

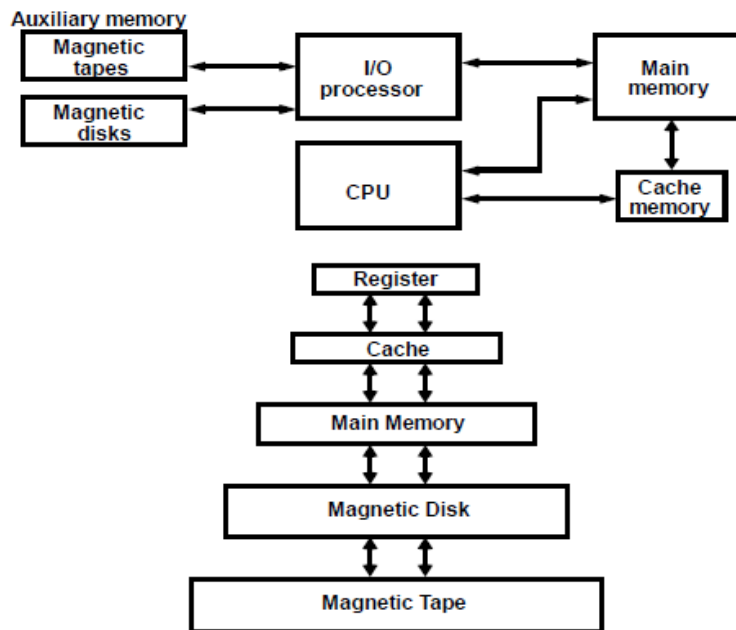
UNIT IV

Memory Organization

Memory Hierarchy:

MEMORY HIERARCHY

Memory Hierarchy is to obtain the highest possible access speed while minimizing the total cost of the memory system



- **The memory unit** is an essential component in any digital computer since it is needed for storing programs and data.
- **A very small computer** with a limited application may be able to fulfill its intended task without the need for additional storage capacity.
- **Most general purpose** computers would run more efficiently if they were equipped with additional storage beyond the capacity of the main memory.
- **There is just not enough** space in one memory unit to accommodate all the programs used in a typical computer.
- **Moreover**, most computer users accumulate and continue to accumulate large amounts of data-processing software.
- **Not all accumulated** information is needed by the processor at the same time.
- **Therefore**, it is more economical to use low-cost storage devices to serve as a backup for storing the information that is not currently used by the CPU.
- **The memory unit** that communicates directly with the CPU is called the main memory.

- **Devices that provide backup** storage are auxiliary memory.
- **The most common** auxiliary memory devices used in computer systems are magnetic disks and tapes.
- **They are used for storing** system programs, large data files, and other backup information.
- **Only programs** and data currently needed by the processor reside in main memory.
- **All other information** is stored in auxiliary memory and transferred to main memory when needed.
- **The total memory** capacity of a computer can be visualized as being a hierarchy of components.
- **The memory hierarchy** system consists of all storage devices employed in a computer system from the slow but high-capacity auxiliary memory to a relatively faster main memory, to an even smaller and faster cache memory accessible to the high-speed processing logic.
- **Figure 1** illustrates the components in a typical memory hierarchy. At the bottom of the hierarchy are the relatively slow magnetic tapes used to store removable files.
- **Next are the magnetic** disks used as backup storage. The main memory occupies a central position by being able to communicate directly with the CPU and with auxiliary memory devices through an I/O processor.
- **When programs not** residing in main memory are needed by the CPU, they are brought in from auxiliary memory.
- **Programs not currently** needed in main memory are transferred into auxiliary memory to provide space for currently used programs and data.
- **A special very-high speed memory** called a cache is sometimes used to increase the speed of processing by making current programs and data available to the CPU at a rapid rate.
- **The cache memory is employed in computer** systems to compensate for the speed differential between main memory access time and processor logic.
- **CPU logic is usually faster** than main memory access time, with the result that processing speed is limited primarily by the speed of main memory.
- **A technique used to compensate** for the mismatch in operating speeds is to employ an extremely fast, small cache between the CPU and main memory whose access time is close to processor logic clock cycle time.
- **The cache is used for storing segments** of programs currently being executed in the CPU and temporary data frequently needed in the present calculations.
- **By making programs** and data available at a rapid rate, it is possible to increase the performance rate of the computer.
- **While the I/O processor** manages data transfers between auxiliary memory and main memory, the cache organization is concerned with the transfer of information between main memory and CPU.
- **Thus each is involved** with a different level in the memory hierarchy system.

- **The reason for having two or three levels of memory hierarchy is economics.**
- **As the storage capacity** of the memory increases, the cost per bit for storing binary information decreases and the access time of the memory becomes longer.
- **The auxiliary memory** has a large storage capacity, is relatively inexpensive, but has low access speed compared to main memory.
- **The cache memory** is very small, relatively expensive, and has very high access speed. Thus as the memory access speed increases, so does its relative cost.
- **The overall goal** of using a memory hierarchy is to obtain the highest-possible average access speed while minimizing the total cost of the entire memory system.
- **Auxiliary and cache memories** are used for different purposes. The cache holds those parts of the program and data that are most heavily used, while the auxiliary memory holds those parts that are not presently used by the CPU.
- **Moreover, the CPU** has direct access to both cache and main memory but not to auxiliary memory. The transfer from auxiliary to main memory is usually done by means of direct memory access of large blocks of data.
- **The typical access time ratio between cache and main memory** is about 1 to 7. For example, a typical cache memory may have an access time of 100 ns, while main memory access time may be 700 ns.
- **Auxiliary memory average access time** is usually 1000 times that of main memory. Block size in auxiliary memory typically ranges from 256 to 2048 words, while cache block size is typically from 1 to 16 words.
- **Many operating systems** are designed to enable the CPU to process a number of independent programs concurrently.
- **This concept, called multiprogramming**, refers to the existence of two or more programs in different parts of the memory hierarchy at the same time.
- **In this way** it is possible to keep all parts of the computer busy by working with several programs in sequence. For example, suppose that a program is being executed in the CPU and an I/O transfer is required.
- **The CPU initiates** the I/O processor to start executing the transfer. This leaves the CPU free to execute another program. In a multiprogramming system, when one program is waiting for input or output transfer, there is another program ready to utilize the CPU.
- **With multiprogramming** the need arises for running partial programs, for varying the amount of main memory in use by a given program, and for moving programs around the memory hierarchy.
- **Computer programs** are sometimes too long to be accommodated in the total space available in main memory.
- **Moreover, a computer system** uses many programs and all the programs cannot reside in main memory at all times. A program with its data normally resides in auxiliary memory.

- **When the program** or a segment of the program is to be executed, it is transferred to main memory to be executed by the CPU. Thus one may think of auxiliary memory as containing the totality of information stored in a computer system.
- **It is the task** of the operating system to maintain in main memory a portion of this information that is currently active.
- **The part of the computer system** that supervises the flow of information between auxiliary memory and main memory is called the memory management system.

1. Main Memory

- **The main memory** is the central storage unit in a computer system. It is a relatively large and fast memory used to store programs and data during the computer operation.
- **The principal technology** used for the main memory is based on semiconductor integrated circuits.
- **Integrated circuit RAM chips** are available in two possible operating modes, static and dynamic. The static RAM consists essentially of internal flip-flops that store the binary information.
- **The stored information** remains valid as long as power is applied to the unit. The dynamic RAM stores the binary information in the form of electric charges that are applied to capacitors.
- **The capacitors are provided** inside the chip by MOS transistors.
- **The stored charge** on the capacitors tends to discharge with time and the capacitors must be periodically recharged by refreshing the dynamic memory.
- **Refreshing is done** by cycling through the words every few milliseconds to restore the decaying charge. The dynamic RAM offers reduced power consumption and larger storage capacity in a single memory chip.
- **The static RAM is easier** to use and has shorter read and write cycles. Most of the main memory in a general-purpose computer is made up of RAM integrated circuit chips, but a portion of the memory may be constructed with ROM chips.
- **Originally, RAM** was used to refer to a random-access memory, but now it is used to designate a read/write memory to distinguish it from a read-only memory, although ROM is also random access.
- **RAM is used** for storing the bulk of the programs and data that are subject to change.
- **ROM** is used for storing programs that are permanently resident in the computer and for tables of constants that do not change in value once the production of the computer is completed.
- **Among other things**, the ROM portion of main memory is needed for storing an initial program called a bootstrap loader.
- **The bootstrap loader** is a program whose function is to start the computer software operating when power is turned on.
- **Since RAM is volatile**, its contents are destroyed when power is turned off.

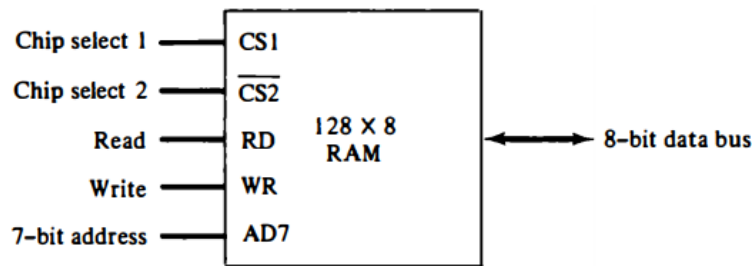
- **The contents of ROM** remain unchanged after power is turned off and on again.
- **The startup of a computer** consists of turning the power on and starting the execution of an initial program.
- **Thus when power** is turned on, the hardware of the computer sets the program counter to the first address of the bootstrap loader.
- **The bootstrap program** loads a portion of the operating system from disk to main memory and control is then transferred to the operating system, which prepares the computer for general use.
- **RAM and ROM chips** are available in a variety of sizes. If the memory needed for the computer is larger than the capacity of one chip, it is necessary to combine a number of chips to form the required memory size.
- **To demonstrate** the chip interconnection, we will show an example of a 1024 x 8 memory constructed with 128 x 8 RAM chips and 512 x 8 ROM chips.

2. RAM and ROM Chips

- **A RAM chip** is better suited for communication with the CPU if it has one or more control inputs that select the chip only when needed.
- **Another common** feature is a bidirectional data bus that allows the transfer of data either from memory to CPU during a read operation or from CPU to memory during a write operation.
- **A bidirectional bus** can be constructed with three-state buffers. A three-state buffer output can be placed in one of three possible states: a signal equivalent to logic 1, a signal equivalent to logic 0, or a high impedance state.
- **The logic 1 and 0** are normal digital signals. The high impedance state behaves like an open circuit, which means that the output does not carry a signal and has no logic significance.

The block diagram of a RAM chip is shown in Fig. 2. The capacity of the memory is 128 words of eight bits (one byte) per word. This requires a 7-bit address and an 8-bit bidirectional data bus.

Figure 2 Typical RAM chip.



(a) Block diagram

CS1	$\overline{\text{CS2}}$	RD	WR	Memory function	State of data bus
0	0	x	x	Inhibit	High-impedance
0	1	x	x	Inhibit	High-impedance
1	0	0	0	Inhibit	High-impedance
1	0	0	1	Write	Input data to RAM
1	0	1	x	Read	Output data from RAM
1	1	x	x	Inhibit	High-impedance

(b) Function table

- **The read and write** inputs specifies the memory operation and the two chips select (CS) control inputs are for enabling the chip only when it is selected by the microprocessor.
- **The availability** of more than one control input to select the chip facilitates the decoding of the address lines when multiple chips are used in the microcomputer.
- **The read and write inputs** are sometimes combined into one line labeled R/W. When the chip is selected, the two binary states in this line specify the two operations of read or write.
- **The function table** listed in Fig. 2(b) specifies the operation of the RAM chip. The unit is in operation only when CS1 = 1 and CS2 = 0. The bar on top of the second select variable indicates that this input is enabled when it is equal to 0.
- **If the chip select** inputs are not enabled, or if they are enabled but the read or write inputs are not enabled, the memory is inhibited and its data bus is in a high-impedance state.
- **When CS1 = 1** and CS2 = 0, the memory can be placed in a write or read mode.
- **When the WR input** is enabled, the memory stores a byte from the data bus into a location specified by the address input lines.
- **When the RD input** is enabled, the content of the selected byte is placed into the data bus. The RD and WR signals control the memory operation as well as the bus buffers associated with the bidirectional data bus.
- **A ROM chip** is organized externally in a similar manner. However, since a ROM can only read, the data bus can only be in an output mode. The block diagram of a ROM chip is shown in Fig. 3. For the same-size chip, it is possible to have more bits of ROM than of RAM, because the internal binary cells in ROM occupy less space than in RAM.
- **For this reason**, the diagram specifies a 512-byte ROM, while the RAM has only 128 bytes.
- **The nine address** lines in the ROM chip specify any one of the 512 bytes stored in it.

- **The two chip** select inputs must be CS1 = 1 and CS2 = 0 for the unit to operate. Otherwise, the data bus is in a high-impedance state.
- **There is no need** for a read or write control because the unit can only read. Thus when the chip is enabled by the two select inputs, the byte selected by the address lines appears on the data bus.

