CHAPTER 18.2
COMPUTER STORAGE

BASIC CONCEPTS

The main memory attached to a processor represents the most crucial resource of the computing system. In most instances, the storage system determines the bulk of any general-purpose computer architecture. Once the word size and instruction set have been specified, the rest of computer architecture and design deals mainly with optimization of the attached memory and storage hierarchy. Early computers were designed by first choosing the main memory hardware. This not only specified much of the remaining architecture, e.g., serial or parallel machine, but also dictated the *processor cycle time*, which was chosen equal to the *memory cycle time.* As computer technology evolved, the logic circuits and hence processor cycle time* improved dramatically. This improved speed demanded more main-memory capacity to keep the processor busy, but the need for increasingly larger capacity at higher speed placed a difficult and usually impossible demand on memory technology. In the early 1960s, a gap appeared between the main-memory and processor cycle times, and the gap grew with time. Fundamentally, it is desirable that main-memory cycle time be approximately equal to the processor cycle time, so this gap could not continually widen without serious consequences. In the late 1960s, this gap became intolerable and was bridged by the "cache" concept, introduced by IBM in the System/36 Model 85.

The cache concept proved to be so useful and important that by the late 1970s it became quite common in small, medium, and large machine architectures. The *cache* is a relatively small high-speed random-access memory that is paged out of main memory and holds the most recently and frequently used instructions and data. The same fundamental concepts that provide the basis for cache design apply equally well to "virtual memory" systems; only the methods of implementation are different.

In terms of implementation methods, there are five types of storage systems used in computers.

1. *Random-access memory* is one for which any location (word, bit, byte, record) of relatively small size has a unique, physically wired-in addressing mechanism and is retrieved in one memory-cycle time interval. The time to retrieve from any given location is made to be the same for all locations.
2. *Direct-access storage* is a system for which any location (word, record, and so on) is not physically wired in and addressing is accomplished by a combination of direct access to reach a general vicinity plus sequential searching, counting, or waiting to find the final location. *Access time* depends on the physical location of the record at any given time; thus access time can vary considerably, from record to record and to a given record when accessed at a different time. Since addressing is not wired in, the storage medium must contain a certain amount of information to assist in the location of the desired data. This is referred to as *stored addressing information.*
3. *Sequential-access storage* designates a system for which the stored words or records do not have a unique address and are stored and retrieved entirely sequentially. Stored addressing information in the form of

*Processor cycle time is roughly 10 logic-gate delays with appropriate circuit and package loads.

simple interrecord gaps is used to separate records and assist in retrieval. Access time varies with the record being accessed, as with direct access, but sequentially accessing may require a search of every record in the storage medium before the correct one is located.

4. *Associative (content-addressable) memory* is a random-access type of memory that in addition to having a conventional wired-in addressing mechanism also has wired-in logic that makes possible a comparison of desired bit locations for a specified match for all words simultaneously during one memory-cycle time. Thus, the specific address of a desired word need not be known since a portion of its contents can be used to access the word. All words that match the specified bit locations are flagged and can then be addressed on subsequent memory cycles.
5. *Read-only memory (ROM)* is a memory that has permanently stored information programmed during the manufacturing process and can only be read and never destroyed. There are several variations of ROM. *Postable* or *programmable* ROM (PROM) is one for which the stored information need not be written in during the manufacturing process but can be written at any time, even while the system is in use, i.e., it can be posted at any time. However, once written, the medium cannot be erased and rewritten. Another variation is a *fast-read, slow-write* memory for which writing is an order of magnitude slower than reading. In one such case, the writing is done much as in random-access memory but very slowly to permit use of low-cost devices. Several types of ROM chips are available. The erasable programmable ROM (EPROM) allows the use of ultraviolet light (UV) to erase its contents. This UV-EPROM can take up to 20 min to erase its contents; a ROM burner may then be used to implant new data. An electrically erasable programmable ROM (EEPROM) allows the use of electrical energy to clear its memory. The flash memory EPROM is a high-speed erasable EPROM.

The various computer-storage types are related to each other through access time. An approximate rule of thumb for access time comparisons is as follows (where T = access time):

$$T_c = 10^{-1}T_m = 10^{-6}T_d = 10^{-9}T_t$$

where T_c = cache time
T_m = main time
T_d = disc time
T_t = tape time

STORAGE-SYSTEM PARAMETERS

In any storage system the most important parameters are the capacity of a given module, the access time to any piece of stored information, the data rate at which the stored information can be read out (once found), the cycle time (how frequently the system can be accessed for new information), and the cost to implement all these functions.

Capacity is simply the maximum number of bits (b), bytes (B), or words that can be assembled in one basic self-contained operating module.

Access time can vary depending on the type of storage. For random-access memory the *access time* is the time from the instant a request appears in an address register until the desired information appears in an output register, where it can subsequently be further processed. For nonrandom-access storage, the access time is the time from the instant an instruction is decoded asking for information until the desired information is found but not read. Thus, access time is a different quantity for random and nonrandom-access storage. In fact, it is the access time that distinguishes the two, as is evident by the definitions above. Access time is made constant on random-access memory whereas on nonrandom storage access time may vary substantially, depending on the location of information being sought and the current position of the storage system relative to that information.

Data rate is the rate (usually bits per second, bytes per second, or words per second) at which data can be read out of a storage device. *Data transfer time* for reading or writing equals the product of the data rate and the quantity of the information being transferred. Data rate is usually associated with nonrandom-access storage where large pieces of information are stored and read serially. Since an entire word is then read out of random-access memory in parallel, data rate has no significance for such memories.

Cycle time is the rate at which a memory can be accessed, i.e., the number of accesses per unit time, and is applicable primarily to random-access storage.

FUNDAMENTAL SYSTEM REQUIREMENTS FOR STORAGE AND RETRIEVAL

In order to be able to store and subsequently find and retrieve information, a storage system must have the following four basic requirements:

Medium for storing energy

Energy source for writing the information, i.e., write transducers on word and bit lines

Energy sources and sensors to read, i.e., read and sense transducers

Information addressing capability, i.e., address-selection mechanism for reading and writing

The fourth requirement implicitly includes some coincidence mechanisms within the system to bring the necessary energy to the proper position on the medium for writing and a coincidence mechanism for associating the sensed information with the proper location during reading. In random-access memory, it is provided by the coincidence of electric pulses within the storage cell, whereas in nonrandom-access storage, it is commonly provided by the coincidence of an electric signal with mechanical position. In many cases, the write energy source serves as the read energy source as well, thus leaving only sense transducers for the third requirement. Nevertheless, a read energy source is still a basic requirement.

The differences between storage systems lie only in how these four requirements are implemented and, more specifically, in the number of transducers required to achieve these necessary functions. Here a *transducer* denotes any device (such as magnetic head, laser, transistor circuits) that generates the necessary energies for reading and writing, senses restored energy and generates a sense signal, or provides the decoding for address selection.

Since random-access memory is so crucial to processor architecture, we discuss this type first and continue in the order defined above.

The organization of random-access memory systems is intimately dependent on the number of functional terminals inherent in the memory cells. The cell and overall array organization are so interwoven that a discussion of one is difficult without the other. We discuss first the fundamental building block, the memory cell, and subsequently build the array structure from this.

