would be needed in each micro instruction. The length of the microinstruction can be reduced as most of the signals not needed simultaneously, and many signals are mutually exclusive. This suggests that **signals can be**

grouped so that mutually exclusive signals are placed in same group. Thus, **at most one microoperation per group is specified in any microinstruction.** Then it is possible to use a binary coding scheme to represent the signals within the group. For example, four bits sufficient to represent 16 available functions in the ALU. Register output control signals can be placed in a group consisting of PC_{out}, MDR_{out}, Z_{out}, $Offset_{out}$, $R0_{out}$, $R1_{out}$, $R2_{out}$, $R3_{out}$ and $Temp_{out}$. Any of these can be selected by unique 4 bit code.

Grouping control signals into fields requires a little more hardware because decoding circuits must be used to decode bit patterns of each field into individual control signals. The cost of this additional hardware is more than offset by the reduced number of bits in each microinstruction, which results in a smaller control store (only 20 bits are needed to store the patterns for 42 signals).

Vertical organization	Horizontal organization
Highly encoded schemes that use compact codes to specify only a small number of control functions in each microinstruction are referred to as a vertical organization.	The minimally encoded scheme in which many resources can be controlled with a single microinstruction is called a horizontal organization.
This approach results in considerably slower operating speeds because more microinstructions are needed to perform the desired control functions.	This approach is useful when a higher operating speed is desired and when the machine structure allows parallel use of resources.

Although fewer bits are required for each microinstruction, this does not imply that the total number of bits in the control store is smaller. The significant factor is **that less hardware is needed to handle the execution of microinstructions.** Horizontal and vertical organizations represent the two organizational extremes in microprogrammed control. Many intermediate schemes are also possible in which degree of encoding is a design parameter.

Two major disadvantage of microprogrammed control is:

1) Having a **separate micro-routine for each machine instruction results in a large total number of microinstructions and a large control-store**. If most machine instructions involve several addressing modes, there can be many instruction and addressing mode combinations. A separate micro-routine for each of these combinations would produce considerable duplication of common parts. We want to organize the microprogram so that the micro-routines share as many common parts as possible. This requires many branch microinstructions to transfer control among various parts.

2) Execution time is longer because it takes more time to carry out the required branches.