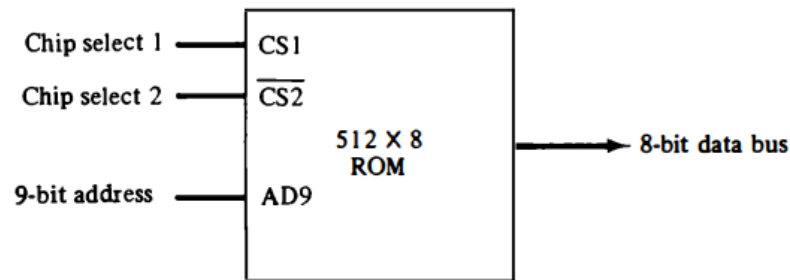


Figure 3 Typical ROM chip.

3. Memory Address Map

- **The designer** of a computer system must calculate the amount of memory required for the particular application and assign it to either RAM or ROM.
- **The interconnection** between memory and processor is then established from knowledge of the size of memory needed and the type of RAM and ROM chips available.
- **The addressing** of memory can be established by means of a table that specifies the memory address assigned to each chip.
- **The table**, called a memory address map, is a pictorial representation of assigned address space for each chip in the system.
- **To demonstrate** with a particular example, assume that a computer system needs 512 bytes of RAM and 512 bytes of ROM. The RAM and ROM chips to be used are specified in Figs. 2 and 3.
- **The memory address** map for this configuration is shown in Table 1.

TABLE 1 Memory Address Map for Microcomputer

Component	Hexadecimal address	Address bus							
		10	9	8	7	6	5	4	3 2 1
RAM 1	0000–007F	0	0	0	x	x	x	x	x x x x
RAM 2	0080–00FF	0	0	1	x	x	x	x	x x x x
RAM 3	0100–017F	0	1	0	x	x	x	x	x x x x
RAM 4	0180–01FF	0	1	1	x	x	x	x	x x x x
ROM	0200–03FF	1	x	x	x	x	x	x	x x x x

- **The component column** specifies whether a RAM or a ROM chip is used. The hexadecimal address column assigns a range of hexadecimal equivalent addresses for each chip.

- **The address** bus lines are listed in the third column. Although there are 16 lines in the address bus, the table shows only 10 lines because the other 6 are not used in this example and are assumed to be zero.
- **The small x's** under the address bus lines designate those lines that must be connected to the address inputs in each chip. The RAM chips have 128 bytes and need seven address lines.
- **The ROM chip has 512 bytes** and needs 9 address lines. The x's are always assigned to the low-order bus lines: lines 1 through 7 for the RAM and lines 1 through 9 for the ROM.
- **It is now necessary** to distinguish between four RAM chips by assigning to each a different address. For this particular example we choose bus lines 8 and 9 to represent four distinct binary combinations.
- **Note that any other** pair of unused bus lines can be chosen for this purpose. The table clearly shows that the nine low-order bus lines constitute a memory space for RAM equal to $2^9 = 512$ bytes.
- **The distinction** between a RAM and ROM address is done with another bus line. Here we choose line 10 for this purpose. When line 10 is 0, the CPU selects a RAM, and when this line is equal to 1, it selects the ROM.
- **The equivalent** hexadecimal address for each chip is obtained from the information under the address bus assignment. The address bus lines are subdivided into groups of four bits each so that each group can be represented with a hexadecimal digit.
- **The first hexadecimal** digit represents lines 13 to 16 and is always 0. The next hexadecimal digit represents lines 9 to 12, but lines 11 and 12 are always 0.
- **The range of hexadecimal** addresses for each component is determined from the x's associated with it. These x's represent a binary number that can range from an all-0's to an all-1's value.

4. Memory Connection to CPU

- **RAM and ROM chips** are connected to a CPU through the data and address buses.
- **The low-order lines** in the address bus select the byte within the chips and other lines in the address bus select a particular chip through its chip select inputs.
- **The connection of memory** chips to the CPU is shown in Fig. 4. This configuration gives a memory capacity of 512 bytes of RAM and 512 bytes of ROM. It implements the memory map of Table 1. Each RAM receives the seven low-order bits of the address bus to select one of 128 possible bytes.
- **The particular RAM chip** selected is determined from lines 8 and 9 in the address bus. This is done through a 2×4 decoder whose outputs go to the CS1 inputs in each RAM chip.
- **Thus, when address** lines 8 and 9 are equal to 00, the first RAM chip is selected. When 01, the second RAM chip is selected, and so on. The RD and WR outputs from the microprocessor are applied to the inputs of each RAM chip.
- **The selection between** RAM and ROM is achieved through bus line 10. The RAMs are selected when the bit in this line is 0, and the ROM when the bit is 1.

- **The other chip select** input in the ROM is connected to the RD control line for the ROM chip to be enabled only during a read operation. Address bus lines 1 to 9 are applied to the input address of ROM without going through the decoder.
- **This assigns addresses** 0 to 511 to RAM and 512 to 1023 to ROM. The data bus of the ROM has only an output capability, whereas the data bus connected to the RAMs can transfer information in both directions.
- **The example just shown** gives an indication of the interconnection complexity that can exist between memory chips and the CPU.
- **The more chips that** are connected, the more external decoders are required for selection among the chips. The designer must establish a memory map that assigns addresses to the various chips from which the required connections are determined.

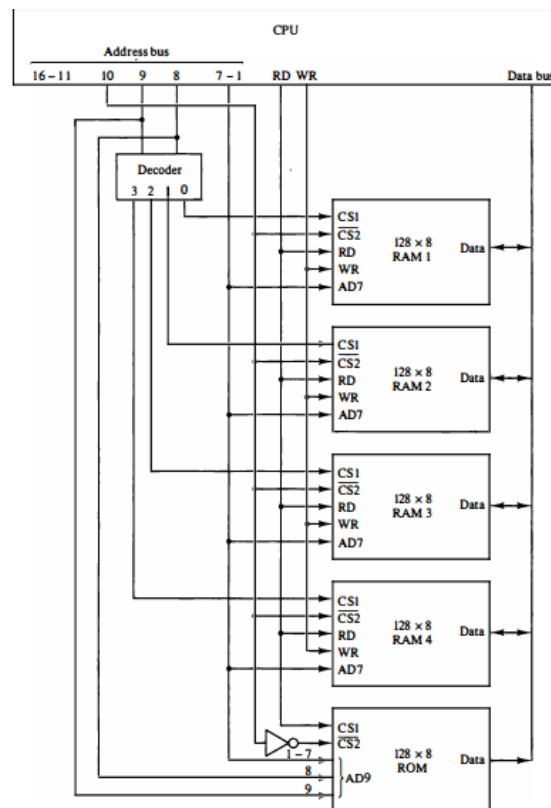


Figure 4 Memory connection to the CPU.

5. Auxiliary Memory

- **The most common** auxiliary memory devices used in computer systems are magnetic disks and tapes. Other components used, but not as frequently, are magnetic drums, magnetic bubble memory, and optical disks.
- **To understand** fully the physical mechanism of auxiliary memory devices one must have knowledge of magnetic, electronics, and electromechanical systems.
- **Although the physical** properties of these storage devices can be quite complex, their logical properties can be characterized and compared by a few parameters. The important

characteristics of any device are its access mode, access time, transfer rate, capacity, and cost.

- **The average time** required to reach a storage location in memory and obtain its contents is called the access time.
- **In electromechanical** devices with moving parts such as disks and tapes, the access time consists of a seek time required to position the read-write head to a location and a transfer time required to transfer data to or from the device. Because the seek time is usually much longer than the transfer time, auxiliary storage is organized in records or blocks.
- **A record is a specified number** of characters or words. Reading or writing is always done on entire records. The transfer rate is the number of characters or words that the device can transfer per second, after it has been positioned at the beginning of the record.
- **Magnetic drums and disks** are quite similar in operation. Both consist of high-speed rotating surfaces coated with a magnetic recording medium. The rotating surface of the drum is a cylinder and that of the disk, a round flat plate.
- **The recording surface** rotates at uniform speed and is not started or stopped during access operations. Bits are recorded as magnetic spots on the surface as it passes a stationary mechanism called a write head.
- **Stored bits are detected** by a change in magnetic field produced by a recorded spot on the surface as it passes through a read head. The amount of surface available for recording in a disk is greater than in a drum of equal physical size.
- **Therefore, more** information can be stored on a disk than on a drum of comparable size. For this reason, disks have replaced drums in more recent computers.

6. Magnetic Disks

- **A magnetic disk** is a circular plate constructed of metal or plastic coated with magnetized material. Often both sides of the disk are used and several disks may be stacked on one spindle with read/write heads available on each surface.
- **All disks rotate** together at high speed and are not stopped or started for access purposes. Bits are stored in the magnetized surface in spots along concentric circles called tracks.
- **The tracks are commonly** divided into sections called sectors. In most systems, the minimum quantity of information which can be transferred is a sector. The subdivision of one disk surface into tracks and sectors is shown in Fig. 5.

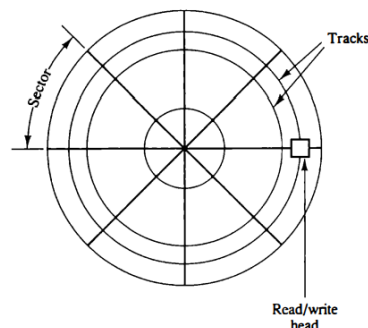


Figure 5 Magnetic disk.

- **Some units** use a single read/write head for each disk surface. In this type of unit, the track address bits are used by a mechanical assembly to move the head into the specified track position before reading or writing.
- **In other disk** systems, separate read/write heads are provided for each track in each surface. The address bits can then select a particular track electronically through a decoder circuit.
- **This type of unit** is more expensive and is found only in very large computer systems. Permanent timing tracks are used in disks to synchronize the bits and recognize the sectors.
- **A disk system** is addressed by address bits that specify the disk number, the disk surface, the sector number and the track within the sector. After the read/write heads are positioned in the specified track, the system has to wait until the rotating disk reaches the specified sector under the read/write head.
- **Information transfer** is very fast once the beginning of a sector has been reached. Disks may have multiple heads and simultaneous transfer of bits from several tracks at the same time.
- **A track in a given** sector near the circumference is longer than a track near the center of the disk. If bits are recorded with equal density, some tracks will contain more recorded bits than others.
- **To make all the records** in a sector of equal length, some disks use a variable recording density with higher density on tracks near the center than on tracks near the circumference. This equalizes the number of bits on all tracks of a given sector.
- **Disks that are permanently** attached to the unit assembly and cannot be removed by the occasional user are called hard disks. A disk drive with removable disks is called a floppy disk. The disks used with a floppy disk drive are small removable disks made of plastic coated with magnetic recording material.
- **There are two sizes** commonly used, with diameters of 5.25 and 3.5 inches. The 3.5-inch disks are smaller and can store more data than can the 5.25-inch disks.
- **Floppy disks** are extensively used in personal computers as a medium for distributing software to computer users.

7. Magnetic Tape

- **A magnetic tape** transport consists of the electrical, mechanical, and electronic components to provide the parts and control mechanism for a magnetic-tape unit.
- **The tape itself** is a strip of plastic coated with a magnetic recording medium. Bits are recorded as magnetic spots on the tape along several tracks.
- **Usually, seven** or nine bits are recorded simultaneously to form a character together with a parity bit. Read/write heads are mounted one in each track so that data can be recorded and read as a sequence of characters.
- **Magnetic tape units** can be stopped, started to move forward or in reverse, or can be rewound.
- **However, they** cannot be started or stopped fast enough between individual characters.

- **For this reason**, information is recorded in blocks referred to as records. Gaps of unrecorded tape are inserted between records where the tape can be stopped.
- **The tape starts** moving while in a gap and attains its constant speed by the time it reaches the next record. Each record on tape has an identification bit pattern at the beginning and end.
- **By reading the bit** pattern at the beginning, the tape control identifies the record number.
- **By reading the bit** pattern at the end of the record, the control recognizes the beginning of a gap.
- **A tape unit** is addressed by specifying the record number and the number of characters in the record. Records may be of fixed or variable length.

8. Associative Memory

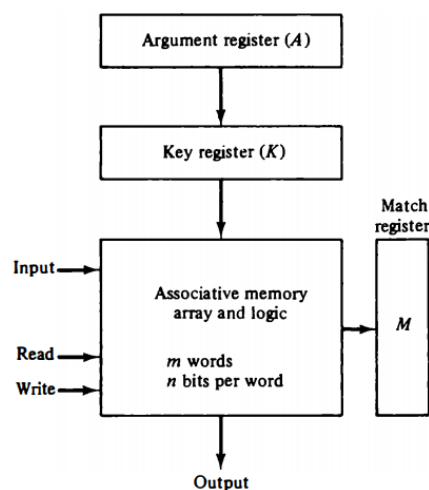
- **Many data-processing** applications require the search of items in a table stored in memory.
- **An assembler program** searches the symbol address table in order to extract the symbol's binary equivalent.
- **An account number** may be searched in a file to determine the holder's name and account status.
- **The established** way to search a table is to store all items where they can be addressed in sequence.
- **The search procedure** is a strategy for choosing a sequence of addresses, reading the content of memory at each address, and comparing the information read with the item being searched until a match occurs.
- **The number of accesses** to memory depends on the location of the item and the efficiency of the search algorithm.
- **Many search algorithms** have been developed to minimize the number of accesses while searching for an item in a random or sequential access memory.
- **The time required** to find an item stored in memory can be reduced considerably if stored data can be identified for access by the content of the data itself rather than by an address.
- **A memory unit accessed** by content is called an associative memory or content addressable memory (CAM).
- **This type of memory** is accessed simultaneously and in parallel on the basis of data content rather than by specific address or location. When a word is written in an associative memory, no address is given.
- **The memory** is capable of finding an empty unused location to store the word.
- **When a word** is to be read from an associative memory, the content of the word, or part of the word, is specified.
- **The memory locates** all words which match the specified content and marks them for reading.