RANDOM-ACCESS MEMORY CELLS

To be useful in random-access memory, the cell must have at least two independent functional terminals consisting of a word line and a common bit/sense line, as shown in Fig 18.2.1*a*. For writing into the cell, the coincidence of a pulse on the word-select line with the desired data on the bit-select line places the cell into one of two stable binary states. For reading, only a pulse on the word-select line is applied, and a sense signal indicating the binary state of the cell is obtained on the sense line. A more versatile but more complex cell is one having three functional terminals, as in Fig. 18.2.1*b*. A coincidence of pulses on both the *x* and *y* lines is necessary to select the cell for both reading and writing. The appearance of data on the bit/sense line puts the cell in the proper binary state. If no data are applied, a sense signal indicating the binary state of the cell is obtained. The use of coincidence for reading as well as writing allows this cell to be used in complex array organizations.

Magnetic ferrite cores, which played a key role in the early history of computers, were first introduced as three-terminal cells. In fact, the first ferrite core cells used four wires through each core (separate bit and sense lines) to achieve a three-functional terminal cell as in Fig. 18.2.1*b*. Refinements gradually allowed the bit and sense line to be physically combined to give a three-wire cell and eventually even a two-wire two-terminal cell.

The introduction of *integrated-circuit cells* in the early 1970s for random-access memory led to the disuse of ferrite cores. There are two general types of integrated-circuit cells, static and dynamic. *Static cells* remain in the state they are set until they are reset or the power is removed and are read nondestructively; i.e., the information is not lost when read. A *dynamic cell* gradually loses its stored information and must be refreshed periodically. In addition, the cell state is destroyed when read, and the data must be rewritten after each read cycle. However, dynamic cells can be made much smaller than static cells, which gives a greater density of bits per semiconductor chip. The resulting lower cost more than compensates for the additional complexity.

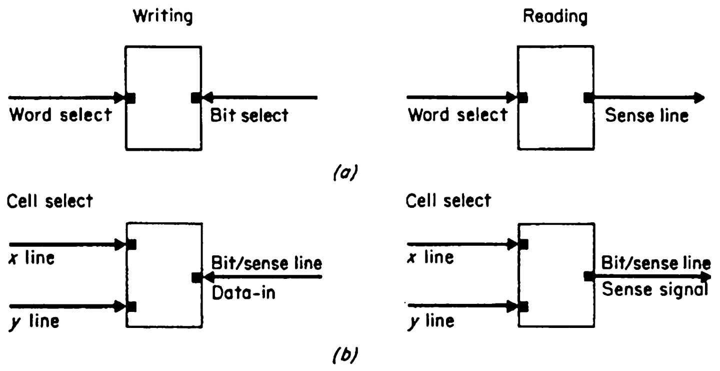


FIGURE 18.2.1 Reading and writing operations for random-access memory cells having two and three functional terminals: (*a*) cell with two functional terminals, and (*b*) cell with three functional terminals.

STATIC CELLS

All static cells use the basic principles of the Eccles-Jordan flip-flop circuit to store binary information in a cross-coupled transistor circuit like that shown in Fig. 18.2.2. Either *junction (bipolar)* or *field-effect transistors (FET)* can be used in the same configuration; only the voltages, currents, and load resistors will be different. To store a 0, transistor T_0 is turned on and T_1 turned off. A stored 1 is just the opposite condition, T_0 off and T_1 on. To achieve these states, it is necessary to control the node voltages at *A* and *B*. For instance, if node *A* voltage V_A is made sufficiently low, the base of T_1 will be low enough to turn T_1 off, causing node *B* to rise in voltage. This turns the base and hence the collector of T_0 on and holds node *A* at a low voltage so that a 0 is stored in the flip-flop. If the voltage at node *B* is made sufficiently low, the opposite state occurs, giving a stored 1. For reading, it is only necessary to sample the voltages at nodes *A* or *B* to see which is high or low. This gives a nondestructive read cell since the state is not changed, only sampled. Hence, nodes *A* and *B* are the access ports to the cell for both reading and writing.

The basic difference between all *static* cells is in how nodes *A* and *B* are accessed. Note that although the flip-flop of Fig. 18.2.2 has two access nodes *A* and *B*, it is functionally only a one-terminal cell since nodes

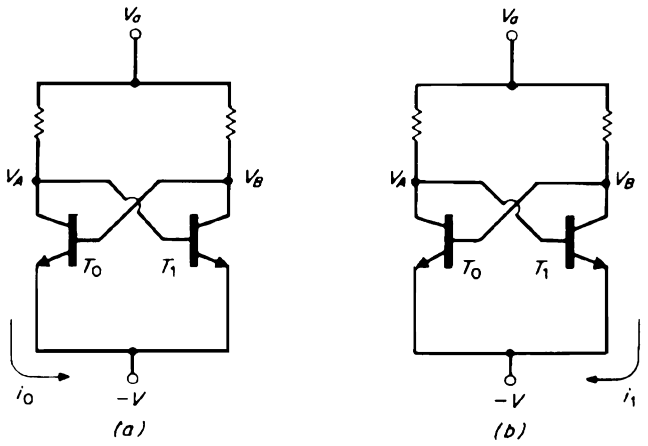


FIGURE 18.2.2 Transistor flip-flop circuits: (*a*) stored 0, T_0 conducts, $V_A = 0$, $V_B = 1$ V; (*b*) stored 1, T_1 conducts, $V_A = 1$ V, $V_B = 0$.

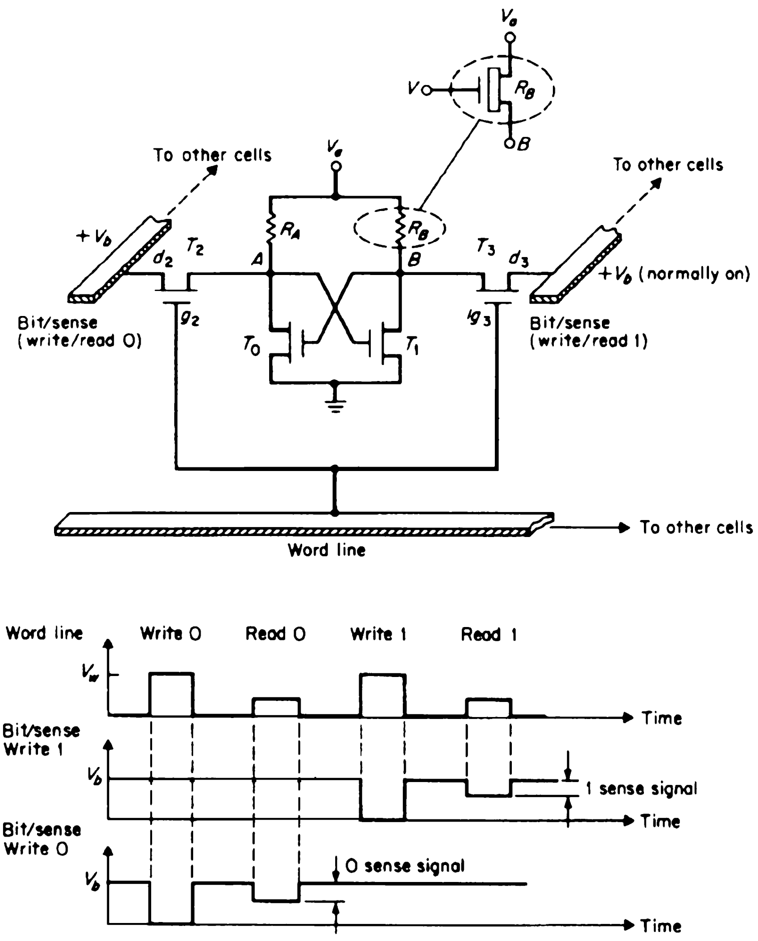


FIGURE 18.2.3 MOSFET two-terminal storage cell.

A and *B* are not independent but operate in a cooperative mode. To make a cell suitable for a random-access array, at least another functional terminal must be added. This can be achieved by the addition of another FET to each node, *A* and *B*, as shown in Fig. 18.2.3. Although the two transistors provide a total of four physical connections to the cell, that is, T_2 provides one gate g_2 and one drain d_2 for node *A* and T_3 provides g_3 and d_3 for node *B*, the circuit operates in a symmetrical, balanced mode. Since these four terminals are not independent, only two functional terminals are present. The operation of the cell can be understood by tracing through the pulsing sequence of Fig. 18.2.3. The peripheral circuits used to obtain these pulse sequences are not shown. An equivalent two-terminal static cell can be achieved by using junction transistors with some charges, but the operating principles remain basically the same.

A two-terminal static cell that was very popular in early integrated-circuit memories used multiemitter transistors. The cell required high power, however, a condition that greatly limits chip density.

Higher density and lower power can be obtained with the Schottky diode flip-flop cell of Fig. 18.2.4, a very popular cell for high-speed, low-power junction transistor memories. Resistors R_c are part of the internal collector contact resistance of the devices and are much smaller than R_L. The pulses on the word and bit/sense lines are used to forward- or back-bias the diodes D_0 and D_1, thus controlling or sensing the voltages and nodes *A* and *B*, as can be seen by tracing through the pulse sequence shown.

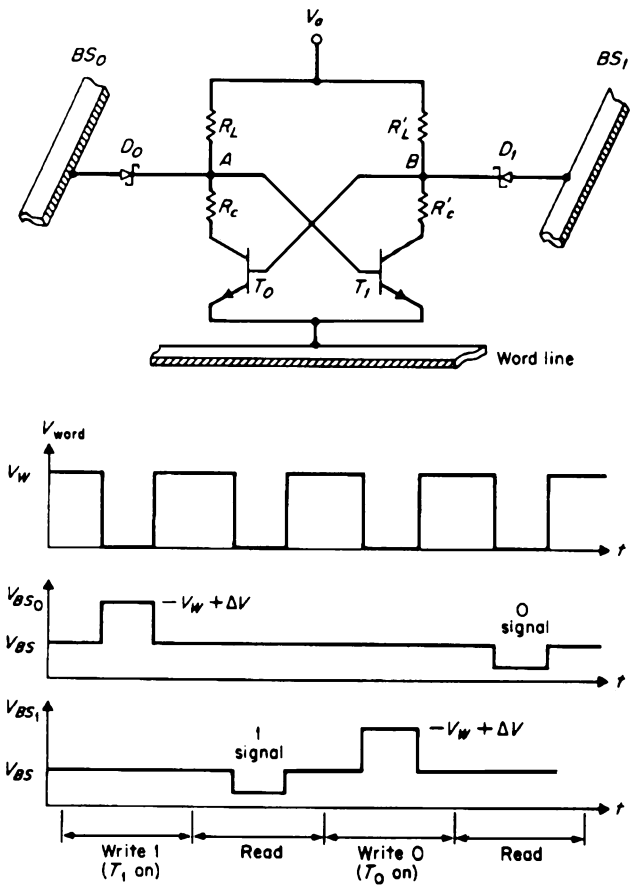


FIGURE 18.2.4 Schottky diode storage cell.

DYNAMIC CELLS

Since the static cells described above operate in a balanced, differential mode, it would seem reasonable that a nondifferential mode would be able to achieve at least a twofold reduction in component count per cell. This is indeed the case, and by sacrificing other properties of static cells, such as their inherent nondestructive read capability, a very significant reduction in cell size is possible. The most widely used such cell is the one-FET dynamic cell shown in a nonintegrated form in Fig. 18.2.5. The essential principle is simply the storing of charge on capacitor C_s for a 1 and no charge for a 0. A single capacitor by itself is sufficient to accomplish this, but an array of such devices requires a means of selecting the desired capacitors for reading and writing and of isolating nonselected capacitors from the array lines. If isolation is not provided, the charge stored on half-selected capacitors may inadvertently be removed, making the scheme inoperable. The isolation and selection means is provided by the simple one-FET device in series with the capacitor, as shown. For operation in an array, the terminal c can be either at ground or + V_c, depending on the technology and details of the cell. Assume that c is at + V_c, as indicated. To write either state, the word line is pulsed high to turn the FET on. If the bit/sense line is at ground, V_c will charge C_s to a stored 1 state. However, if the bit/sense line is at + V_c, any charge on C_s will be removed or no charge is stored if none was there originally, giving a stored 0. For reading, the word line is pulsed high to turn the FET on with the sense line at a normally high voltage. If there was

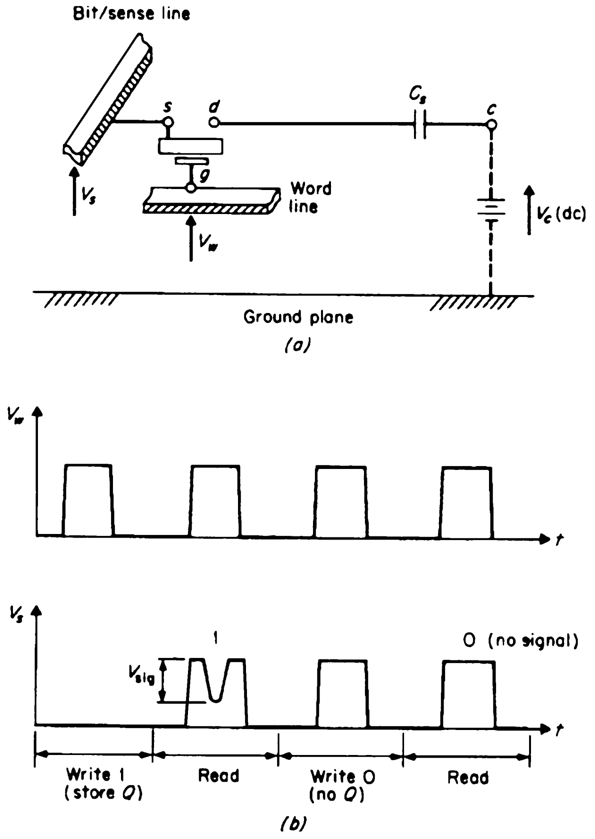


FIGURE 18.2.5 One-FET-device dynamic storage cell: (*a*) general equivalent circuit; (*b*) pulsing sequence.

charge on the capacitor, it will discharge through the bit/sense line to give a signal as shown. If there was no stored charge, no signal would be obtained.

Note that the reading is destructive since a stored 1 is discharged to a 0 and requires regeneration after each read operation in an array. Note also that the FET must carry current in both directions; i.e., the current charging C_s during writing is in the opposite direction of the current during reading. This cell has one further disadvantage: in an integrated structure, the charge on C_s will unavoidably leak off in a time typically measured in milliseconds. Hence the cell requires periodic refreshing, which necessitates additional peripheral circuits and complications. This feature gives rise to the term *dynamic cell.* Despite these disadvantages, this technique allows a very substantial improvement in cell density and cost, at very adequate cycle times and has become very popular.