- **Because of its organization**, the associative memory is uniquely suited to do parallel searches by data association. Moreover, searches can be done on an entire word or on a specific field within a word.
- **An associative memory** is more expensive than a random access memory because each cell must have storage capability as well as logic circuits for matching its content with an external argument.
- **For this reason**, associative memories are used in applications where the search time is very critical and must be very short.

9. Hardware Organization

- **The block diagram** of an associative memory is shown in Fig. 6. It consists of a memory array and logic for m words with n bits per word. The argument register A and key register K each have n bits, one for each bit of a word.
- **The match register M has m bits**, one for each memory word. Each word in memory is compared in parallel with the content of the argument register. The words that match the bits of the argument register set a corresponding bit in the match register.
- **After the matching process**, those bits in the match register that have been set indicate the fact that their corresponding words have been matched.
- **Reading is accomplished** by a sequential access to memory for those words whose corresponding bits in the match register have been set.
- **The key register provides** a mask for choosing a particular field or key in the argument word.
- **The entire argument** is compared with each memory word if the key register contains all 1's.
- **Otherwise, only those bits** in the argument that have 1's in their corresponding position of the key register are compared.

Figure 6 Block diagram of associative memory.

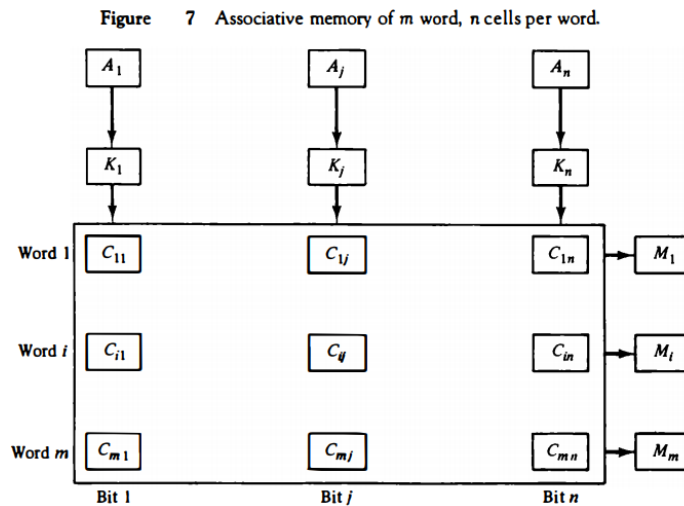


- **Thus the key** provides a mask or identifying piece of information which specifies how the reference to memory is made. To illustrate with a numerical example, suppose that the

argument register A and the key register K have the bit configuration shown below. Only the three leftmost bits of A are compared with memory words because K has 1's in these positions.

A	101 111100	
K	111 000000	
Word 1	100 111100	no match
Word 2	101 000001	match

- **Word 2 matches** the unmasked argument field because the three leftmost bits of the argument and the word are equal.
- **The relation between** the memory array and external registers in an associative memory is shown in Fig. 7.
- **The cells in the array** are marked by the letter C with two subscripts. The first subscript gives the word number and the second specifies the bit position in the word.
- **Thus cell C_{ij}** is the cell for bit j in word i. A bit A_j in the argument register is compared with all the bits in column j of the array provided that $K_j = 1$.
- **This is done for all columns $j = 1, 2, \dots, n$.** If a match occurs between all the unmasked bits of the argument and the bits in word i, the corresponding bit M_i in the match register is set to 1.
- **If one or more unmasked bits** of the argument and the word do not match, M_i is cleared to 0.



10. Match Logic

- **The match logic** for each word can be derived from the comparison algorithm for two binary numbers.
- **First**, we neglect the key bits and compare the argument in A with the bits stored in the cells of the words. Word i is equal to the argument in A if $A_j = F_{ij}$ for $j = 1, 2, \dots, n$. Two bits are equal if they are both 1 or both 0.

$$M_i = \prod_{j=1}^n (A_j F_{ij} + A'_j F'_{ij} + K'_j)$$

- **where \prod is a product** symbol designating the AND operation of all n terms. We need m such functions, one for each word $i = 1, 2, 3, \dots, m$.
- **The circuit for matching** one word is shown in Fig. 9. Each cell requires two AND gates and one OR gate. The inverters for A_j and K_j are needed once for each column and are used for all bits in the column.
- **The output of all OR** gates in the cells of the same word go to the input of a common AND gate to generate the match signal for M_i . M_i will be logic 1 if a match occurs and 0 if no match occurs.
- **Note** that if the key register contains all 0's, output M_j will be a 1 irrespective of the value of A or the word. This occurrence must be avoided during normal operation.

11. Read Operation

- **If more** than one word in memory matches the unmasked argument field, all the matched words will have 1's in the corresponding bit position of the match register.
- **It is then** necessary to scan the bits of the match register one at a time. The matched words are read in sequence by applying a read signal to each word line whose corresponding M_i bit is a 1.

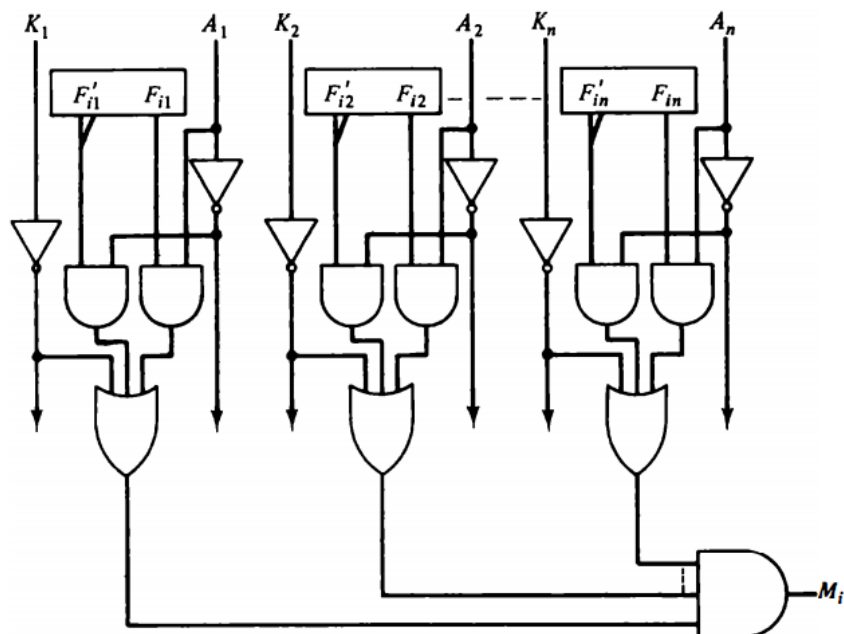


Figure 9 Match logic for one word of associative memory.

- **In most applications**, the associative memory stores a table with no two identical items' under a given key.

- **In this case**, only one word may match the unmasked argument field. By connecting output M_i directly to the read line in the same word position (instead of the M register), the content of the matched word will be presented automatically at the output lines and no special read command signal is needed.
- **Furthermore**, if we exclude words having a zero content, an all-zero output will indicate that no match occurred and that the searched item is not available in memory

12. Write Operation

- **An associative memory** must have a write capability for storing the information to be searched.
- **Writing** in an associative memory can take different forms, depending on the application.
- **If the entire memory** is loaded with new information at once prior to a search operation then the writing can be done by addressing each location in sequence.
- **This will make** the device a random access memory for writing and a content addressable memory for reading. The advantage here is that the address for input can be decoded as in a random access memory.
- **Thus instead** of having m address lines, one for each word in memory, the number of address lines can be reduced by the decoder to d lines, where $m = 2^d$.
- **If unwanted words** have to be deleted and new words inserted one at a time, there is a need for a special register to distinguish between active and inactive words.
- **This register**, sometimes called a tag register, would have as many bits as there are words in the memory.
- **For every active** word stored in memory, the corresponding bit in the tag register is set to 1. A word is deleted from memory by clearing its tag bit to 0. Words are stored in memory by scanning the tag register until the first 0 bit is encountered.
- **This gives** the first available inactive word and a position for writing a new word. After the new word is stored in memory it is made active by setting its tag bit to 1. An unwanted word when deleted from memory can be cleared to all 0's if this value is used to specify an empty location.
- **Moreover**, the words that have a tag bit of 0 must be masked (together with the K_i bits) with the argument word so that only active words are compared.

13. Cache Memory

- **Analysis** of a large number of typical programs has shown that the references to memory at any given interval of time tend to be confined within a few localized areas in memory.
- **This phenomenon** is known as the property of locality of reference locality of reference.
- **The reason** for this property may be understood considering that a typical computer program flows in a straight-line fashion with program loops and subroutine calls encountered frequently.

- **When a program** loop is executed, the CPU repeatedly refers to the set of instructions in memory that constitute the loop. Every time a given subroutine is called, its set of instructions is fetched from memory.
- **Thus loops** and subroutines tend to localize the references to memory for fetching instructions. To a lesser degree, memory references to data also tend to be localized.
- **Table-lookup** procedures repeatedly refer to that portion in memory where the table is stored. Iterative procedures refer to common memory locations and array of numbers are confined within a local portion of memory.
- **The result** of all these observations is the locality of reference property, which states that over a short interval of time, the addresses generated by a typical program refer to a few localized areas of memory repeatedly, while the remainder of memory is accessed relatively infrequently.
- **If the active** portions of the program and data are placed in a fast small memory, the average memory access time can be reduced, thus reducing the total execution time of the program.
- **Such a fast small** memory is referred to as a cache memory. It is placed between the CPU and main memory as illustrated in Fig. 1 .
- **The cache memory access** time is less than the access time of main memory by a factor of 5 to 10. The cache is the fastest component in the memory hierarchy and approaches the speed of CPU components.
- **The fundamental idea** of cache organization is that by keeping the most frequently accessed instructions and data in the fast cache memory, the average memory access time will approach the access time of the cache.
- **Although** the cache is only a small fraction of the size of main memory, a large fraction of memory requests will be found in the fast cache memory because of the locality of reference property of programs.
- **The basic operation** of the cache is as follows. When the CPU needs to access memory, the cache is examined. If the word is found in the cache, it is read from the fast memory.
- **If the word addressed** by the CPU is not found in the cache, the main memory is accessed to read the word. A block of words containing the one just accessed is then transferred from main memory to cache memory. The block size may vary from one word (the one just accessed) to about 16 words adjacent to the one just accessed. In this manner, some data are transferred to cache so that future references to memory find the required words in the fast cache memory.
- **The performance** of cache memory is frequently measured in terms of a quantity called hit ratio. When the CPU refers to memory and finds the word in cache, it is said to produce a hit. If the word is not found in cache, it is in main memory and it counts as a miss.
- **The ratio of the number of hits divided** by the total CPU references to memory (hits plus misses) is the hit ratio.
- **The hit ratio** is best measured experimentally by running representative programs in the computer and measuring the number of hits and misses during a given interval of time.

- **Hit ratios** of 0.9 and higher have been reported. This high ratio verifies the validity of the locality of reference property.
- **The average memory** access time of a computer system can be improved considerably by use of a cache. If the hit ratio is high enough so that most of the time the CPU accesses the cache instead of main memory, the average access time is closer to the access time of the fast cache memory.
- **For example**, a computer with cache access time of 100 ns, a main memory access time of 1000 ns, and a hit ratio of 0.9 produces an average access time of 200 ns. This is a considerable improvement over a similar computer without a cache memory, whose access time is 1000 ns.
- **The basic characteristic** of cache memory is its fast access time. Therefore, very little or no time must be wasted when searching for words in the cache. The transformation of data from main memory to cache memory is referred to as a mapping process.

Three types of mapping procedures are of practical interest when considering the organization of cache memory:

1. Associative mapping
2. Direct mapping
3. Set-associative mapping

- **To help** in the discussion of these three mapping procedures we will use a specific example of a memory organization as shown in Fig. 10. The main memory can store 32K words of 12 bits each.
- **The cache** is capable of storing 512 of these words at any given time. For every word stored in cache, there is a duplicate copy in main memory.

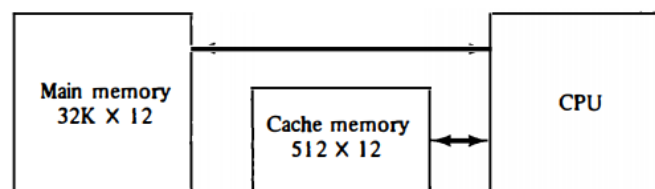


Figure 10 Example of cache memory.