RANDOM-ACCESS MEMORY ORGANIZATION

A memory is organized into bytes or byte groups known as words. If *N* is a word of fixed length, each word is assigned an address, or location, in memory. Each word has the same number of bits, called the word length. Each access usually retrieves the entire word. The addresses in memory run consecutively, from address 0 to the largest address.

In any memory system, if E units are to be selected, an address length of N bits is required which satisfies

$$2^N = E$$

In most cases of interest E is equal to W, the number of logical words in the system.

DIGITAL MAGNETIC RECORDING

Magnetic recording is attractive for data processing since it is inexpensive, easily transportable, unaffected by normal environments, and can be reused many times with no processing or developing steps.

The essential parts of a simplified but complete magnetic recording system are shown in Fig. 18.2.6. They consist of a *controller* to perform all the logic functions as well as write-current generation and signal detection; serial-to-parallel conversion registers; a read-write head with an air gap to provide a magnetic field for writing and sensing the flux during reading; and finally the medium. The wired-in cells, array, and transducers of random-access memory have been replaced by one read-write transducer, which is shared by all stored bits, and a shared controller. Coincident selection is still required for reading and writing, and this is obtained by the coincidence of electric signals in the read-write head with physical positioning of the medium under the head.

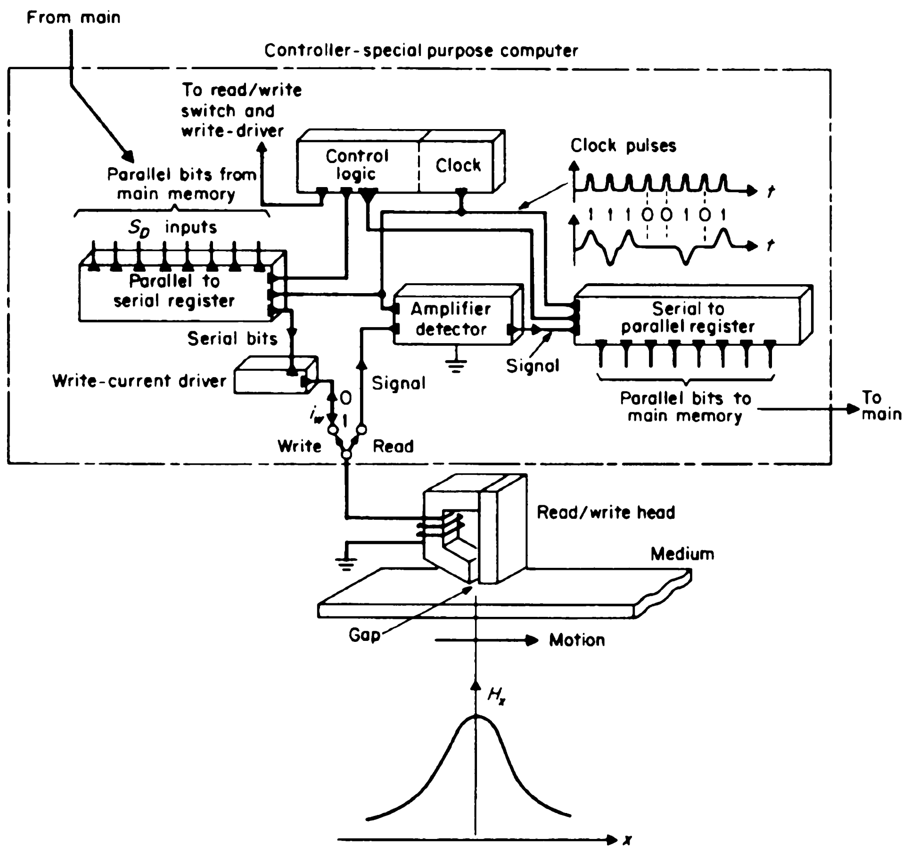


FIGURE 18.2.6 Schematic of simplified complete magnetic recording system.

The recording medium is very similar in principle to ferrite material used in cores. The common material for digital magnetic recording is ferric oxide, Fe_2O_3. It has remained essentially unchanged for many years except for reductions in the particle size, smoother surfaces, and thinner, more uniform coating, all necessary for high density. This material remained the sole medium for discs and tapes until the late 1970s when NiCo (nickel cobalt) was introduced. These new materials have a higher coercive force and can be deposited thinner and smoother, allowing higher recording densities. Operation of these media requires a reasonably rectangular magnetic hysteresis loop with two stable residual states $+B_r$ and $-B_r$ for storing binary information. The media must be capable of being switched an infinite number of times by a magnetic field produced by the write head, which exceeds the coercive force. Stored information is sensed by moving the magnetized bits at constant velocity under the read head to provide time-changing flux and hence an induced sense signal.

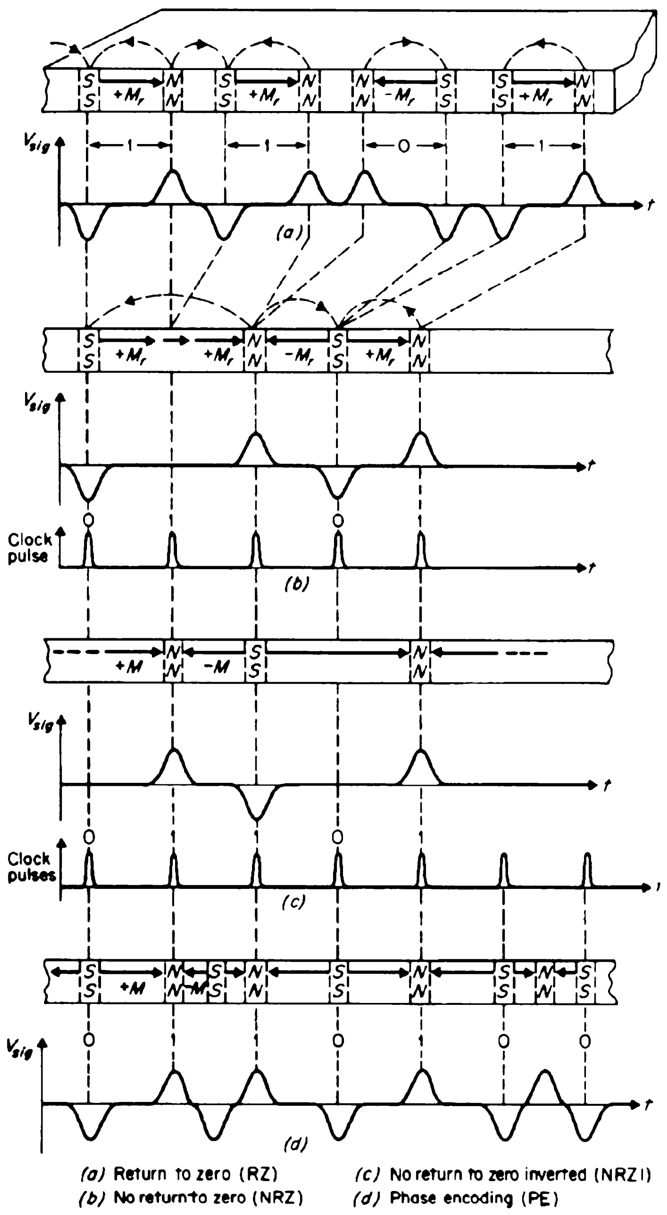
The essence of magnetic recording consists of being able to write very small binary bits, to place these bits as close together as possible, to obtain an unambiguous read-back voltage from these bits, and to convert this continuously varying voltage into discrete binary signals. The writing is done by the trailing edge of the write field.

The minimum size of one stored bit is determined by the minimum transition length required within the medium to change from $+B_r$ to $-B_r$ without self-demagnetizing. The smaller the transition length, the larger the self-demagnetizing field. The minimum spacing at which adjacent bits can now be placed with respect to a given bit is governed mainly by the distortion of the sense signal when adjacent bits are too close, referred to as *bit crowding*. This results from the overlapping of the fringe field from adjacent bits when they are too close and this total, overlapped magnetic field is picked up in the read head as a different induced signal from that produced by a single transition. Conversion of the analog read-back signal to digital form requires accurate clocking; this means that clocking information must be built into the coded information, particularly at higher densities.

Neglecting clocking and analog-to-digital conversion problems for the moment, the signals obtained during a read cycle are just a continuous series of 1s and 0s. A precise means of identifying the exact beginning and end of the desired string of data is necessary, and furthermore, some means for identifying specific parts within the data string is often desirable. Since the only way to recognize particular pieces of stored information is through the sequence of pulse patterns, special sequences of patterns such as gaps, address markers, and numerous other coded patterns are inserted into the data. These can be recognized by the logic hardware built into the controller, which is a special-purpose computer attached to the storage unit. These special recorded patterns along with other types of aids are referred to as the *stored addressing information* and constitute at least a part of the addressing mechanism.

Coding schemes are chosen primarily to increase the linear bit density. The particular coding scheme used determines the frequency content of the write currents and read-back signals. Different codes place different requirements on the mode of operation and frequency response of various parts of the system such as clocking technique, timing accuracy, head time constant, medium response, and others. Each of these can influence the recording density in different ways, but in the overall design the trade-offs are made in the direction of higher density. Thus special coding schemes are not fundamentally necessary but only meet practical needs. For instance, it is possible to store bits by magnetizing the medium over a given region where say $+M_r$ (magnetization) is a stored 1 and $-M_r$ is a stored 0, as in Fig. 18.2.7*a*. The transition region in between 1s and 0s is assumed to have $M = 0$ except for the small regions of north and south poles on the edges as shown. As the medium is moved past the read head, a signal proportional to dM/dt or dM/dx is induced, so that the north poles induce, say, a positive signal and south poles a negative signal as shown (polarity arbitrary).

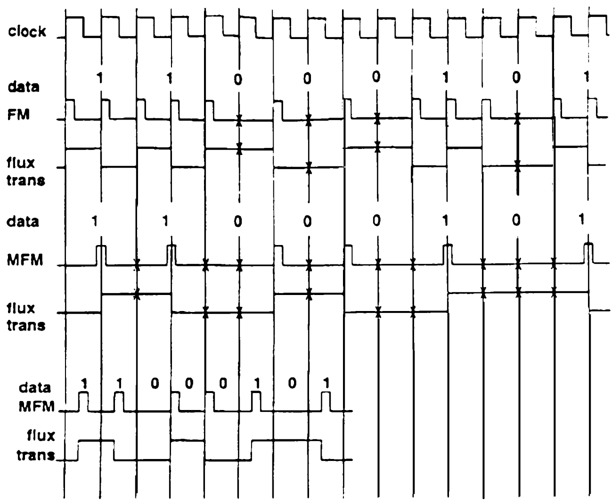
This code is known as *return to zero* (RZ) since the magnetization returns to zero after each bit. Each bit has one north- and one south-pole region, so that two pulses per bit result. Not only are two pulses per bit redundant, but considerable space is wasted on the medium for regions separating stored bits. It is possible to push these bits closer together, as in Fig. 18.2.7*b*, so that the magnetization does not return to 0 when two successive bits are identical, as shown for the first two 1s; hence the name *nonreturn to zero* (NRZ). The result is then only one transition region and one signal pulse per bit. By adjusting the clocking pulses to coincide with the signal peaks, we have a coding scheme in which 0s are always negative signals and 1s are always positive (Fig. 18.2.7*b*). The difficulty is that only a change from 1 to 0 or 0 to 1 produces a pulse; a string of only 1s or 0s produces no signals. This requires considerable logic and accurate clocking in the controller to avoid accumulated errors as well as to separate 1s from 0s.



(a) Return to zero (RZ)
(b) No return to zero (NRZ)
(c) No return to zero inverted (NRZI)
(d) Phase encoding (PE)

FIGURE 18.2.7 Cross-sectional view of magnetic recording medium showing stored bits, signal patterns, and clocking for various codes.

One popular coding scheme is a slightly revised version of the above, namely, *nonreturn to zero inverted* (NRZI), in which all 1s are recorded as a transition (signal pulse) and all 0s as no transition, as shown in Fig. 18.2.7*c*. There is no ambiguity between 1s and 0s, but again a string of 0s produces no pulses. A *double-clock* scheme, with two clocks, each triggered by the peaks of alternate signals, is used to set the clock timing period to the following strobe point. NRZI is a common coding scheme for magnetic tapes used at medium density.



(e) Time diagram for encoding FM, MFM, and double density MFM (X indicates no flux transition)

FIGURE 18.2.7 (*Continued*)

For greater than density 1600 b/in. the clocking and sensing become critical, and *phase encoding* (PE), shown in Fig. 18.2.7*d*, is often used. Since 1s give a positive signal and 0s give a negative signal, a signal is available for every bit. Phase encoding requires additional transitions within the medium, e.g., between successive 1s or successive 0s, as shown. Density, however, is usually limited by sensing, clocking, and other problems rather than by the medium capability.

For magnetic-disc recording, a double-frequency NRZI code is often used. This is obtained from NRZI by adding an additional transition just before each stored bit. The additional transition generates an additional pulse to serve as a clocking pulse. Hence a well-specified window is provided between bits to avoid clocking problems when a string of 0s is encountered. Other popular and useful codes are frequency modulation (FM), modified frequency-modulation (MFM), which is derived from the DF code, and the run-length-limited code. Since the latter does not maintain a distinction between data and clock transitions, a special algorithm is required to retrieve the data.

FM encodes 1 and 0 in different frequencies, but entails a minimum of one pulse per digit for both. A pulse always appears at the beginning of a clock cycle. If the data are 0s, there are no further pulses; if the data are 1s, there is an additional pulse. MFM eliminates the automatic pulse for each digit, and so is more efficient. MFM retains the pulse for 1; for encoding 0, there is a pulse at the beginning of a period unless it was preceded by a 0. The encoding is illustrated in Fig. 18.2.7*e*. RLL is even more efficient. The run length is the number of no-flux transitions between two consecutive transitions. Figure 18.2.7*e* shows the encoding for 11000101. FM has 12 flux transitions, while MFM requires only six transitions. In personal computers, MFM is used to achieve *double density* recording, whereas FM is used for single density.

Writing the transition (north or south) in the medium is done by the trailing edge of the fringe field produced by the write head. Since writing is rather straightforward and is not a limiting factor, we dwell here on the more difficult problems of reading and clocking.