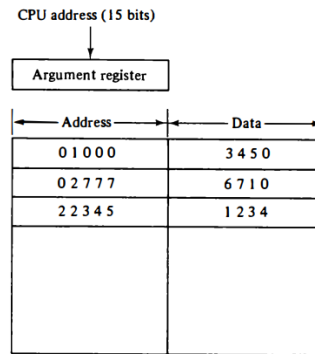
- **The CPU** communicates with both memories. It first sends a 15-bit address to cache. If there is a hit, the CPU accepts the 12-bit data from cache. If there is a miss, the CPU reads the word from main memory and the word is then transferred to cache.

14. Associative Mapping

- **The fastest** and most flexible cache organization uses an associative memory. This organization is illustrated in Fig. 11.
- **The associative memory** stores both the address and content (data) of the memory word.
- **This permits any** location in cache to store any word from main memory.
- **The diagram shows** three words presently stored in the cache.

- The **address value** of 15 bits is shown as a five-digit octal number and its corresponding 12-bit word is shown as a four-digit octal number.
- A **CPU address** of 15 bits is placed in the argument register and the associative memory is searched for a matching address.

Figure 11 Associative mapping cache (all numbers in octal).

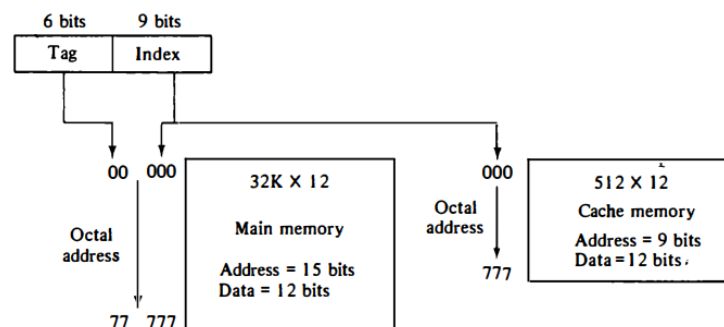


- If the **address** is found, the corresponding 12-bit data is read and sent to the CPU.
- If **no match occurs**, the main memory is accessed for the word. The address--data pair is then transferred to the associative cache memory.
- If the **cache** is full, an address--data pair must be displaced to make room for a pair that is needed and not presently in the cache.
- The **decision** as to what pair is replaced is determined from the replacement algorithm that the designer chooses for the cache.
- A **simple procedure** is to replace cells of the cache in round-robin order whenever a new word is requested from main memory.
- This constitutes a first-in first-out (FIFO) replacement policy.

15. Direct Mapping

- **Associative memories** are expensive compared to random-access memories because of the added logic associated with each cell.

Figure 12 Addressing relationships between main and cache memories.



- The **possibility** of using a random-access memory for the cache is investigated in Fig. 12.

- **The CPU** address of 15 bits is divided into two fields. The nine least significant bits constitute the index field and the remaining six bits form the tag field.
- **The figure** shows that main memory needs an address that includes both the tag and the index bits.
- **The number of bits** in the index field is equal to the number of address bits required to access the cache memory.
- **In the general case**, there are 2^k words in cache memory and 2^n words in main memory.
- **The n-bit memory address** is divided into two fields: k bits for the index field and n - k bits for the tag field.
- **The direct mapping cache** organization uses the n-bit address to access the main memory and the k-bit index to access the cache.
- **The internal** organization of the words in the cache memory is as shown in Fig. 13(b).

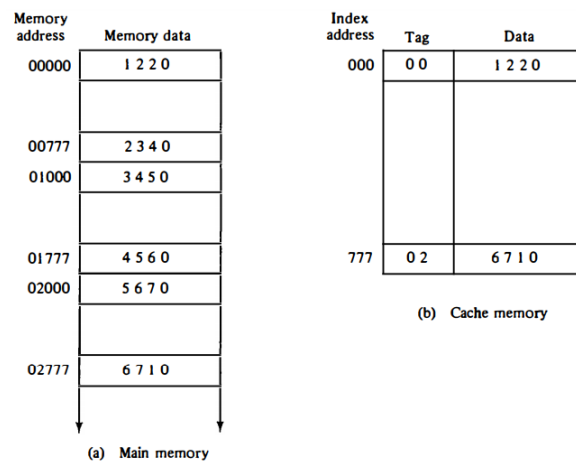


Figure : 13 Direct mapping cache organization.

- **Each word** in cache consists of the data word and its associated tag.
- **When a new word** is first brought into the cache, the tag bits are stored alongside the data bits.
- **When the CPU generates** a memory request, the index field is used for the address to access the cache.
- **The tag field** of the CPU address is compared with the tag in the word read from the cache.
- **If the two tags** match, there is a hit and the desired data word is in cache.
- **If there is no match**, there is a miss and the required word is read from main memory. It is then stored in the cache together with the new tag, replacing the previous value.
- **The disadvantage** of direct mapping is that the hit ratio can drop considerably if two or more words whose addresses have the same index but different tags are accessed repeatedly.

- **However, this possibility** is minimized by the fact that such words are relatively far apart in the address range (multiples of 512 locations in this example.)
- **To see how the direct-mapping** organization operates, consider the numerical example shown in Fig. 13. The word at address zero is presently stored in the cache (index = 000, tag = 00, data = 1220).
- **Suppose that the CPU** now wants to access the word at address 02000. The index address is 000, so it is used to access the cache. The two tags are then compared.
- **The cache tag** is 00 but the address tag is 02, which does not produce a match. Therefore, the main memory is accessed and the data word 5670 is transferred to the CPU.
- **The cache word** at index address 000 is then replaced with a tag of 02 and data of 5670.
- **The direct-mapping example** just described uses a block size of one word. The same organization but using a block size of B words is shown in Fig. 14.

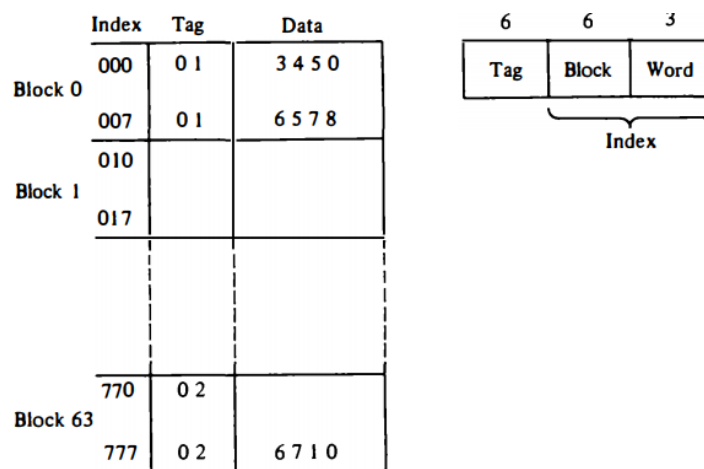


Figure 14 Direct mapping cache with block size of 8 words.

- **The index field** is now divided into two parts: the block field and the word field.
- **In a 512-word cache** there are 64 blocks of 8 words each, since $64 \times 8 = 512$.
- **The block number** is specified with a 6-bit field and the word within the block is specified with a 3-bit field.
- **The tag field** stored within the cache is common to all eight words of the same block.
- **Every time a miss** occurs, an entire block of eight words must be transferred from main memory to cache memory.
- **Although this takes** extra time, the hit ratio will most likely improve with a larger block size because of the sequential nature of computer programs.

16. Set-Associative Mapping

- **It was mentioned** previously that the disadvantage of direct mapping is that two words with the same index in their address but with different tag values cannot reside in cache memory at the same time.

- **A third type of cache** organization, called set-associative mapping, is an improvement over the direct mapping organization in that each word of cache can store two or more words of memory under the same index address.
- **Each data word** is stored together with its tag and the number of tag-data items in one word of cache is said to form a set.
- **An example** of a set-associative cache organization for a set size of two is shown in Fig. 15.
- **Each index address** refers to two data words and their associated tags. Each tag requires six bits and each data word has 12 bits, so the word length is $2(6 + 12) = 36$ bits. An index address of nine bits can accommodate 512 words. Thus the size of cache memory is 512×36 .
- **It can** accommodate 1024 words of main memory since each word of cache contains two data words.
- **In general**, a set-associative cache of set size k will accommodate k words of main memory in each word of cache.

Index	Tag	Data	Tag	Data
000	0 1	3 4 5 0	0 2	5 6 7 0
777	0 2	6 7 1 0	0 0	2 3 4 0

Figure 15 Two-way set-associative mapping cache.

- **The octal** numbers listed in Fig. 15 are with reference to the main memory contents illustrated in Fig. 13(a).
- **The words stored** at addresses 01000 and 02000 of main memory are stored in cache memory at index address 000.
- **Similarly**, the words at addresses 02777 and 00777 are stored in cache at index address 777.
- **When the CPU** generates a memory request, the index value of the address is used to access the cache. The tag field of the CPU address is then compared with both tags in the cache to determine if a match occurs.
- **The comparison logic** is done by an associative search of the tags in the set similar to an associative memory search: thus the name "set-associative."
- **The hit ratio** will improve as the set size increases because more words with the same index but different tags can reside in cache.

- **However**, an increase in the set size increases the number of bits in words of cache and requires more complex comparison logic.
- **When a miss** occurs in a set-associative cache and the set is full, it is necessary to replace one of the tag-data items with a new value.
- **The most common replacement** algorithms used are: random replacement, first-in, firstout (FIFO), and least recently used (LRU).
- **With the random replacement policy** the control chooses one tag-data item for replacement at random. The FIFO procedure selects for replacement the item that has been in the set the longest.
- **The LRU algorithm** selects for replacement the item that has been least recently used by the CPU.
- **Both FIFO and LRU** can be implemented by adding a few extra bits in each word of cache.

17. Writing into Cache

- **An important aspect** of cache organization is concerned with memory write requests. When the CPU finds a word in cache during a read operation, the main memory is not involved in the transfer.
- **However**, if the operation is a write, there are two ways that the system can proceed.
- **The simplest** and most commonly used procedure is to update main memory with every memory write operation, with cache memory being updated in parallel if it contains the word at the specified address.
- **This is called** the write-through method.
- **This method** has the advantage that main memory always contains the same data as the cache.
- **This characteristic** is important in systems with direct memory access transfers.
- **It ensures** that the data residing in main memory are valid at all times so that an I/O device communicating through DMA would receive the most recent updated data.
- **The second procedure** is called the write-back method. In this method only the cache location is updated during a write operation.
- **The location** is then marked by a flag so that later when the word is removed from the cache it is copied into main memory.
- **The reason** for the write-back method is that during the time a word resides in the cache, it may be updated several times; however, as long as the word remains in the cache, it does not matter whether the copy in main memory is out of date, since requests from the word are filled from the cache.
- **It is only** when the word is displaced from the cache that an accurate copy need be rewritten into main memory.
- **Analytical** results indicate that the number of memory writes in a typical program ranges between 10 and 30 percent of the total references to memory.

18. Cache Initialization

- **One more aspect** of cache organization that must be taken into consideration is the problem of initialization.
- **The cache** is initialized when power is applied to the computer or when the main memory is loaded with a complete set of programs from auxiliary memory.
- **After initialization** the cache is considered to be empty, but in effect it contains some non valid data.
- **It is customary to** include with each word in cache a valid bit to indicate whether or not the word contains valid data.
- **The cache is initialized** by clearing all the valid bits to 0.
- **The valid bit of** a particular cache word is set to 1 the first time this word is loaded from main memory and stays set unless the cache has to be initialized again.
- **The introduction** of the valid bit means that a word in cache is not replaced by another word unless the valid bit is set to 1 and a mismatch of tags occurs.
- **If the valid** bit happens to be 0, the new word automatically replaces the invalid data.
- **Thus** the initialization condition has the effect of forcing misses from the cache until it fills with valid data.

19. Virtual Memory

- **In a memory hierarchy** system, programs and data are first stored in auxiliary memory. Portions of a program or data are brought into main memory as they are needed by the CPU.
- **Virtual memory** is a concept used in some large computer systems that permit the user to construct programs as though a large memory space were available, equal to the totality of auxiliary memory.
- **Each address** that is referenced by the CPU goes through an address mapping from the so-called virtual address to a physical address in main memory.
- **Virtual memory** is used to give programmers the illusion that they have a very large memory at their disposal, even though the computer actually has a relatively small main memory.
- **A virtual memory** system provides a mechanism for translating program-generated addresses into correct main memory locations. This is done dynamically, while programs are being executed in the CPU.
- **The translation or mapping** is handled automatically by the hardware by means of a mapping table.

20. Address Space and Memory Space

- **An address used** by a programmer will be called a virtual address, and the set of such addresses the address space.
- **An address in main memory** is called a location or physical address.