Read-back signals can best be understood in terms of the reciprocity theorem of mutual inductance. This theorem states that for any two coils in a linear medium, the mutual inductance from coil 1 to 2 is the same as that from coil 2 to 1. When applied to magnetic recording, the net result is that the signal, as a function of time observed across the read-head winding induced by a step-function magnetization transition, has a shape that is identical to the curve H_x versus x for the same position of the medium below the head gap (Fig. 18.2.6). It is necessary only to replace x by vt, where v = velocity, for the translation of the x scale on H_x to the time scale on V_{sig} versus t. The H_x-versus-x curve with a multiplication factor is often referred to as the *sensitivity function.*

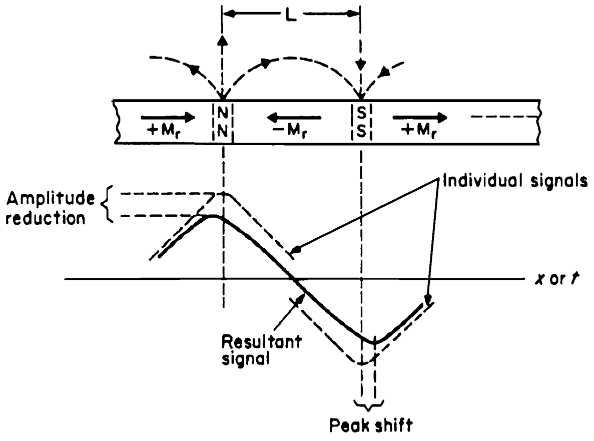


FIGURE 18.2.8 Bit crowding on read-back showing amplitude reduction and peak shift.

The writing of a transition is done by only one small portion of the H_x-versus-x curve, whereas the sense signal is determined by the entire shape of H_x versus x; that is, the signal is spread out.

This fact gives rise to *bit crowding,* which makes the read-back process more detrimental in limiting density than the writing process. To understand bit crowding, suppose there are two step-function transitions of north and south poles separated by some distance L, as in Fig. 18.2.8. When these transitions are far apart, their individual sense signals shown by the dashed lines appear at the read winding. However, as L becomes small, the signals begin to overlap and, in fact, subtract from each other,* giving both a reduction in peak amplitude and a time shift in the peak position as shown. This represents, to a large extent, the actual situation in practice. The transitions can be written closer together than they can be read.

Clocking or strobing of the serial data as they come from the head to convert them into digital characters is another fundamental problem. If perfect clock circuits with no drift and hence no accumulated error could be made, the clocking problem would disappear. But all circuits have tolerances, and as the bit density increases, the time between bits becomes comparable to the drift in clock cycle times. Since the drift can be different during reading and writing, serious detection errors can result. For high density, it is necessary to have some clocking information contained within the stored patterns as in the PE and double-frequency NRZI codes discussed previously.

MAGNETIC TAPE

The most common use of magnetic tape systems is providing backup for hard disk drives.

Since there are either seven or nine tracks written across the width of the tape, either seven or nine read-write heads are required to store one complete character or byte at a time. The bit spacing along a track is approximately the reciprocal of the linear density, or 0.00125 for an 800 b/in. system. The actual transition lengths are generally about half this bit-cell spacing. In many systems there are separate read and write gaps in tandem to check the reliability of the recording by reading immediately after writing. The tape is mechanically moved back and forth in contact with the heads, all under the direction of a controller. Tape transports in digital systems must be able to start and stop quickly, and achieve high tape speed rapidly. The streaming tape drive used in PCs does not start and stop quickly; it is used to backup disk systems.

The stored addressing information in tapes is relatively simple, consisting of specially coded bits and tape marks in addition to *interrecord gaps* (IRG). The latter are black space on the tape to provide space to accelerate

*Linear superposition is possible since the air gap makes the read head linear.

and decelerate between records since reading and writing can only be done at constant velocity. The common gap sizes are 0.6 and 0.75 in. Typically, a tape recorded at 800 b/in. with eight tracks plus parity storing records of 1K B each and a gap of 0.6 in between each record can hold over 10^8 b of data. Even though this represents a large capacity, the gap spaces consume nearly 50 percent of the tape surface, a rather extravagant amount. In order to increase efficiency, records are often combined into groups known as blocks. Since the system can stop and start only at an interrecord gap, the entire block is read into main memory for further processing during one read operation. The highest-density tapes can hold nearly eight times the above amount. Half-inch reel tapes are beginning to be replaced by shorter reels mounted in self-contained cartridges. One tape drive can hold multiple cartridges, which can be mechanically selected and mounted. Typical cartridge tapes are 165 in. long and record 18 tracks on $^1/_2$-in. tape at about 25K flux transitions per inch (972 flux transitions/mm).

DIRECT-ACCESS STORAGE SYSTEMS—DISCS

The major type of direct-access storage system is a disc. The system's recording head usually consists of one gap, which is used for both reading and writing. The head is "flown" on an air cushion above the disc surface at separations in the neighborhood of 5 to 100 μin., depending on the system. A well-controlled separation is vital to reliable recording.

For discs, the medium consists of very thin coatings of about 10 μin. or less of the same magnetic material used on tapes but applied to polished aluminum discs. Several discs are usually mounted on one shaft, all rotated in unison. Each surface is serviced by one head. The arms and heads are moved mechanically along a radial line, and each fixed position sweeps out a track on each surface, the entire group of heads sweeping out a cylinder. A typical bit cell is a rectangle 0.005 in. wide by 0.0005 in. long for a 2000 b/in. linear density. The transition length is about half this size.

The fundamental difference between various disc systems centers on the stored addressing information and addressing mechanisms built into the system. Some manufacturers provide a rather complex track format permitting the files to be organized in many different ways, using keys, identifying numbers, stored data, or other techniques for finding a particular word. Thus, the user can "program" the tracks and records to retrieve a particular word "on the fly." This provides a very versatile system but only with additional cost, since considerable function must be built into the controller. Other systems use a very simple track format consisting mainly of gaps and sector marks that do not permit the user to include programmable information about the data. This scheme is more suitable for well-organized data, such as scientific data; it still can be used in other applications with more user involvement.

Floppy discs are used in wide application as inexpensive high-density, medium-speed peripherals. Floppy discs often consist of the same flexible medium (hence the name floppy) as magnetic tape but cut in the form of a disc. Such discs straighten out when spun. The read-write mechanism is usually identical to that above except that the head is in contact with the medium, as in tape. This causes wear that is more significant than in ordinary discs, required frequent replacement of the disc and occasional replacement of the head, particularly when heavily used.

Floppy discs use tracks in concentric circles with movable heads and track-following servo. The systems record on both sides. The major deviations from flying-head discs are much smaller track density and data rate. The linear bit densities are quite comparable to those of tape. Typical parameters are 77 to 150 tracks per disc surface, 1600 to 6800 b/in., rotation from 90 to 3600 r/min.

Optical discs provide more storage than magnetic discs and are commonly in use. One optical disc can contain the same information as 20 or more floppy discs.

VIRTUAL-MEMORY SYSTEMS

Virtual memory is a term usually applied to the concept of paying a gain memory out of a disc. This concept makes the normal-sized main memory appear to the user as large as the virtual-address space* while still appearing to run at essentially the speed of the actual memory. Virtual memories are particularly useful

*Represented by a register holding the virtual address.

in multiprogrammed systems. On the average, virtual memory provides for better management of the memory resource with little wasted space because of fragmentation, which can otherwise be quite severe.