- **The set of such locations** is called the memory space.
- **Thus the address space** is the set of addresses generated by programs as they reference instructions and data; the memory space consists of the actual main memory locations directly addressable for processing.
- **In most computers** the address and memory spaces are identical. The address space is allowed to be larger than the memory space in computers with virtual memory.
- **As an illustration**, consider a computer with a main-memory capacity of 32K words ($K = 1024$). Fifteen bits are needed to specify a physical address in memory since $32K = 2^{15}$. Suppose that the computer has available auxiliary memory for storing $2^{20} = 1024K$ words.
- **Thus auxiliary memory** has a capacity for storing information equivalent to the capacity of 32 main memories. Denoting the address space by N and the memory space by M , we then have for this example $N = 1024K$ and $M = 32K$.
- **In a multiprogram** computer system, programs and data are transferred to and from auxiliary memory and main memory based on demands imposed by the CPU.
- **Suppose that program 1** is currently being executed in the CPU. Program 1 and a portion of its associated data are moved from auxiliary memory into main memory as shown in Fig. 16.
- **Portions of programs and data** need not be in contiguous locations in memory since information is being moved in and out, and empty spaces may be available in scattered locations in memory.
- **In a virtual memory system**, programmers are told that they have the total address space at their disposal.
- **Moreover**, the address field of the instruction code has a sufficient number of bits to specify all virtual addresses.
- **In our example**, the address field of an instruction code will consist of 20 bits but physical memory addresses must be specified with only 15 bits.
- **Thus CPU will** reference instructions and data with a 20-bit address, but the information at this address must be taken from physical memory because access to auxiliary storage for individual words will be prohibitively long.

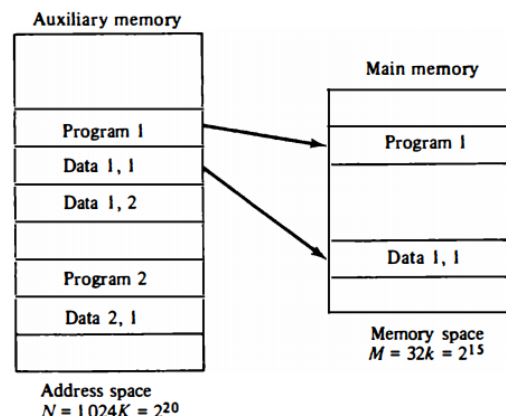
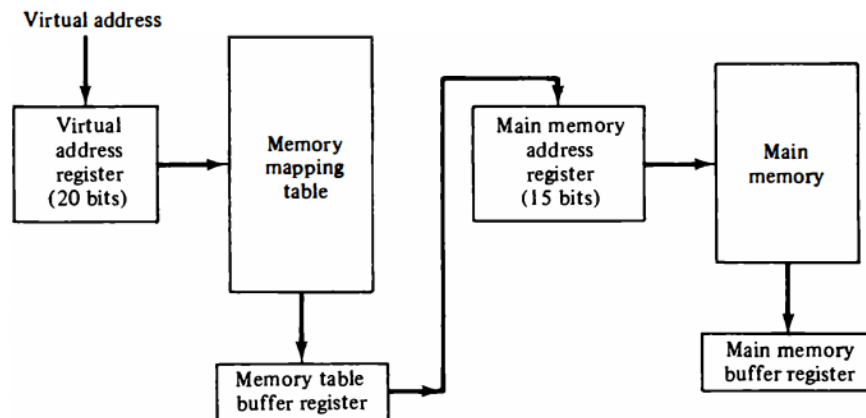


Figure 16 Relation between address and memory space in a virtual memory system.

- **(Remember** that for efficient transfers, auxiliary storage moves an entire record to the main memory.)
- **A table** is then needed, as shown in Fig. 17, to map a virtual address of 20 bits to a physical address of 15 bits. The mapping is a dynamic operation, which means that every address is translated immediately as a word is referenced by CPU.
- **The mapping table** may be stored in a separate memory as shown in Fig. 17 or in main memory. In the first case, an additional memory unit is required as well as one extra memory access time.

Figure 17 Memory table for mapping a virtual address.

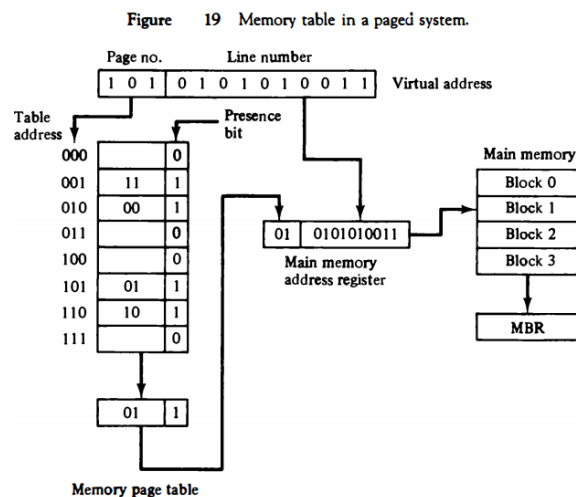


- **In the second case**, the tab takes space from main memory and two accesses to memory are required with the program running at half speed.

21. Address Mapping Using Pages

- **The table implementation** of the address mapping is simplified if the information in the address space and the memory space are each divided into groups of fixed size.
- **The physical memory** is broken down into groups of equal size called blocks, which may range from 64 to 4096 words each.
- **The term page refers** to groups of address space of the same size. For example, if a page or block consists of 1K words, then, using the previous example, address space is divided into 1024 pages and main memory is divided into 32 blocks.
- **Although** both a page and a block are split into groups of 1K words, a page refers to the organization of address space, while a block refers to the organization of memory space.
- **The programs** are also considered to be split into pages. Portions of programs are moved from auxiliary memory to main memory in records equal to the size of a page. The term "page frame" is sometimes used to denote a block.
- **Consider a computer** with an address space of 8K and a memory space of 4K.
- **If we split** each into groups of 1K words we obtain eight pages and four blocks as shown in Fig. 18. At any given time, up to four pages of address space may reside in main memory in any one of the four blocks.

- **The mapping from address space to memory space** is facilitated if each virtual address is considered to be represented by two numbers: a page number address and a line within the page.
- **In a computer** with 2^p words per page, p bits are used to specify a line address and the remaining high-order bits of the virtual address specify the page number. In the example of Fig. 18, a virtual address has 13 bits.
- **Since each page** consists of $2^{10} = 1024$ words, the high order three bits of a virtual address will specify one of the eight pages and the low-order 10 bits give the line address within the page. Note that the line address in address space and memory space is the same; the only mapping required is from a page number to a block number.
- **The organization** of the memory mapping table in a paged system is shown in Fig. 19.
- **The memory-page table** consists of eight words, one for each page. The address in the page table denotes the page number and the content of the word gives the block number where that page is stored in main memory.
- **The table shows** that pages 1, 2, 5, and 6 are now available in main memory in blocks 3, 0, 1, and 2, respectively.
- **A presence bit** in each location indicates whether the page has been transferred from auxiliary memory into main memory. A 0 in the presence bit indicates that this page is not available in main memory.
- **The CPU references** a word in memory with a virtual address of 13 bits. The three high-order bits of the virtual address specify a page number and also an address for the memory-page table.



- **The content of the word** in the memory page table at the page number address is read out into the memory table buffer register.
- **If the presence bit** is a 1, the block number thus read is transferred to the two high-order bits of the main memory address register.
- **The line number** from the virtual address is transferred into the 10 low-order bits of the memory address register. A read signal to main memory **transfers** the content of the word to the main memory buffer register ready to be used by the CPU.

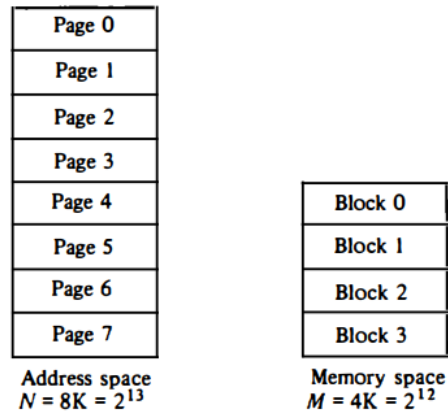


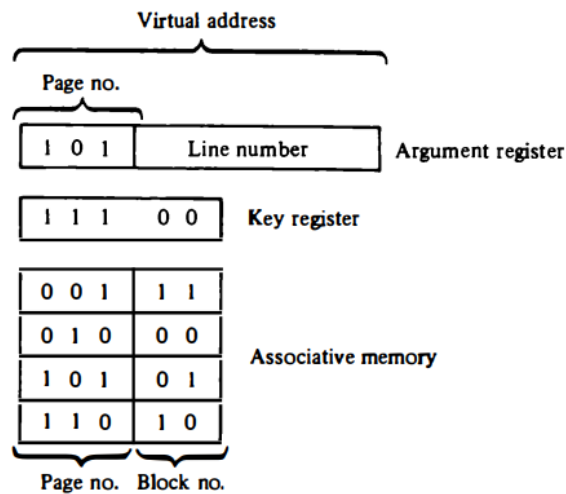
Figure 18 Address space and memory space split into groups of 1K words.

- **If the presence bit** in the word read from the page table is 0, it signifies that the content of the word referenced by the virtual address does not reside in main memory.
- **A call to the operating system** is then generated to fetch the required page from auxiliary memory and place it into main memory before resuming computation.

22. Associative Memory Page Table

- **A random-access** memory page table is inefficient with respect to storage utilization.
- **In the example** of Fig. 19 we observe that eight words of memory are needed, one for each page, but at least four words will always be marked empty because main memory cannot accommodate more than four blocks.
- **In general, a system** with n pages and m blocks would require a memory-page table of n locations of which up to m blocks will be marked with block numbers and all others will be empty.
- **As a second** numerical example, consider an address space of 1024K words and memory space of 32K words.
- **If each page or block contains 1K words**, the number of pages is 1024 and the number of blocks 32. The capacity of the memory-page table must be 1024 words and only 32 locations may have a presence bit equal to 1.
- **At any given time**, at least 992 locations will be empty and not in use.
- **A more efficient way** to organize the page table would be to construct it with a number of words equal to the number of blocks in main memory.
- **In this way the size of the memory** is reduced and each location is fully utilized. This method can be implemented by means of an associative memory with each word in memory containing a page number together with its corresponding block number.

Figure 20 An associative memory page table.



- The **page field** in each word is compared with the page number in the virtual address.
- If a **match occurs**, the word is read from memory and its corresponding block number is extracted. Consider again the case of eight pages and four blocks as in the example of Fig. 19.
- We **replace** the random access memory-page table with an associative memory of four words as shown in Fig. 20. Each entry in the associative memory array consists of two fields. The first three bits specify a field for storing the page number. The last two bits constitute a field for storing the block number.
- The **virtual address** is placed in the argument register. The page number bits in the argument register are compared with all page numbers in the page field of the associative memory.
- If the **page number** is found, the 5-bit word is read out from memory.
- The **corresponding block number**, being in the same word, is transferred to the main memory address register. If no match occurs, a call to the operating system is generated to bring the required page from auxiliary memory.

23. Page Replacement

- A **virtual memory system** is a combination of hardware and software techniques. The memory management software system handles all the software operations for the efficient utilization of memory space.
- It **must decide** (1) which page in main memory ought to be removed to make room for a new page, (2) when a new page is to be transferred from auxiliary memory to main memory, and (3) where the page is to be placed in main memory.
- The **hardware** mapping mechanism and the memory management software together constitute the architecture of a virtual memory.
- When a **program starts** execution, one or more pages are transferred into main memory and the page table is set to indicate their position.

- **The program** is executed from main memory until it attempts to reference a page that is still in auxiliary memory.
- **This condition** is called page fault. When page fault occurs, the execution of the present program is suspended until the required page is brought into main memory. Since loading a page from auxiliary memory to main memory is basically an I/O operation, the operating system assigns this task to the VO processor.
- **In the meantime**, control is transferred to the next program in memory that is waiting to be processed in the CPU. Later, when the memory block has been assigned and the transfer completed, the original program can resume its operation.
- **When a page fault occurs** in a virtual memory system, it signifies that the page referenced by the CPU is not in main memory.
- **A new page is then** transferred from auxiliary memory to main memory. If main memory is full, it would be necessary to remove a page from a memory block to make room for the new page. The policy for choosing pages to remove is determined from the replacement algorithm that is used.
- **The goal of a replacement policy** is to try to remove the page least likely to be referenced in the immediate future. Two of the most common replacement algorithms used are the first-in, first-out (FIFO) and the least recently used (LRU).
- **The FIFO algorithm selects** for replacement the page that has been in memory the longest time. Each time a page is loaded into memory, its identification number is pushed into a FIFO stack. FIFO will be full whenever memory has no more empty blocks.
- **When a new page must be loaded**, the page least recently brought in is removed. The page to be removed is easily determined because its identification number is at the top of the FIFO stack. The FIFO replacement policy has the advantage of being easy to implement.
- **It has the disadvantage** that under certain circumstances pages are removed and loaded from memory too frequently.
- **LRU The LRU policy** is more difficult to implement but has been more attractive on the assumption that the least recently used page is a better candidate for removal than the least recently loaded page as in FIFO.
- **The LRU algorithm** can be implemented by associating a counter with every page that is in main memory. When a page is referenced, its associated counter is set to zero.
- **At fixed intervals of time**, the counters associated with all pages presently in memory are incremented by 1. The least recently used page is the page with the highest count.
- **The counters are often** called aging registers, as their count indicates their age, that is, how long ago their associated pages have been referenced.

24. Memory Management Hardware

- **In a multiprogramming** environment where many programs reside in memory it becomes necessary to move programs and data around the memory, to vary the amount of memory in use by a given program, and to prevent a program from changing other programs.

- **The demands on computer memory** brought about by multiprogramming have created the need for a memory management system. A memory management system is a collection of hardware and software procedures for managing the various programs residing in memory.
- **The memory management software** is part of an overall operating system available in many computers. Here we are concerned with the hardware unit associated with the memory management system.

The basic components of a memory management unit are:

1. A facility for dynamic storage relocation that maps logical memory references into physical memory addresses
2. A provision for sharing common programs stored in memory by different users
3. Protection of information against unauthorized access between users and preventing users from changing operating system functions

- **The dynamic storage** relocation hardware is a mapping process similar to the paging system. The fixed page size used in the virtual memory system causes certain difficulties with respect to program size and the logical structure of programs.
- **It is more convenient** to divide programs and data into logical parts called segments. A segment is a set of logically related instructions or data elements associated with a given name.
- **Segments may be** generated by the programmer or by the operating system.
- **Examples of segments** are a subroutine, an array of data, a table of symbols, or a user's program.
- **The sharing** of common programs is an integral part of a multiprogramming system. For example, several users wishing to compile their Fortran programs should be able to share a single copy of the compiler rather than each user having a separate copy in memory.
- **Other system programs** residing in memory are also shared by all users in a multiprogramming system without having to produce multiple copies.
- **The third issue** in multiprogramming is protecting one program from unwanted interaction with another.
- **An example of unwanted** interaction is one user's unauthorized copying of another user's program. Another aspect of protection is concerned with preventing the occasional user from performing operating system functions and thereby interrupting the orderly sequence of operations in a computer installation.
- **The secrecy of certain** programs must be kept from unauthorized personnel to prevent abuses in the confidential activities of an organization.
- **The address generated** by a segmented program is called a logical address. This is similar to a virtual address except that logical address space is associated with variable-length segments rather than fixed-length pages.

- **The logical** address may be larger than the physical memory address as in virtual memory, but it may also be equal, and sometimes even smaller than the length of the physical memory address.
- **In addition to relocation** information, each segment has protection information associated with it. Shared programs are placed in a unique segment in each user's logical address space so that a single physical copy can be shared.
- **The function of the memory management** unit is to map logical addresses into physical addresses similar to the virtual memory mapping concept.

25. Segmented-Page Mapping

- **It was already** mentioned that the property of logical space is that it uses variable-length segments.
- **The length** of each segment is allowed to grow and contract according to the needs of the program being executed.
- **One way of** specifying the length of a segment is by associating with it a number of equal-size pages. To see how this is done, consider the logical address shown in Fig. 21 .
- **The logical address** is partitioned into three fields. The segment field specifies a segment number. The page field specifies the page within the segment and the word field gives the specific word within the page. A page field of k bits can specify up to 2^k pages.

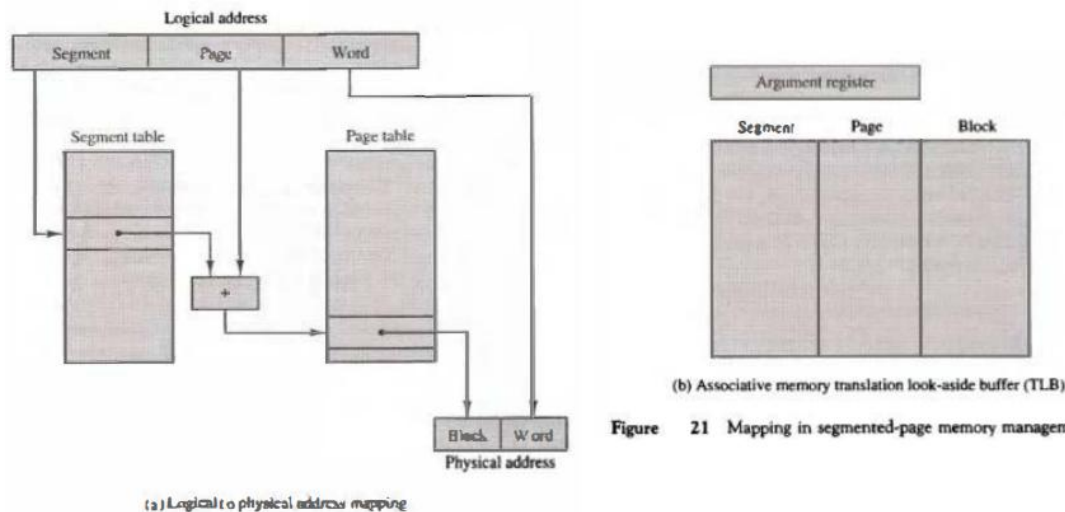


Figure 21 Mapping in segmented-page memory management unit.

- **A segment number may be associated** with just one page or with as many as 2^k -pages. Thus the length of a segment would vary according to the number of pages that are assigned to it.
- **The mapping** of the logical address into a physical address is done by means of two tables, as shown in Fig. 21(a). The segment number of the logical address specifies the address for the segment table.
- **The entry** in the segment table is a pointer address for a page table base. The page table base is added to the page number given in the logical address.

- **The sum** produces a pointer address to an entry in the page table. The value found in the page table provides the block number in physical memory.
- **The concatenation** of the block field with the word field produces the final physical mapped address. The two mapping tables may be stored in two separate small memories or in main memory.
- **In either case**, a memory reference from the CPU will require three accesses to memory: one from the segment table, one from the page table, and the third from main memory.
- **This would slow** the system significantly when compared to a conventional system that requires only one reference to memory.
- **To avoid this** speed penalty, a fast associative memory is used to hold the most recently referenced table entries.
- **(This type of memory** is sometimes called a translation look aside buffer, abbreviated TLB.) The first time a given block is referenced, its value together with the corresponding segment and page numbers are entered into the associative memory as shown in Fig. 21(b).
- **Thus the mapping process** is first attempted by associative search with the given segment and page numbers.
- **If it succeeds**, the mapping delay is only that of the associative memory. If no match occurs, the slower table mapping of Fig. 21(a) is used and the result transformed into the associative memory for future reference.

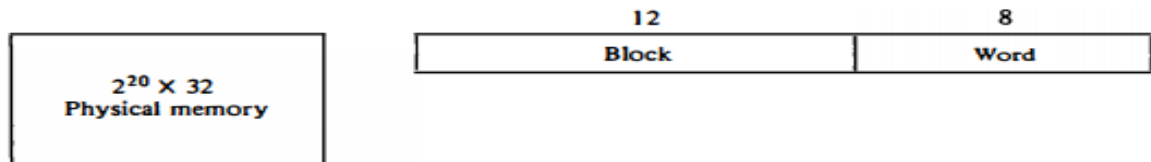
26. Numerical Example

- **A numerical example** may clarify the operation of the memory management unit. Consider the 20-bit logical address specified in Fig. 22(a). The 4-bit segment number specifies one of 16 possible segments.
- **The 8-bit page** number can specify up to 256 pages, and the 8-bit word field implies a page size of 256 words.
- **This configuration** allows each segment to have any number of pages up to 256. The smallest possible segment will have one page or 256 words. The largest possible segment will have 256 pages, for a total of $256 \times 256 = 64K$ words.
- **The physical memory shown** in Fig. 22(b) consists of 220 words of 32 bits each. The 20-bit address is divided into two fields: a 12-bit block number and an 8-bit word number.
- **Thus, physical memory** is divided into 4096 blocks of 256 words each. A page in a logical address has a corresponding block in physical memory.
- **Note that both the logical and physical address** have 20 bits. In the absence of a memory management unit, the 20-bit address from the CPU can be used to access physical memory directly.
- **Consider a program loaded** into memory that requires five pages. The operating system may assign to this program segment 6 and pages 0 through 4, as shown in Fig. 23(a). The total logical address range for the program is from hexadecimal 60000 to 604FF.

- **When the program** is loaded into physical memory, it is distributed among five blocks in physical memory where the operating system finds empty spaces. The correspondence between each memory block and logical page number is then entered in a table as shown in

4	8	8
Segment	Page	Word

(a) Logical address format: 16 segments of 256 pages each, each page has 256 words



(b) Physical address format: 4096 blocks of 256 words each, each word has 32 bits

Figure 22 An example of logical and physical addresses.

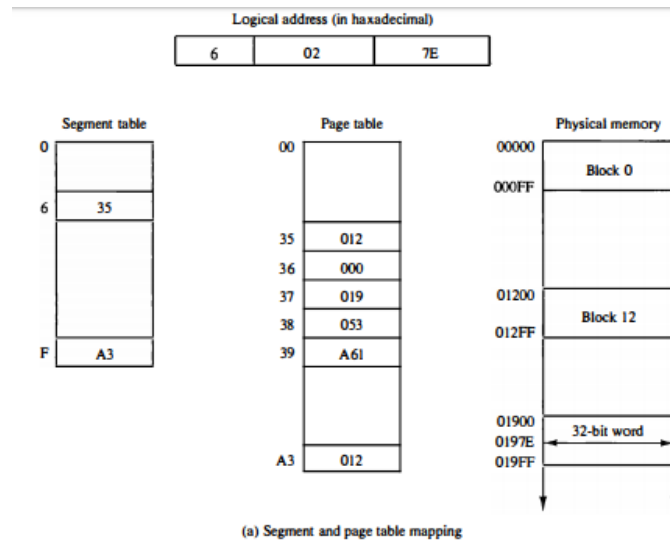
- **Fig. 23(b).** The information from this table is entered in the segment and page tables as shown in Fig. 24(a). Now consider the specific logical address given in Fig. 24.
- **The 20-bit** address is listed as a five-digit hexadecimal number.
- **It refers** to word number 7E of page 2 in segment 6. The base of segment 6 in the page table is at address 35. Segment 6 has associated with it five pages, as shown in the page table at addresses 35 through 39.
- **Page 2 of segment 6** is at address $35 + 2 = 37$. The physical memory block is found in the page table to be 019. Word 7E in block 19 gives the 20-bit physical address 0197E. Note that page 0 of segment 6 maps into block 12 and page 1 maps into block 0.

Figure 23 Example of logical and physical memory address assignment.

Hexadecimal address	Page number
60000	Page 0
60100	Page 1
60200	Page 2
60300	Page 3
60400	Page 4
604FF	

Segment	Page	Block
6	00	012
6	01	000
6	02	019
6	03	053
6	04	A61

(a) Logical address assignment (b) Segment-page versus memory block assignment



Segment	Page	Block
6	02	019
6	04	A61

(b) Associative memory (TLB)

Figure : 24 Logical to physical memory mapping example (all numbers are in hexadecimal).

- **The associative memory** in Fig. 24(b) shows that pages 2 and 4 of segment 6 have been referenced previously and therefore their corresponding block numbers are stored in the associative memory.
- **From this example** it should be evident that the memory management system can assign any number of pages to each segment.
- **Each logical page** can be mapped into any block in physical memory. Pages can move to different blocks in memory depending on memory space requirements.
- **The only updating** required is the change of the block number in the page table. Segments can grow or shrink independently without affecting each other.
- **Different** segments can use the same block of memory if it is required to share a program by many users.
- **For example, block number 12** in physical memory can be assigned a second logical address F0000 through F00FF. This specifies segment number 15 and page 0, which maps to block 12 as shown in Fig. 24(a).

27. Memory Protection

- **Memory protection** can be assigned to the physical address or the logical address.

- **The protection** of memory through the physical address can be done by assigning to each block in memory a number of protection bits that indicate the type of access allowed to its corresponding block.
- **Every time** a page is moved from one block to another it would be necessary to update the block protection bits.
- **A much better** place to apply protection is in the logical address space rather than the physical address space.
- **This can be done** by including protection information within the segment table or segment register of the memory management hardware.
- **The content of each entry** in the segment table or a segment register is called a descriptor.
- **A typical descriptor** would contain, in addition to a base address field, one or two additional fields for protection purposes. A typical format for a segment descriptor is shown in Fig. 25.
- **The base address** field gives the base of the page table address in a segmented-page organization or the block base address in a segment register organization.
- **This is the address** used in mapping from a logical to the physical address. The length field gives the segment size by specifying the maximum number of pages assigned to the segment.
- **The length field** is compared against the page number in the logical address. A size violation occurs if the page number falls outside the segment length boundary.
- **Thus a given program** and its data cannot access memory not assigned to it by the operating system. The protection field in a segment descriptor specifies the access rights available to the particular segment. In a segmented-page organization, each **entry in the page table** may have its own protection field to describe the access rights of each page.

Figure 25 Format of a typical segment descriptor.

Base address	Length	Protection
--------------	--------	------------

- **The protection information** is set into the descriptor by the master control program of the operating system. Some of the access rights of interest that are used for protecting the programs residing in memory are:

1. Full read and write privileges
2. Read only (write protection)
3. Execute only (program protection)
4. System only (operating system protection)

- **Full read and write** privileges are given to a program when it is executing its own instructions. Write protection is useful for sharing system programs such as utility programs and other library routines.