To understand *fragmentation,* suppose that a number of programs to be processed require small, medium, and large amounts of memory on a multiprogrammed machine. Suppose further that a small and a large program are both resident in main memory and that the small one has been completed. The operating system attempts to swap into memory a medium-sized program, to be processed, but it cannot fit in the space freed by the small program. If none of the other waiting programs is small enough to fit, this memory space is wasted until the large program has been completed. It can be seen that with many different-sized programs, fitting several of them into available memory space becomes difficult and leads to much unusable memory at any one time, i.e., fragmentation. Virtual memory avoids this by breaking all programs into *pages* of equal size, and dynamically paging them into main memory on demand. The identical concepts used in a virtual memory are applicable to a cache speed paged out of main memory; only the details of implementation are different, as shown later. We discuss first the basic concepts as applied to a main memory paged out of a disc and indicate, whenever possible, the differences applicable to a cache.

All virtual memories start with a *virtual address* that is larger than the address of the available main memory. Such being the case, the desired information may not be resident in the memory. Hence it is necessary to find out *if*, in fact, it is present. If the information is resident, it is necessary to determine *where* it is residing because the physical address cannot bear a one-to-one correspondence to the virtual address. In fact, in the most general case, there is no relationship between the two, so that an address translation scheme is necessary to find *where* the information does reside.

If the requested page is not resident in memory, a page *fault* results. This requires a separate process to find the page on the disc, remove some page from memory, and bring the new page into this open spot, called a *page frame.* Which page to remove from main storage is determined by a page-replacement algorithm that replaces some page "not recently used." The address-translation and page-replacement functions are shown conceptually in Fig. 18.2.9.

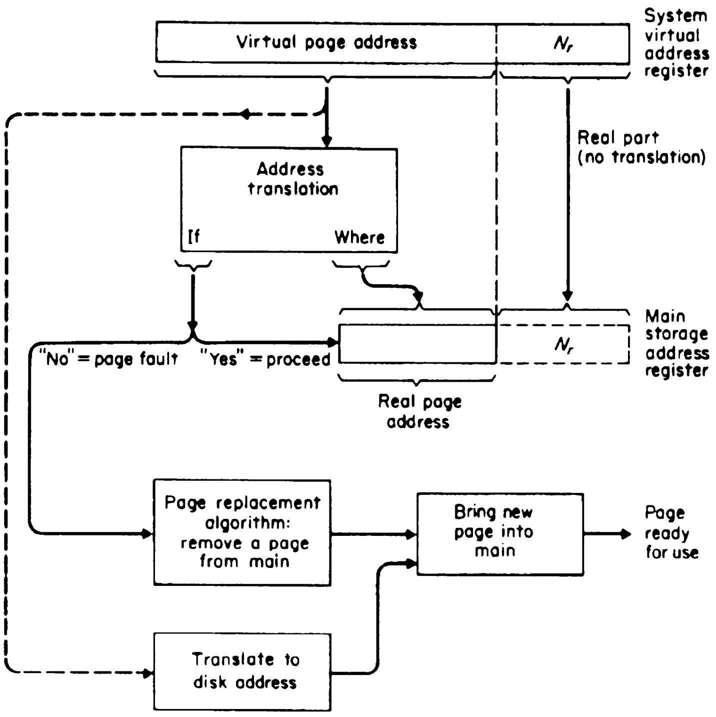

Virtual page address
N_r
System virtual address register
Real part (no translation)
Address translation
If
Where
"No" = page fault
"Yes" = proceed
N_r
Main storage address register
Real page address
Page replacement algorithm: remove a page from main
Bring new page into main
Page ready for use
Translate to disk address

FIGURE 18.2.9 Block diagram of address-translation and page-replacement process.

Thus there are at last three fundamental requirements: (1) a mapping function to specify how pages from the disc are to be mapped into physical locations in memory, (2) an address translation function to determine *if* and *where* a virtual page is located in main memory, and (3) a replacement algorithm to determine which page in memory is to be removed when the *if* translation is a "no," i.e., when a page fault occurs. These are the three fundamental requirements needed to implement a virtual memory system, either a virtual main memory as here described, or a cache.

MAPPING FUNCTION AND ADDRESS TRANSLATION

The mapping function is a logical construct, whereas the address-translation function is the physical implementation of the mapping function. Mapping functions cover a range from *direct mapping* to fully *associative mapping*, with a continuum of *set-associative* mapping functions in between. A very simple way to understand maps is to consider the example of building one's own personal telephone directory "paged' out of larger telephone books. Assume that the personal directory is of fixed size, say 4(26) = 104 names, addresses, and associated telephone numbers. A direct mapping from the large books to the personal director is an alphabetical listing of the 104 names. Given any name, we can go directly to the directory entry, if present. Such an address translation could be hard-wired if desired. There are two difficulties with direct mapping: (1) it is very difficult to change, and (2) suppose we allow one entry for Jones. If later we wish to include another Jones, there is no room. If both Joneses are needed, there is a conflict unless we restructure the entire directory. Because of such conflicts, direct maps are seldom used.

At the other end of the spectrum is a *fully associative directory* in which 104 names in any combinations are placed in any positions of the directory. This directory is very easily changed because a name not frequently used can simply be removed and a new name entered in its place without regard to the logical (alphabetical) structure of the two names. For instance, if the directory is full and we wish to make a new entry Zeyer, we first find a name not used much recently. If Abas is in position 50 of the directory, remove Abas and replace the entry with Zeyer. The major difficulty, obviously, is in searching the directory. If we wish to know the number for Smith, we must associatively search the entire directory (worst case) to find the desired information. This is very time-consuming and impractical in most cases.

Imagine the usual telephone directory that is associatively organized. There are several ways to resolve the fundamental conflict between ease of search and ease of change. The fully associative directory can be augmented with a separate, directly organized and accessed table that contains a list of all names. However, the only other piece of data is a number indicating the entry this name now occupies in the associative directory. If a directory entry is changed, the new entry number must be placed in this table. If we wish to access a given name, a direct access to the table gives the entry number . A subsequent direct access to this entry number gives the desired address and telephone number.

The penalty is the two accesses plus the storage and maintenance of the translation table. Nevertheless, this is exactly the scheme used in all virtual main memories paged out of disc, drum, or tape. This table is typically broken into a hierarchy of two tables called *segment table* and *page table* to facilitate the sharing of segments among users and to allow the tables to also be pages as units of, say, 4K B. These tables are built, manipulated, and maintained by supervisory programs and generally are invisible to the user. Although these tables can consume large amounts of main memory and of system overhead, the saving greatly exceeds the loss.

An example of such a virtual memory with a fully associative mapping using a two-level table-translation scheme is illustrated in Fig. 18.2.10. Each user has a separate segment and page table (the two are, in principle, one table) stored in main memory along with the user's data and programs. When an access is required to a virtual address, say N_vN_r, as in Fig. 18.2.10, a sequence of several accesses is required. The user ID register bits μ give a direct address to the origin of that user's segment table in main memory. The higher-order segment-index bits (SI) of the virtual address (typically 4 to 8 b) specify the index (depth) into this table for the required entry. This segment table entry contains a flag specifying *if* this entry is valid or not, and a *where* specifying the origin of that user's page table in memory, as shown. The lower-order page index bits (PI) of the virtual address specify the index into the page table as shown. The page-table entry so accessed contains an *if* bit to indicate whether the entry is valid (*if* page is present in main memory) and a *where* address that

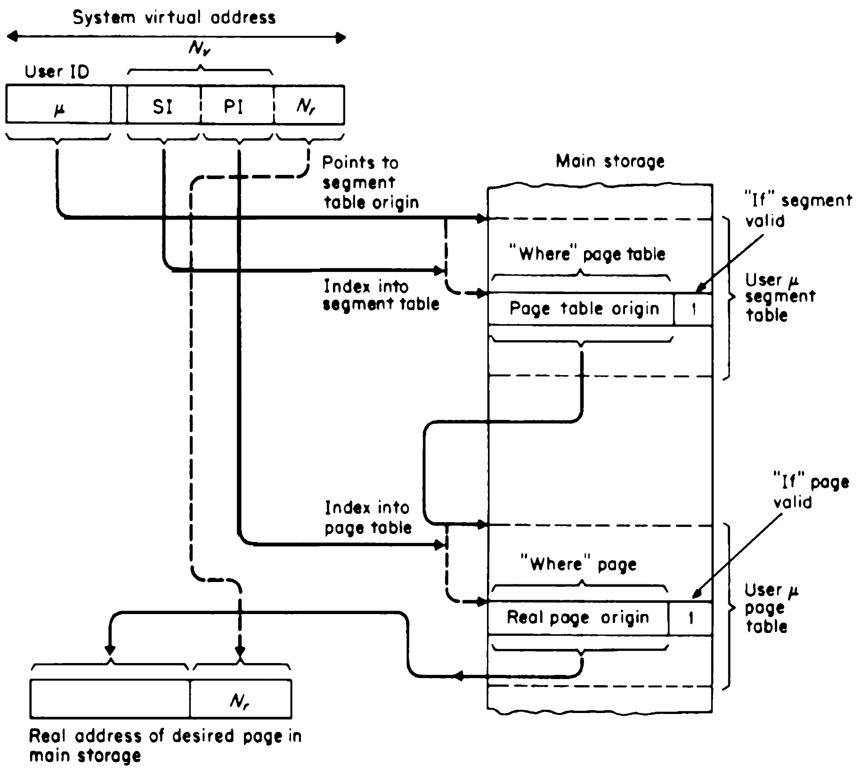


FIGURE 18.2.10 Virtual-storage-address translation using a two-level table (segment and page) for each user.

gives the real main-memory address of the desired page. The lower-order N_r bits of the address, typically 11 or 12 b for 2K- or 4K-B pages, are real, representing the word or byte of the page and hence do not require translation.

In cache memories, speed is so important that a fully associative mapping with a table-translation scheme is impractical. Instead a set-associative mapping and address translation is implemented directly in hardware. Although it is not generally realized, set-associative mapping and translation is commonly used in everyday life. It is a combination of a partially direct mapping and selection, with a partially associative mapping and translation. Examples include the 4-way set associative organization in Fig. 18.2.11 and the most common type of personal telephone directory as shown in Fig. 18.2.12, where a small index knob is moved to the first letter of the name being sought. Suppose there is one known position on the directory for each letter of the alphabet and we organize it with a set associativity of four. This means that for each letter there are exactly four entries or names possible, and these four can be in any order, as shown by the insert of Fig. 18.2.12. To find the telephone number for any given name such as Becker, we first do a direct selection to the letter B followed by an associative search over the four entries. Thus it is apparent that a set-associative access is a combination of the two limiting cases, namely, part direct and part associative. Many different combinations are possible with various advantages and disadvantages.

This set-associative directory (Fig. 18.2.12) with four names or entries per set is exactly the same fundamental type used in many cache-memory systems. The directory is implemented in a random-access memory array as shown in Fig. 18.2.11, where there are 128(4) entries total, requiring nine total virtual page address bits. Each set of 4 is part of one physical word (horizontal) of the random-access array, so there are 128 such

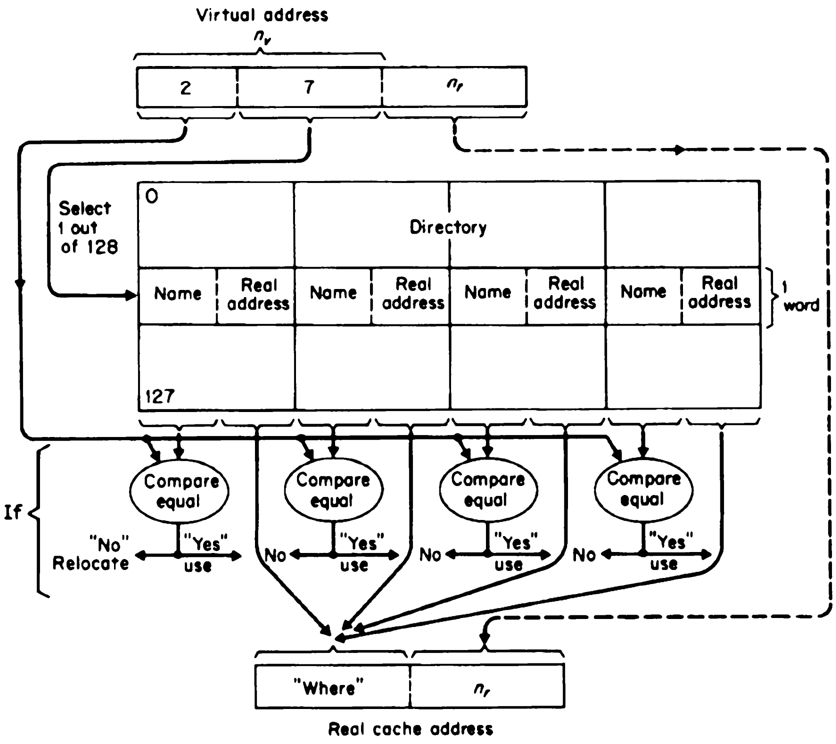


FIGURE 18.2.11 Cache directory using 4-way set associative organization.

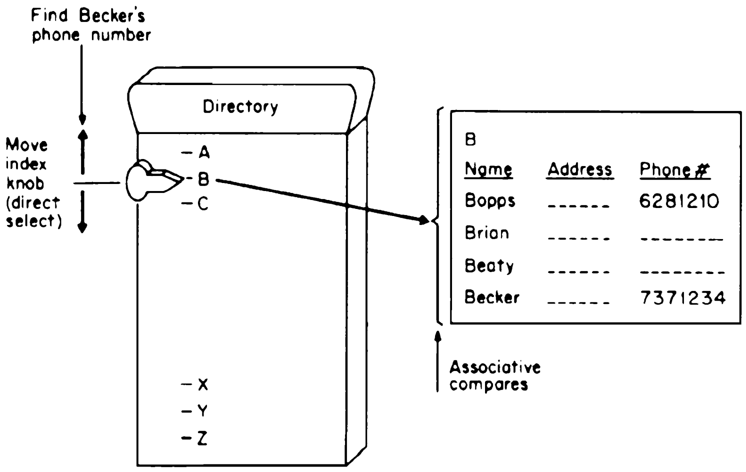


FIGURE 18.2.12 Fundamentals of a general set-associative directory showing direct and associative parts of the addressing process.

words, requiring seven address bits. The total virtual address $n_v = 9$ must be used in the address translation to determine *if* and *where* the cache page resides. As before, the lower-order bits n_r, which represent the byte within the page, need not be translated. Seven virtual bits are used to select directly one of the 128 sets as shown. This is analogous to moving the index knob in Fig. 18.2.12 and reads out all four names of the set. The NAME part is 2 b long, representing one of the four of the set. All four are compared simultaneously with the 2 b of the virtual address. *If* one of these gives a "yes" on compare equal, then the correct "real" address of the page in the cache, which resides in the directory with the correct NAME, is gated to the "real" cache-address register.