- **These system programs are** stored in an area of memory where they can be shared by many users. They can be read by all programs, but no writing is allowed. This protects them from being changed by other programs.
- **The execute-only** condition protects programs from being copied. It restricts the segment to be referenced only during the instruction fetch phase but not during the execute phase.
- **Thus it allows the users** to execute the segment program instructions but prevents them from reading the instructions as data for the purpose of copying their content.
- **Portions of the operating system** will reside in memory at any given time. These system programs must be protected by making them inaccessible to unauthorized users.
- **The operating system** protection condition is placed in the descriptors of all operating system programs to prevent the occasional user from accessing operating system segments.

28. Multiprocessors

- **A multiprocessor system** is an interconnection of two or more CPUs with memory and input-output equipment. The term "processor" In multiprocessor can mean either a central processing unit (CPU) or an input-output processor (IOP).
- **However**, a system with a single CPU and one or more IOPs is usually not included in the definition of a multiprocessor system unless the IOP has computational facilities comparable to a CPU.
- **As it is most commonly defined**, a multiprocessor system implies the existence of multiple CPUs, although usually there will be one or more IOPs as well.
- **Multiprocessors** are classified as multiple instruction stream, multiple data stream (MIMO) systems.
- **There are some similarities** between multiprocessor and multicomputer systems since both support concurrent operations.
- **However, there exists** an important distinction between a system with multiple computers and a system with multiple processors.
- **Computers are** interconnected with each other by means of communication lines to form a computer network.
- **The network consists** of several autonomous computers that may or may not communicate with each other.
- **A multiprocessor system** is controlled by one operating system that provides interaction between processors and all the components of the system cooperate in the solution of a problem.
- **Although some large-scale computers** include two or more CPUs in their overall system, it is the emergence of the microprocessor that has been the major motivation for multiprocessor systems.
- **The fact that microprocessors** take very little physical space and are very inexpensive brings about the feasibility of interconnecting a large number of microprocessors into one composite system.

- **Very-large-scale integrated circuit** technology has reduced the cost of computer components to such a low level that the concept of applying multiple processors to meet system performance requirements has become an attractive design possibility.
- **Multiprocessing** improves the reliability of the system so that a failure or error in one part has a limited effect on the rest of the system.
- **If a fault causes** one processor to fail, a second processor can be assigned to perform the functions of the disabled processor.
- **The system** as a whole can continue to function correctly with perhaps some loss in efficiency. The benefit derived from a multiprocessor organization is an improved system performance. The system derives its high performance from the fact that computations can proceed in parallel in one of two ways.

1. Multiple independent jobs can be made to operate in parallel.

2. A single job can be partitioned into multiple parallel tasks.

- **An overall function** can be partitioned into a number of tasks that each processor can handle individually.
- **System tasks may be allocated** to special purpose processors whose design is optimized to perform certain types of processing efficiently.
- **An example is a computer system** where one processor performs the computations for an industrial process control while others monitor and control the various parameters, such as temperature and flow rate.
- **Another example** is a computer where one processor performs high speed floating-point mathematical computations and another takes care of routine data-processing tasks.
- **Multiprocessing** can improve performance by decomposing a program into parallel executable tasks. This can be achieved in one of two ways. The user can explicitly declare that certain tasks of the program be executed in parallel.
- **This must be done prior** to loading the program by specifying the parallel executable segments.
- **Most multiprocessor manufacturers** provide an operating system with programming language constructs suitable for specifying parallel processing.
- **The other, more efficient way** is to provide a compiler with multiprocessor software that can automatically detect parallelism in a user's program.
- **The compiler checks for data dependency** in the program. If a program depends on data generated in another part, the part yielding the needed data must be executed first.
- **However, two parts of a program** that do not use data generated by each can run concurrently.
- **The parallelizing compiler** checks the entire program to detect any possible data dependencies. These that have no data dependency are then considered for concurrent scheduling on different processors.

- **Multiprocessors** are classified by the way their memory is organized. A multiprocessor system with common shared memory is classified as a shared memory or tightly coupled multiprocessor.
- **This does not preclude each processor** from having its own local memory. In fact, most commercial tightly coupled multiprocessors provide a cache memory with each CPU. In addition, there is a global common memory that all CPUs can access.
- **Information can therefore** be shared among the CPUs by placing it in the common global memory. An alternative model of microprocessor is the distributed-memory or loosely coupled system.
- **Each processor element** in a loosely coupled system has its own private local memory. The processors are tied together by a switching scheme designed to route information from one processor to another through a message-passing scheme.
- **The processors relay program** and data to other processors in packets. A packet consists of an address, the data content, and some error detection code.
- **The packets are addressed** to a specific processor or taken by the first available processor, depending on the communication system used.
- **Loosely coupled systems** are most efficient when the interaction between tasks is minimal, whereas tightly coupled systems can tolerate a higher degree of interaction between tasks.

29. Interconnection Structures

- **The components** that form a multiprocessor system are CPUs, IOPs connected to input-output devices, and a memory unit that may be partitioned into a number of separate modules.
- **The interconnection between the components** can have different physical configurations, depending on the number of transfer paths that are available between the processors and memory in a shared memory system or among the processing elements in a loosely coupled system.
- **There are several physical forms** available for establishing an interconnection network such as.

1. Time-shared common bus
2. Multiport memory
3. Crossbar switch
4. Multistage switching network
5. Hypercube system

30. Time-Shared Common Bus

- **A common-bus** multiprocessor system consists of a number of processors connected through a common path to a memory unit. A time-shared common bus for five processors is

shown in Fig. 1. Only one processor can communicate with the memory or another processor at any given time.

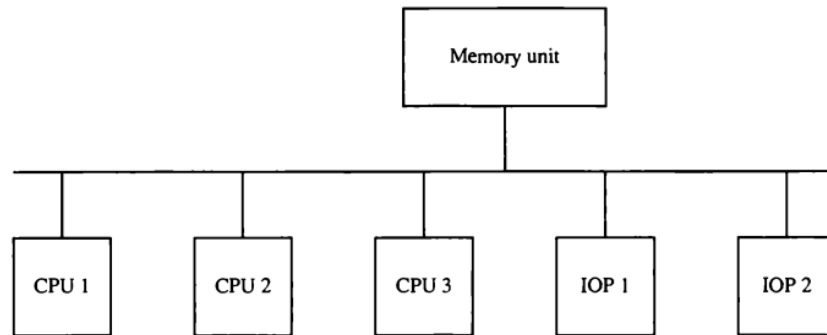


Figure 1 Time-shared common bus organization.

- **Transfer operations** are conducted by the processor that is in control of the bus at the time.
- **Any other processor** wishing to initiate a transfer must first determine the availability status of the bus, and only after the bus becomes available can the processor address the destination unit to initiate the transfer.
- **A command is issued to inform the destination unit** what operation is to be performed.
- **The receiving unit recognizes** its address in the bus and responds to the control signals from the sender, after which the transfer is initiated.
- **The system may** exhibit transfer conflicts since one common bus is shared by all processors. These conflicts must be resolved by incorporating a bus controller that establishes priorities among the requesting units.
- **A single common-bus system** is restricted to one transfer at a time.
- **This means that when one processor** is communicating with the memory, all other processors are either busy with internal operations or must be idle waiting for the bus.
- **As a consequence**, the total overall transfer rate within the system is limited by the speed of the single path.
- **The processors in the system** can be kept busy more often through the implementation of two or more independent buses to permit multiple simultaneous bus transfers. However, this increases the system cost and complexity.
- **A more economical implementation** of a dual bus structure is depicted in Fig. 2.

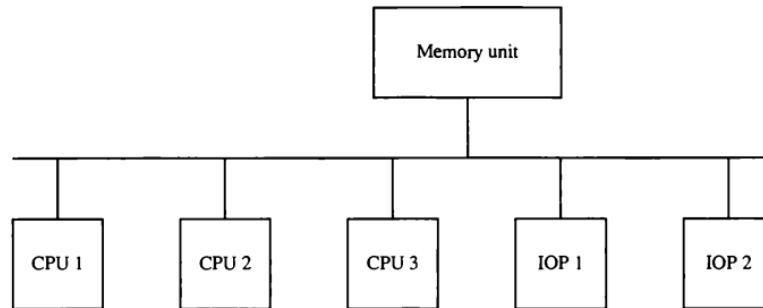


Figure 1 Time-shared common bus organization.

- **Here we have a number** of local buses each connected to its own local memory and to one or more processors. Each local bus may be connected to a CPU, an IOP, or any combination of processors.
- **A system bus controller** links each local bus to a common system bus. The I/O devices connected to the local IOP, as well as the local memory, are available to the local processor.
- **The memory connected** to the common system bus is shared by all processors. If an IOP is connected directly to the system bus, the I/O devices attached to it may be made available to all processors.
- **Only one processor** can communicate with the shared memory and other common resources through the system bus at any given time.
- **The other processors** are kept busy communicating with their local memory and I/O devices. Part of the local memory may be designed as a cache memory attached to the CPU.
- **In this way, the average** access time of the local memory can be made to approach the cycle time of the CPU to which it is attached.

31. Multiport Memory

- **A multiport memory** system employs separate buses between each memory module and each CPU.
- **This is shown in Fig. 3** for four CPUs and four memory modules (MMs).
- **Each processor bus** is Connected to each memory module.
- **A processor bus** consists of the address, data, and control lines required to communicate with memory.
- **The memory module** is said to have four ports and each port accommodates one of the buses. The module must have internal control logic to determine which port will have access to memory at any given time. Memory access conflicts are resolved by assigning fixed priorities to each memory port.
- **The priority** for memory access associated with each processor may be established by the physical port position that its bus occupies in each module.
- **Thus CPU 1** will have priority over CPU 2, CPU 2 will have priority over CPU 3, and CPU 4 will have the lowest priority.

- **The advantage** of the multi port memory organization is the high transfer rate that can be achieved because of the multiple paths between processors and memory.
- **The disadvantage** is that it requires expensive memory control logic and a large number of cables and connectors. As a consequence, this interconnection structure is usually appropriate for systems with a small number of processors.

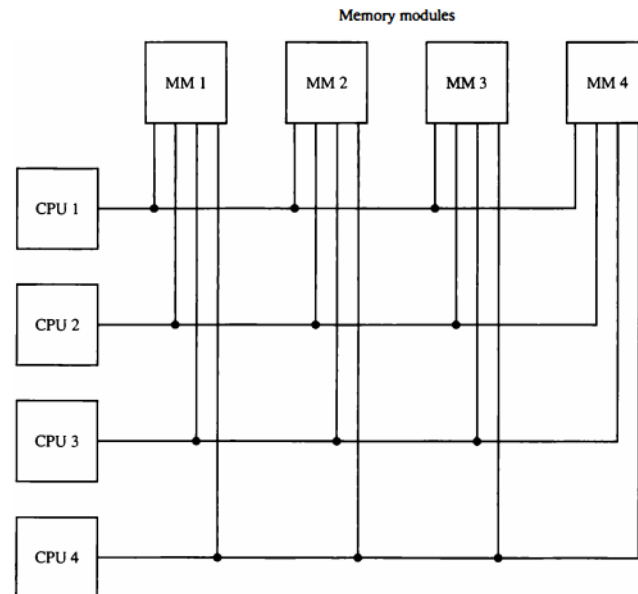


Figure 3 Multiport memory organization.

32. Interprocessor Communication and Synchronization

- **The various processors** in a multiprocessor system must be provided with a facility for communicating with each other. A communication path can be established through common input-output channels.
- **In a shared memory** multiprocessor system, the most common procedure is to set aside a portion of memory that is accessible to all processors.
- **The primary use of the** common memory is to act as a message center similar to a mailbox, where each processor can leave messages for other processors and pick up messages intended for it.
- **The sending processor** structures a request, a message, or a procedure, and places it in the memory mailbox. Status bits residing in common memory are generally used to indicate the condition of the mailbox, whether it has meaningful information, and for which processor it is intended.
- **The receiving processor can check the mailbox** periodically to determine if there are valid messages for it. The response time of this procedure can be time consuming since a processor will recognize a request only when polling messages.
- **A more efficient procedure** is for the sending processor to alert the receiving processor directly by means of an interrupt signal.

- **This can be accomplished through** a software-initiated interprocessor interrupt by means of an instruction in the program of one processor which when executed produces an external interrupt condition in a second processor.
- **This alerts the interrupted processor** of the fact that a new message was inserted by the interrupting processor.
- **In addition to shared memory**, a multiprocessor system may have other shared resources. For example, a magnetic disk storage unit connected to an IOP may be available to all CPUs.
- **This provides a facility** for sharing of system programs stored in the disk. A communication path between two CPUs can be established through a link between two I/O Ps associated with two different CPUs. This type of link allows each CPU to treat the other as an I/O device so that messages can be transferred through the I/O path.
- **To prevent conflicting** use of shared resources by several processors there must be a provision for assigning resources to processors.
- **This task is given to the operating system.** There are three organizations that have been used in the design of operating system for multiprocessors: master-slave configuration, separate operating system, and distributed operating system.
- **In a master-slave mode**, one processor, designated the master, always executes the operating system functions. The remaining processors, denoted as slaves, do not perform operating system functions.
- **If a slave processor** needs an operating system service, it must request it by interrupting the master and waiting until the current program can be interrupted.
- **In the separate operating system** organization, each processor can execute the operating system routines it needs.
- **This organization is more suitable** for loosely coupled systems where every processor may have its own copy of the entire operating system.
- **In the distributed operating system** organization, the operating system routines are distributed among the available processors.
- **However, each particular** operating system function is assigned to only one processor at a time.
- **This type of organization** is also referred to as a floating operating system since the routines float from one processor to another and the execution of the routines may be assigned to different processors at different times. In a loosely coupled multiprocessor system the memory is distributed among the processors and there is no shared memory for passing information. The communication between processors is by means of message passing through I/O channels.
- **The communication** is initiated by one processor calling a procedure that resides in the memory of the processor with which it wishes to communicate. When the sending processor and receiving processor name each other as a source and destination, a channel of communication is established.
- **A message is then sent** with a header and various data objects used to communicate between nodes. There may be a number of possible paths available to send the message between any two nodes.

- **The operating system in each node** contains routing information indicating the alternative paths that can be used to send a message to other nodes.
- **The communication efficiency** of the interprocessor network depends on the communication routing protocol, processor speed, data link speed, and the topology of the network.

33. Interprocessor Synchronization

- **The instruction set** of a multiprocessor contains basic instructions that are used to implement communication and synchronization between cooperating processes.
- **Communication** refers to the exchange of data between different processes.
- **For example, parameters** passed to a procedure in a different processor constitute interprocessor communication. Synchronization refers to the special case where the data used to communicate between processors is control information.
- **Synchronization is needed** to enforce the correct sequence of processes and to ensure mutually exclusive access to shared writable data. Multiprocessor systems usually include various mechanisms to deal with the synchronization of resources.
- **Low-level primitives** are implemented directly by the hardware. These primitives are the basic mechanisms that enforce mutual exclusion for more complex mechanisms implemented in software.
- **A number of hardware** mechanisms for mutual exclusion have been developed. One of the most popular methods is through the use of a binary semaphore

34. Mutual Exclusion with a Semaphore

- **A properly functioning** multiprocessor system must provide a mechanism that will guarantee orderly access to shared memory and other shared resources.
- **This is necessary** to protect data from being changed simultaneously by two or more processors. This mechanism has been termed mutual exclusion. Mutual exclusion must be provided in a multiprocessor system to enable one processor to exclude or lock out access to a shared resource by other processors when it is in a critical section.
- **A critical section** is a program sequence that, once begun, must complete execution before another processor accesses the same shared resource.
- **A binary variable** called a semaphore is often used to indicate whether or not a processor is executing a critical section. A semaphore is a software controlled flag that is stored in a memory location that all processors can access.
- **When the semaphore** is equal to 1, it means that a processor is executing a critical program, so that the shared memory is not available to other processors.
- **When the semaphore** is equal to 0, the shared memory is available to any requesting processor. Processors that share the same memory segment agree by convention not to use the memory segment unless the semaphore is equal to 0, indicating that memory is available. They also agree to set the semaphore to 1 when they are executing a critical section and to clear it to 0 when they are finished.

- **Testing and setting** the semaphore is itself a critical operation and must be performed as a single indivisible operation.
- **If it is not**, two or more processors may test the semaphore simultaneously and then each set it, allowing them to enter a critical section at the same time.
- **This action would allow** simultaneous execution of critical section, which can result in erroneous initialization of control parameters and a loss of essential information.
- **A semaphore** can be initialized by means of a test and set instruction in conjunction with a hardware lock mechanism. A hardware lock is a processor generated signal that serves to prevent other processors from using the system bus as long as the signal is active.
- **The test-and-set** instruction tests and sets a semaphore and activates the lock mechanism during the time that the instruction is being executed. This prevents other processors from changing the semaphore between the time that the processor is testing it and the time that it is setting it.
- **Assume that** the semaphore is a bit in the least significant position of a memory word whose address is symbolized by SEM. Let the mnemonic TSL designate the "test and set while locked" operation.
- **The instruction** TSL SEM will be executed in two memory cycles (the first to read and the second to write without interference as follows:

- $R \leftarrow M[SEM]$ Test semaphore
- $M[SEM] \leftarrow 1$ Set semaphore

- **The semaphore** is tested by transferring its value to a processor register R and then it is set to 1. The value in R determines what to do next.
- **If the processor finds that $R = 1$** , it knows that the semaphore was originally set. (The fact that it is set again does not change the semaphore value.)
- **That means that another** processor is executing a critical section, so the processor that checked the semaphore does not access the shared memory. If $R = 0$, it means that the common memory (or the shared resource that the semaphore represents) is available.
- **The semaphore is set to 1** to prevent other processors from accessing memory. The processor can now execute the critical section. The last instruction in the program must clear location SEM to zero to release the shared resource to other processors.
- **Note that the lock signal** must be active during the execution of the test-and-set instruction. It does not have to be active once the semaphore is set.
- **Thus the lock mechanism** prevents other processors from accessing memory while the semaphore is being set. The semaphore itself, when set, prevents other processors from accessing shared memory while one processor is executing a critical section.

35. Cache Coherence

- **The primary advantage of cache** is its ability to reduce the average access time in uniprocessors.

- **When the processor** finds a word in cache during a read operation, the main memory is not involved in the transfer. If the operation is to write, there are two commonly used procedures to update memory.
- **In the write-through policy**, both cache and main memory are updated with every write operation. In the write-back policy, only the cache is updated and the location is marked so that it can be copied later into main memory.
- **In a shared memory** multiprocessor system, all the processors share a common memory. In addition, each processor may have a local memory, part or all of which may be a cache. The compelling reason for having separate caches for each processor is to reduce the average access time in each processor.
- **The same information** may reside in a number of copies in some caches and main memory.
- **To ensure the ability** of the system to execute memory operations correctly, the multiple copies must be kept identical. This requirement imposes a cache coherence problem.
- **A memory scheme** is coherent if the value returned on a load instruction is always the value given by the latest store instruction with the same address.
- **Without a proper solution** to the cache coherence problem, caching cannot be used in bus-oriented multiprocessors with two or more processors.

36. Conditions for Incoherence

- **Cache coherence problems** exist in multiprocessors with private caches because of the need to share writable data. Read-only data can safely be replicated without cache coherence enforcement mechanisms.
- **To illustrate the problem**, consider the three-processor configuration with private caches shown in Fig. 12. Sometime during the operation an element X from main memory is loaded into the three processors, P_1 , P_2 , and P_3 .

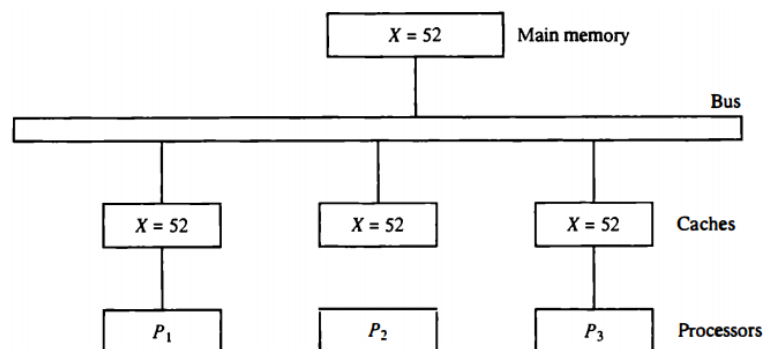
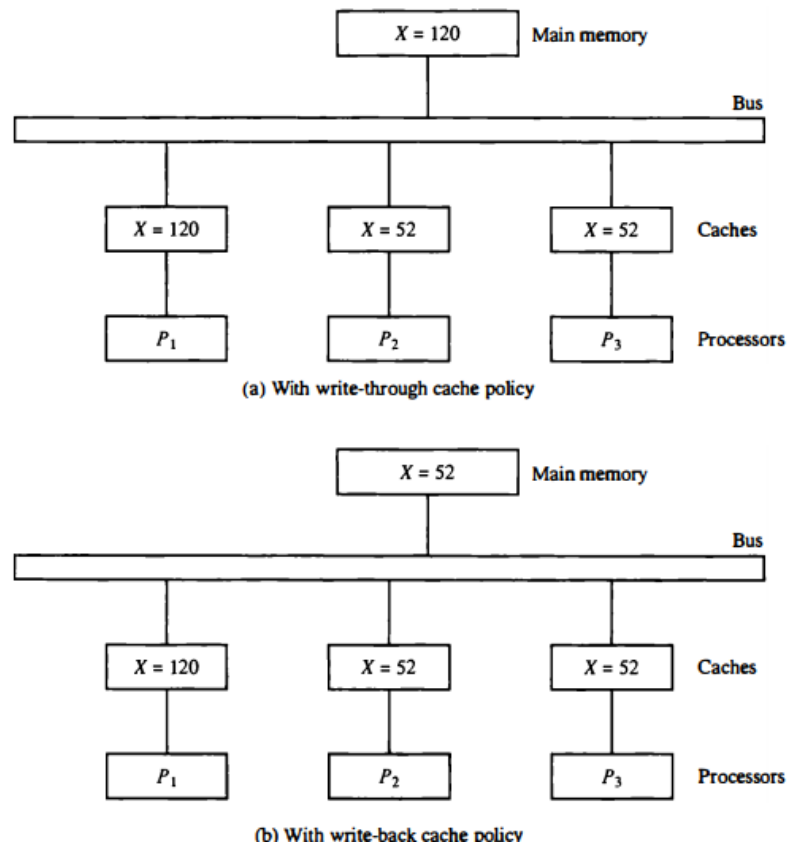


Figure 12 Cache configuration after a load on X.

- **As a consequence, it is also** copied into the private caches of the three processors. For simplicity, we assume that X contains the value of 52. The load on X to the three processors results in consistent copies in the caches and main memory.

- **If one of the processors** performs a store to X, the copies of X in the caches become inconsistent. A load by the other processors will not return the latest value. Depending on the memory update policy used in the cache, the main memory may also be inconsistent with respect to the cache.
- **This is shown in Fig. 13.** A store to X (of the value of 120) into the cache of processor P₁ updates memory to the new value in a write-through policy.

Figure 13 Cache configuration after a store to X by processor P₁.



- **A write-through policy maintains** consistency between memory and the originating cache, but the other two caches are inconsistent since they still hold the old value. In a write-back policy, main memory is not updated at the time of the store.
- **The copies in the other two caches** and main memory are inconsistent. Memory is updated eventually when the modified data in the cache are copied back into memory.
- **Another configuration** that may cause consistency problems is a direct memory access (DMA) activity in conjunction with an IOP connected to the system bus. In the case of input, the DMA may modify locations in main memory that also reside in cache without updating the cache.
- **During a DMA output**, memory locations may be read before they are updated from the cache when using a write-back policy. I/O-based memory incoherence can be overcome by making the IOP a participant in the cache coherent solution that is adopted in the system.

37. Solutions to the Cache Coherence Problem

- **A simple scheme** is to disallow private caches for each processor and have a shared cache memory associated with main memory. Every data access is made to the shared cache.
- **This method violates** the principle of closeness of CPU to cache and increases the average memory access time. In effect, this scheme solves the problem by avoiding it.
- **For performance** considerations it is desirable to attach a private cache to each processor. One scheme that has been used allows only non shared and read-only data to be stored in caches.
- **Such items are called cachable.** Shared writable data are non cachable. The compiler must tag data as either cachable or non cachable, and the system hardware makes sure that only cachable data are stored in caches.
- **The non cachable data remain in main memory.** This method restricts overhead the type of data stored in caches and introduces extra software that may degrade performance.
- **A scheme that allows** writable data to exist in at least one cache is a method that employs a centralized global table in its compiler. The status of memory blocks is stored in the central global table.
- **Each block is identified as read-only (RO) or read and write (RW).** All caches can have copies of blocks identified as RO.
- **Only one cache can have a copy of an RW block.** Thus if the data affected are updated in the cache with an RW block, the other caches are not because they do not have a copy of this block.
- **The cache coherence problem** can be solved by means of a combination of software and hardware or by means of hardware-only schemes. The two methods mentioned previously use software based procedures that require the ability to tag information in order to disable caching of shared writable data.
- **Hardware-only** solutions are handled by the hardware automatically and have the advantage of higher speed and program transparency.
- **In the hardware** solution, the cache controller is specially designed to allow it to monitor all bus requests from CPUs and IOPs.
- **All caches** attached to the bus constantly monitor the network for possible write operations.
- **Depending on the method** used, they must then either update or invalidate their own cache copies when a match is detected. The bus controller that monitors this action is referred to as a snoopy cache controller. This is basically a hardware unit designed to maintain a bus-watching mechanism over all the caches attached to the bus.
- **Various schemes** have been proposed to solve the cache coherence problem by means of snoopy cache protocol
- **The simplest method is to adopt a** write through policy and use the following procedure. All the snoopy controllers watch the bus for memory store operations.
- **When a word in a cache** is updated by writing into it, the corresponding location in main memory is also updated.

- **The local snoopy controllers** in all other caches check their memory to determine if they have a copy of the word that has been overwritten.
- **If a copy exists** In a remote cache, that location is marked invalid.
- **Because all** caches snoop on all bus writes, whenever a word is written, the net effect is to update it In the original cache and main memory and remove It from all other caches.
- **If at some future time a processor accesses the invalid** Item from its cache, the response is equivalent to a cache miss, and the updated Item is transferred from main memory. In this way, inconsistent versions are prevented.